Zilog

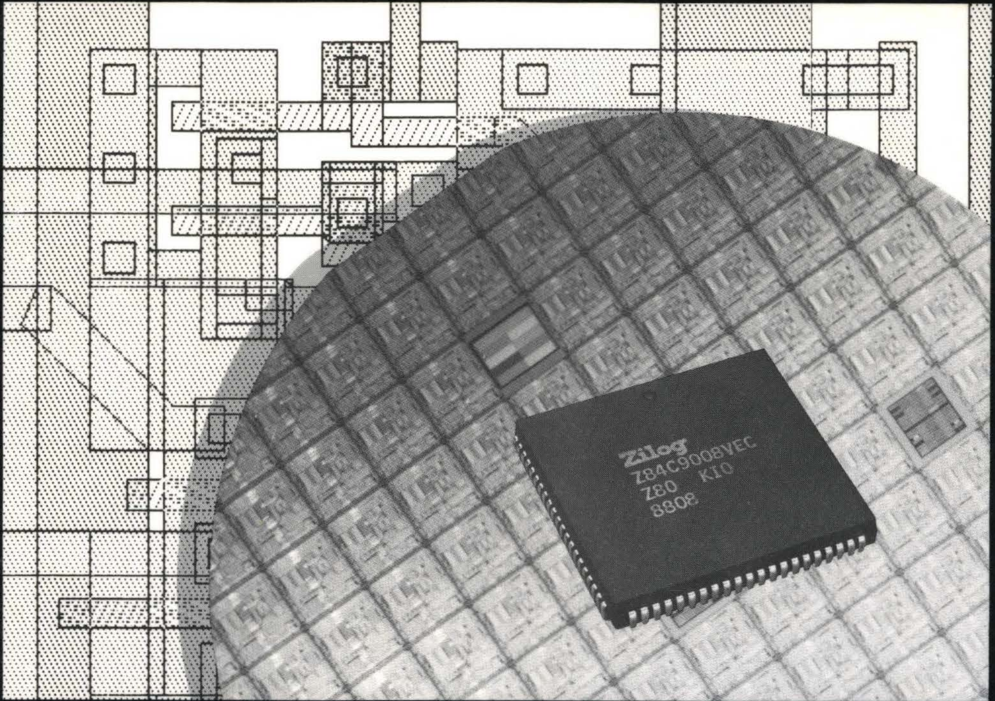
Z80 FAMILY DATA BOOK

JANUARY 1989

Zilog

Z80 FAMILY DATA BOOK

JANUARY 1989



Zilog

Product Specification

January 1989

Z80® Family Data Book

Z80® Family Data Book
Table of Contents

Zilog

January 1989

Zilog Z80 Family
An Industry Standard 8-Bit Architecture with 16-Bit Migration Path

Zilog remains an industry leader, thanks to continuing innovation in integrated design and new superintegration technology. At Zilog, innovation means using proven, sophisticated mainframe and minicomputer concepts and translating them into the latest LSI technologies. Integration means more than designing an ever-greater number of functions onto a single chip.

This guide to the Z80 family of state-of-the-art microprocessors and intelligent peripheral controllers demonstrates Zilog's continued support for the Z80 microprocessor and the other members of the Z80 product family - family first introduced in 1976 that continues to enjoy growing customer support while family chips are upgraded to newer and ever-higher standards.

The **Z8400/84C00 CPU Central Processing Unit** rapidly established itself as the most sophisticated, most powerful, and most versatile 8-bit microprocessor in the world.

In addition to being source-code compatible with the 8080A microprocessor, the Z80 offers more instructions than the 8080A (158 vs 78) and numerous other features that simplify hardware requirements and reduce programming effort while increasing throughput. The dual-register set of the Z80 CPU allows high-speed context switching and more efficient interrupt processing. Two index registers give additional memory addressing flexibility and simplify the task of programming. Interfacing to dynamic memory is simplified by on-chip, programmable refresh logic. Block moves plus string and bit manipulation instructions reduce programming effort, program size, and execution time. The CMOS versions of the family retain all the functions of the standard NMOS components while providing dramatic power savings and increased reliability.

The four traditional functions of a microcomputer system (parallel I/O, serial I/O, counting/timing, and direct memory access) are easily implemented by the following well-proven family of Z80 peripheral devices: Z80 PIO, Z80 SIO, Z80 DART, Z80 CTC, and Z80 DMA.

The easily programmed, dual channel **Z8420/84C20 Z80 PIO Parallel Input/Output Controller** offers two 8-bit I/O ports with individual handshake and pattern recognition logic. Both I/O ports operate in either a byte or a bit mode. In addition, this device can be programmed to generate interrupts for various status conditions.

All common data communications protocols, asynchronous as well as synchronous, are remarkably well handled by the **Z8440/84C40 Z80 SIO Serial Input/Output Controller**. This dual-channel receiver/transmitter device offers on-chip parity and CRC generation/checking. FIFO buffering and flag and frame detection generation logic are also offered.

If asynchronous-only applications are required, the cost effective Z8470 Z80 Dart Dual Asynchronous Receiver/Transmitter can be used in place of the Z80 SIO. The Z80 Dart offers all Z80 SIO asynchronous features in two channels.

Timing and event-counting functions are the forte of the **Z8430/84C30 Z80 Counter/Timer Controller**. The CTC provides four counters, each with individually programmable prescalers. The CTC is a convenient source of programmable clock rates for the SIO.

With the **Z8410/84C10 Z80 DMA Direct Memory Access Controller**, data can be transferred directly between any two ports (typically, I/O and memory). The DMA transfers, searches, or search/transfers data in Byte-by-Byte, Burst, or Continuous modes.

The **Z180** microprocessor integrates an enhanced Z80 CPU and many of the functions traditionally assigned to peripheral circuits onto a single chip. The Z180 provides an easy software upgrade path. Old Z80 designs can be converted to the Z180 with essentially no loss in software investment. New designs will benefit from the processor's low cost, powerful instruction set, real low power consumption, and high level of integration. Recent advances in CMOS technology and chip-packing densities inspired the Z180. The processor is essentially a Z80 core (with a few added instructions), and a number of on-board peripherals. The most important of these is Memory Management Unit (MMU), which translates 16 bit addresses to 20 bits. Although this translation gives programs access to 1 MB of memory, the code uses only 16 bit (64K) addresses. Z80 software compatibility is thus completely maintained. Other Z180 peripherals include two 16 bit counter/timers, programmable refresh and wait state generation, a pair of DMA controllers, and three serial ports with on-chip baud rate generation.

The **Z80280** brings 16-bit CPU and sophisticated features required by complex, high performance applications to the Z80 architecture. Z280 maintains complete object code compatibility with the Z80. One of the unique features of the Z280 is its bus size. By strapping a single pin on the chip, the designer can select 8 or 16 bit bus widths. Thus to use existing designs, an 8-bit Z80 compatible bus can be used. Higher performance systems can be designed using the Z280's 16 bit mode, in which all memory references use true 16 bit accesses. A single processor can be used in both medium and high performance products, without changing the software. The Z280 includes a Memory Management Unit (MMU), which gives the processor access to 16 MB of memory. Other features of the Z280 include on-chip instruction and cache memory, 3-stage pipeline, dual operating modes, four channel DMA Controller, three 16 bit counter/timers, programmable refresh and wait state generation, and a serial port with on-chip baud rate generation.

Zilog's Superintegration Strategy

In these days, when success in the semiconductor business requires that each competitor have a significant "edge", Zilog finds itself in a unique position. Customers have welcomed ASIC products specifically tailored for their particular needs. As these chips grow larger, however, economics dictates that there be a higher degree of organization in the architecture of the chips than exists with conventional gate arrays and standard cells; the microprocessor "core" offers the ideal, may be even unique solution to this requirement.

There are only three or four microprocessor families which are sufficiently well known in the industry to be recognized by a wide user base. The Zilog Z80 microprocessor family is well positioned in that ranking.

Zilog is making its microprocessor cores and peripheral cells available to the industry in a concept we have called "Superintegration". In this concept, popular one-chip arrangements of the Zilog cores and cells are offered to our customers as "standard products" which may be tailored in software to meet particular customer needs.

The Zilog Superintegration concept offers three major benefits:

- The economics, quality and reliability of standard structures.
- The familiar Zilog product architectures and operating systems.
- User customization via software.

One of the first "Superintegration" products of the Z80 family is the **Z84C90 Killer I/O (KIO)**. This chip combines the features of Z84C30 (CTC), Z84C4x (SIO), Z84C20 (PIO), a byte-wide bit programmable I/O port, and a crystal oscillator on a single chip. The **Z84C01** is the Z80 CPU with a built-in clock generator/controller. The **Z84C80 GLU** is a collection of various circuits required to interface Z80 with memory and I/O peripheral devices. The features included on this chip include; the crystal oscillator, dynamic memory interface controller, static memory interface, memory and chip I/O selects, watch-dog timer, five types of wait state generators, and Z8500 peripheral interface.

Zilog

ADVANCED INFORMATION
Product Specification

Z8400/Z84C00 NMOS/CMOS Z80® CPU Central Processing Unit

January 1989

FEATURES

- The extensive instruction set contains 158 instructions, including the 8080A instruction set as a subset.
- Single 5 volt power supply.
- NMOS version for low cost high performance solutions, CMOS version for high performance low power designs.
- NMOS Z0840004 - 4 MHz, Z0840006 - 6.17 MHz, Z0840008 - 8 MHz.

  CMOS Z84C0004 - DC to 4 MHz, Z84C0006 - DC to 6.17 MHz, Z84C0008 - DC to 8 MHz, Z84C0010 - DC to 10 MHz.
- 6 MHz version can be operated at 6.144 MHz clock.
- The Z80 microprocessors and associated family of peripherals can be linked by a vectored interrupt system. This system can be daisy-chained to allow implementation of a priority interrupt scheme.
- Duplicate set of both general-purpose and flag registers.
- Two sixteen bit index registers.
- Three modes of maskable interrupts:
  Mode 0—8080A similar;
  Mode 1—Non-Z80 environment, location 38H;
  Mode 2—Z80 family peripherals, vectored interrupts.
- On-chip dynamic memory refresh counter.

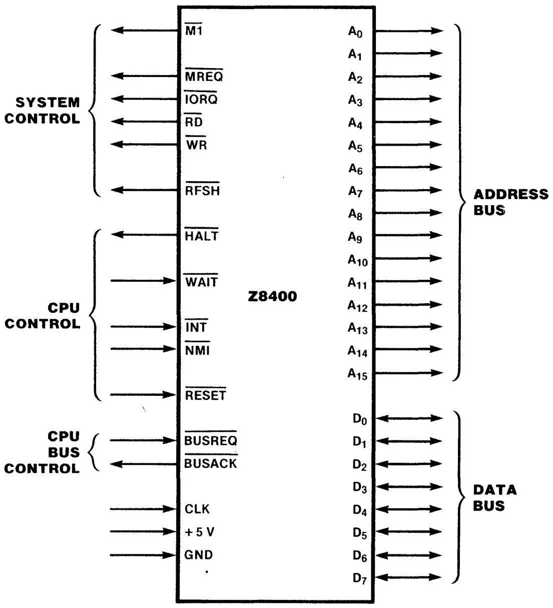


Figure 1. Pin Functions

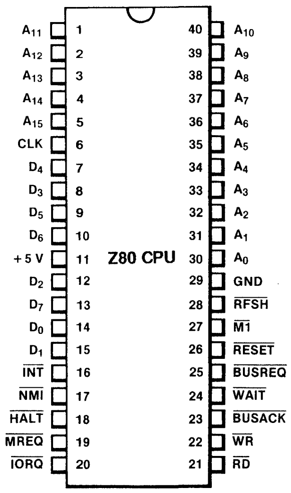
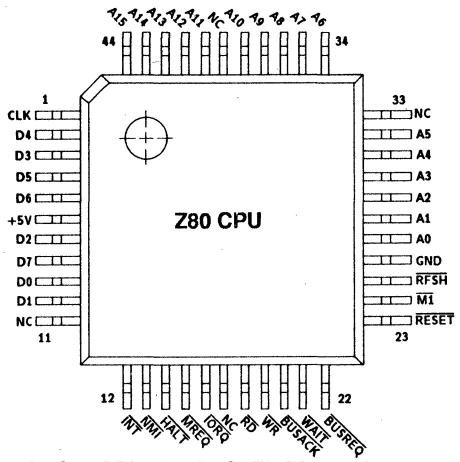


Figure 2. 40-pin Dual-In-Line (DIP), Pin Assignments



Z80 CPU
A15 A14 A13 A12 A11 NC A10 A9 A8 A7 A6
44
34
1
CLK
D4
D3
D5
D6
+5V
D2
D7
D0
D1
NC
11
33
NC
A5
A4
A3
A2
A1
A0
GND
RFSH
M1
RESET
23
12
22
INT NMI HALT MREQ IORQ NC RD WR BUSACK WAIT BUSREQ

44 pin Quad Flat Pack (QFP), Pin Assignments (Only available for 84C00)

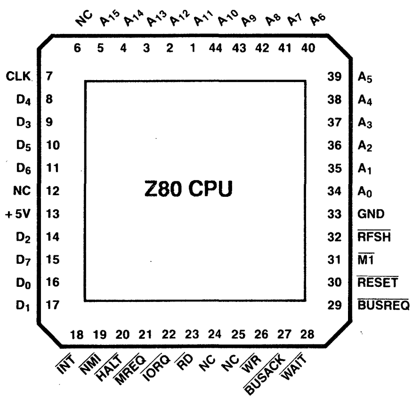


Figure 2b. 44-Pin Chip Carrier Pin Assignments

GENERAL DESCRIPTION

The CPUs are fourth-generation enhanced microprocessors with exceptional computational power. They offer higher system throughput and more efficient memory utilization than comparable second- and third-generation microprocessors. The internal registers contain 208 bits of read/write memory that are accessible to the programmer. These registers include two sets of six general-purpose registers which may be used individually as either 8-bit registers or as 16-bit register pairs. In addition, there are two sets of accumulator and flag registers. A group of "Exchange" instructions makes either set of main or alternate registers accessible to the programmer. The alternate set allows operation in foreground-background mode or it may be reserved for very fast interrupt response.

The CPU also contains a Stack Pointer, Program Counter, two index registers, a Refresh register (counter), and an Interrupt register. The CPU is easy to incorporate into a system since it requires only a single +5V power source. All output signals are fully decoded and timed to control standard memory or peripheral circuits; the CPU is supported by an extensive family of peripheral controllers. The internal block diagram (Figure 3) shows the primary functions of the processors. Subsequent text provides more detail on the I/O controller family, registers, instruction set, interrupts and daisy chaining, and CPU timing.

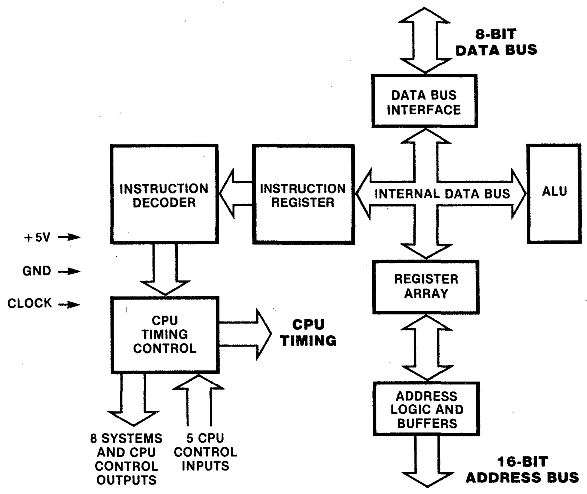


Figure 3. Z80C CPU Block Diagram

CPU REGISTERS

Figure 4 shows three groups of registers within the CPU. The first group consists of duplicate sets of 8-bit registers: a principal set and an alternate set [designated by ′ (prime), e.g., A′]. Both sets consist of the Accumulator register, the Flag register, and six general-purpose registers. Transfer of data between these duplicate sets of registers is accomplished by use of "Exchange" instructions. The result is faster response to interrupts and easy, efficient implementation of such versatile programming techniques as background-foreground data processing. The second set of registers consists of six registers with assigned functions. These are the I (Interrupt register), the R (Refresh register), the IX and IY (Index registers) the SP (Stack Pointer), and the PC (Program Counter). Tne third group consists of two interrupt status flip-flops, plus an additional pair of flip-flops which assists in identifying the interrupt mode at any particular time. Table 1 provides further information on these registers.

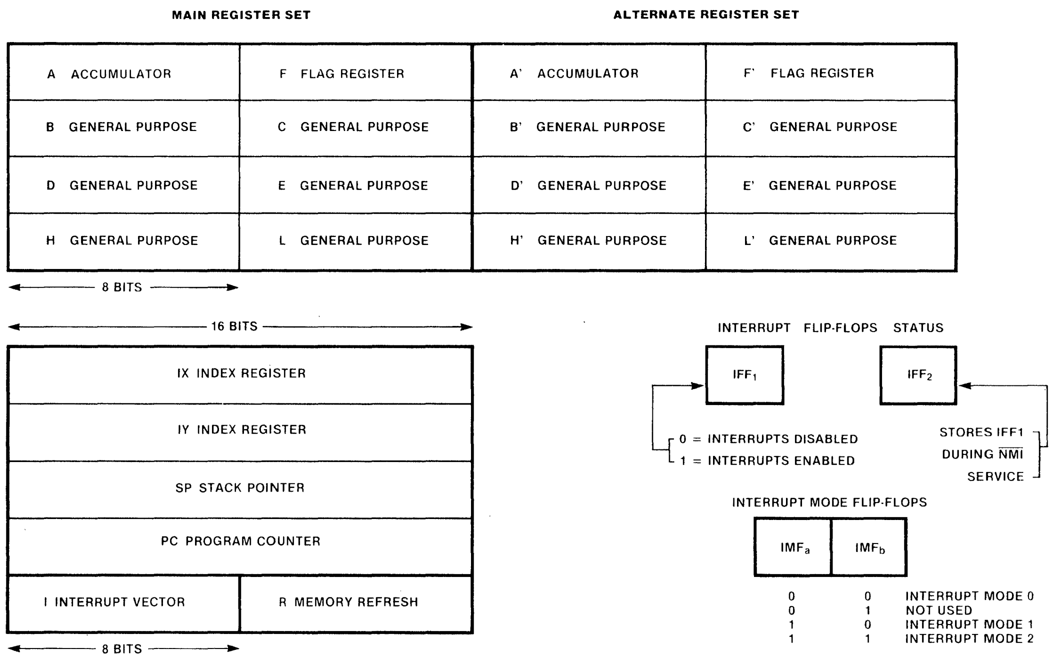


Figure 4. CPU Registers

INTERRUPTS: GENERAL OPERATION

The CPU accepts two interrupt input signals: $\overline{NMI}$ and $\overline{INT}$. The $\overline{NMI}$ is a non-maskable interrupt and has the highest priority. $\overline{INT}$ is a lower priority interrupt and it requires that interrupts be enabled in software in order to operate. $\overline{INT}$ can be connected to multiple peripheral devices in a wired-OR configuration.

The Z80 has a single response mode for interrupt service on the non-maskable interrupt. The maskable interrupt, $\overline{INT}$, has three programmable response modes available. These are:

- Mode 0 — similar to the 8080 microprocessor.
- Mode 1 — Peripheral Interrupt service, for use with non-8080/Z80 systems.
- **Mode 2 - a vectored interrupt scheme, usually daisy-chained, for use with the Z80 Family and compatible peripheral devices.**

The CPU services interrupts by sampling the $\overline{NMI}$ and $\overline{INT}$ signals at the rising edge of the last clock of an instruction. Further interrupt service processing depends upon the type of interrupt that was detected. Details on interrupt responses are shown in the CPU Timing Section.

Non-Maskable Interrupt ($\overline{NMI}$). The nonmaskable interrupt cannot be disabled by program control and therefore will be accepted at all times by the CPU. $\overline{NMI}$ is usually reserved for servicing only the highest priority type interrupts, such as that for orderly shutdown after power

Table 1. Z80C CPU Registers

Register		Size (Bits)	Remarks
A, A′	Accumulator	8	Stores an operand or the results of an operation.
F, F′	Flags	8	See Instruction Set.
B, B′	General Purpose	8	Can be used separately or as a 16-bit register with C.
C, C′	General Purpose	8	Can be used separately or as a 16-bit register with C.
D, D′	General Purpose	8	Can be used separately or as a 16-bit register with E.
E, E′	General Purpose	8	Can be used separately or as a 16-bit register with E.
H, H′	General Purpose	8	Can be used separately or as a 16-bit register with L.
L, L′	General Purpose	8	Can be used separately or as a 16-bit register with L.
			Note: The (B,C), (D,E), and (H,L) sets are combined as follows:
			B — High byte C — Low byte
			D — High byte E — Low byte
			H — High byte L — Low byte
I	Interrupt Register	8	Stores upper eight bits of memory address for vectored interrupt processing.
R	Refresh Register	8	Provides user-transparent dynamic memory refresh. Automatically incremented and placed on the address bus during each instruction fetch cycle.
IX	Index Register	16	Used for indexed addressing.
IY	Index Register	16	Used for indexed addressing
SP	Stack Pointer	16	Holds address of the top of the stack. See Push or Pop in instruction set.
PC	Program Counter	16	Holds address of next instruction.
IFF_1-IFF_2	Interrupt Enable	Flip-Flops	Set or reset to indicate interrupt status (see Figure 4).
IMFa-IMFb	Interrupt Mode	Flip-Flops	Reflect Interrupt mode (see Figure 4).

failure has been detected. After recognition of the $\overline{\text{NMI}}$ signal (providing $\overline{\text{BUSREQ}}$ is not active), the CPU jumps to restart location 0066H. Normally, software starting at this address contains the interrupt service routine.

Maskable Interrupt ($\overline{\text{INT}}$). Regardless of the interrupt mode set by the user, the CPU response to a maskable interrupt input follows a common timing cycle. After the interrupt has been detected by the CPU (provided that interrupts are enabled and $\overline{\text{BUSREQ}}$ is not active) a special interrupt processing cycle begins. This is a special fetch ($\overline{\text{M1}}$) cycle in which $\overline{\text{IORQ}}$ becomes active rather than $\overline{\text{MREQ}}$, as in a normal $\overline{\text{M1}}$ cycle. In addition, this special $\overline{\text{M1}}$ cycle is automatically extended by two $\overline{\text{WAIT}}$ states, to allow for the time required to acknowledge the interrupt request.

Mode 0 Interrupt Operation. This mode is similar to the 8080 microprocessor interrupt service procedures. The interrupting device places an instruction on the data bus. This is normally a Restart instruction, which will initiate a call **to the selected one of eight restart locations in page zero of memory. Unlike the 8080, the Z80 CPU responds to the Call instruction with only one interrupt acknowledge cycle followed by two memory read cycles.**

Mode 1 Interrupt Operation. Mode 1 operation is very similar to that for the $\overline{\text{NMI}}$. The principal difference is that the Mode 1 interrupt has only one restart location, 0038H.

Mode 2 Interrupt Operation. This interrupt mode has been designed to most effectively utilize the capabilities of the Z80 microprocessor and its associated peripheral family.The interrupting peripheral device selects the starting address of the interrupt service routine. It does this by placing an 8-bit vector on the data bus during the interrupt acknowledge cycle. The CPU forms a pointer using this byte as the lower 8 bits and the contents of the I register as the upper 8 bits. This points to an entry in a table of addresses for interrupt service routines. The CPU then jumps to the routine at that

address. This flexibility in selecting the interrupt service routine address allows the peripheral device to use several different types of service routines. These routines may be located at any available location in memory. Since the interrupting device supplies the low-order byte of the 2-byte vector, bit 0 (A_0) must be a zero.

Interrupt Enable/Disable Operation. Two flip-flops, IFF_1 and IFF_2, referred to in the register description, are used to signal the CPU interrupt status. Operation of the two flip-flops is described in Table 2. For more details, refer to the *Z80 CPU Technical Manual* (03-0029-01) and *Z80 Assembly Language Programming Manual* (03-0002-01).

Table 2. State of Flip-Flops

Action	IFF_1	IFF_2	Comments
CPU Reset	0	0	Maskable interrupt $\overline{INT}$ disabled
DI instruction execution	0	0	Maskable interrupt $\overline{INT}$ disabled
EI instruction execution	1	1	Maskable interrupt $\overline{INT}$ enabled
LD A,I instruction execution	•	•	$IFF_2 \rightarrow$ Parity flag
LD A,R instruction execution	•	•	$IFF_2 \rightarrow$ Parity flag
Accept $\overline{NMI}$	0	•	Maskable interrupt $\overline{INT}$ disabled
RETN instruction execution	IFF_2	•	$IFF_2 \rightarrow IFF_1$ at completion of an $\overline{NMI}$ service routine.

INSTRUCTION SET

The microprocessor has one of the most powerful and versatile instruction sets available in any 8-bit microprocessor. It includes such unique operations as a block move for fast, efficient data transfers within memory, or between memory and I/O. It also allows operations on any bit in any location in memory.

The following is a summary of the instruction set which shows the assembly language mnemonic, the operation, the flag status, and gives comments on each instruction. For an explanation of flag notations and symbols for mnemonic tables, see the Symbolic Notations section which follows these tables. The *Z80 CPU Technical Manual* (03-0029-01), the *Programmer's Reference Guide* (03-0012-03), and *Assembly Language Programming Manual* (03-0002-01) contain significantly more details for programming use.

The instructions are divided into the following categories:

- □ 8-bit loads
- □ 16-bit loads
- □ Exchanges, block transfers, and searches
- □ 8-bit arithmetic and logic operations
- □ General-purpose arithmetic and CPU control
- □ 16-bit arithmetic operations
- □ Rotates and shifts
- □ Bit set, reset, and test operations
- □ Jumps
- □ Calls, returns, and restarts
- □ Input and output operations

A variety of addressing modes are implemented to permit efficient and fast data transfer between various registers, memory locations, and input/output devices. These addressing modes include:

- □ Immediate
- □ Immediate extended
- □ Modified page zero
- □ Relative
- □ Extended
- □ Indexed
- □ Register
- □ Register indirect
- □ Implied
- □ Bit

PIN DESCRIPTIONS

A_0-A_{15}. *Address Bus* (output, active High, 3-state). A_0-A_{15} form a 16-bit address bus. The Address Bus provides the address for memory data bus exchanges (up to 64K bytes) and for I/O device exchanges.

$\overline{\mathrm{BUSACK}}$. *Bus Acknowledge* (output, active Low). Bus Acknowledge indicates to the requesting device that the CPU address bus, data bus, and control signals $\overline{\mathrm{MREQ}}$, $\overline{\mathrm{IORQ}}$, $\overline{\mathrm{RD}}$, and $\overline{\mathrm{WR}}$ have entered their high-impedance states. The external circuitry can now control these lines.

$\overline{\mathrm{BUSREQ}}$. *Bus Request* (input, active Low). Bus Request has a higher priority than $\overline{\mathrm{NMI}}$ and is always recognized at the end of the current machine cycle. $\overline{\mathrm{BUSREQ}}$ forces the CPU address bus, data bus, and control signals $\overline{\mathrm{MREQ}}$, $\overline{\mathrm{IORQ}}$, RD, and WR to go to a high-impedance state so that other devices can control these lines. $\overline{\mathrm{BUSREQ}}$ is normally wired-OR and requires an external pullup for these applications. Extended $\overline{\mathrm{BUSREQ}}$ periods due to extensive DMA operations can prevent the CPU from properly refreshing dynamic RAMs.

D_0-D_7. *Data Bus* (input/output, active High, 3-state). D_0-D_7 constitute an 8-bit bidirectional data bus, used for data exchanges with memory and I/O.

$\overline{\mathrm{HALT}}$. *Halt State* (output, active Low). $\overline{\mathrm{HALT}}$ indicates that the CPU has executed a Halt instruction and is awaiting either a nonmaskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU executes NOPs to maintain memory refresh.

$\overline{\mathrm{INT}}$. *Interrupt Request* (input, active Low). Interrupt Request is generated by I/O devices. The CPU honors a request at the end of the current instruction if the internal software-controlled interrupt enable flip-flop (IFF) is enabled. $\overline{\mathrm{INT}}$ is normally wired-OR and requires an external pullup for these applications.

$\overline{\mathrm{IORQ}}$. *Input/Output Request* (output, active Low, 3-state). $\overline{\mathrm{IORQ}}$ indicates that the lower half of the address bus holds a valid I/O address for an I/O read or write operation. $\overline{\mathrm{IORQ}}$ is also generated concurrently with $\overline{\mathrm{M1}}$ during an interrupt acknowledge cycle to indicate that an interrupt response vector can be placed on the data bus.

$\overline{\mathrm{M1}}$. *Machine Cycle One* (output, active Low). $\overline{\mathrm{M1}}$, together with $\overline{\mathrm{MREQ}}$, indicates that the current machine cycle is the opcode fetch cycle of an instruction execution. $\overline{\mathrm{M1}}$, together with $\overline{\mathrm{IORQ}}$, indicates an interrupt acknowledge cycle.

$\overline{\mathrm{MREQ}}$. *Memory Request* (output, active Low, 3-state). $\overline{\mathrm{MREQ}}$ indicates that the address bus holds a valid address for a memory read or memory write operation.

$\overline{\mathrm{NMI}}$. *Non-Maskable Interrupt* (input, negative edge-triggered). $\overline{\mathrm{NMI}}$ has a higher priority than $\overline{\mathrm{INT}}$. $\overline{\mathrm{NMI}}$ is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop, and automatically forces the CPU to restart at location 0066H.

$\overline{\mathrm{RD}}$. *Read* (output, active Low, 3-state). $\overline{\mathrm{RD}}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.

$\overline{\mathrm{RESET}}$. *Reset* (input, active Low). $\overline{\mathrm{RESET}}$ initializes the CPU as follows: it resets the interrupt enable flip-flop, clears the PC and Registers I and R, and sets the interrupt status to Mode 0. During reset time, the address and data bus go to a high-impedance state, and all control output signals go to the inactive state. Note that $\overline{\mathrm{RESET}}$ must be active for a minimum of three full clock cycles before the reset operation is complete.

$\overline{\mathrm{RFSH}}$. *Refresh* (output, active Low). $\overline{\mathrm{RFSH}}$, together with $\overline{\mathrm{MREQ}}$, indicates that the lower seven bits of the system's address bus can be used as a refresh address to the system's dynamic memories.

$\overline{\mathrm{WAIT}}$. *Wait* (input, active Low). $\overline{\mathrm{WAIT}}$ indicates to the CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter a Wait state as long as this signal is active. Extended $\overline{\mathrm{WAIT}}$ periods can prevent the CPU from properly refreshing dynamic memory.

$\overline{\mathrm{WR}}$. *Write* (output, active Low, 3-state). $\overline{\mathrm{WR}}$ indicates that the CPU data bus holds valid data to be stored at the addressed memory or I/O location.

CPU TIMING

The Z80 CPU executes instructions by proceeding through a specific sequence of operations:

- Memory read or write
- I/O device read or write
- Interrupt acknowledge

The basic clock period is referred to as a T time or cycle, and three or more T cycles make up a machine cycle (M1, M2 or M3 for instance). Machine cycles can be extended either by the CPU automatically inserting one or more Wait states or by the insertion of one or more Wait states by the user.

Instruction Opcode Fetch. The CPU places the contents of the Program Counter (PC) on the address bus at the start of the cycle (Figure 5). Approximately one-half clock cycle later, $\overline{MREQ}$ goes active. When active, $\overline{RD}$ indicates that the memory data can be enabled onto the CPU data bus.

The CPU samples the $\overline{WAIT}$ input with the falling edge of clock state T_2. During clock states T_3 and T_4 of an $\overline{M1}$ cycle, dynamic RAM refresh can occur while the CPU starts decoding and executing the instruction. When the Refresh Control signal becomes active, refreshing of dynamic memory can take place.

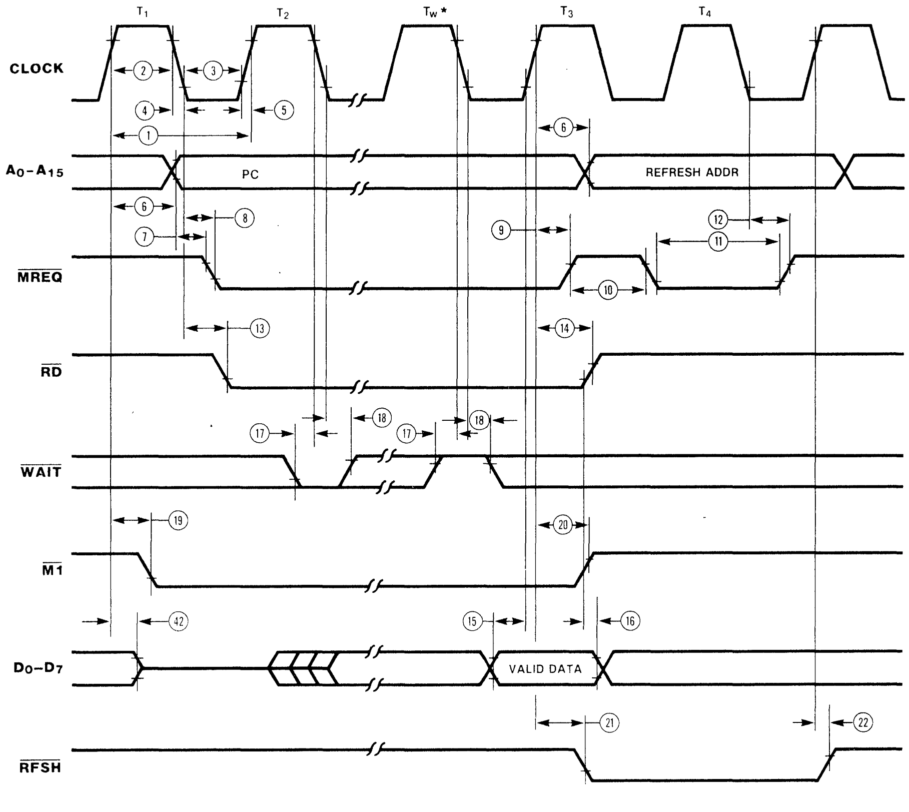


Figure 5. Instruction Opcode Fetch

Memory Read or Write Cycles. Figure 6 shows the timing of memory read or write cycles other than an opcode fetch ($\overline{M1}$) cycle. The $\overline{MREQ}$ and $\overline{RD}$ signals function exactly as in the fetch cycle. In a memory write cycle, $\overline{MREQ}$ also becomes active when the address bus is stable. The $\overline{WR}$ line is active when the data bus is stable, so that it can be used directly as an R/$\overline{W}$ pulse to most semiconductor memories.

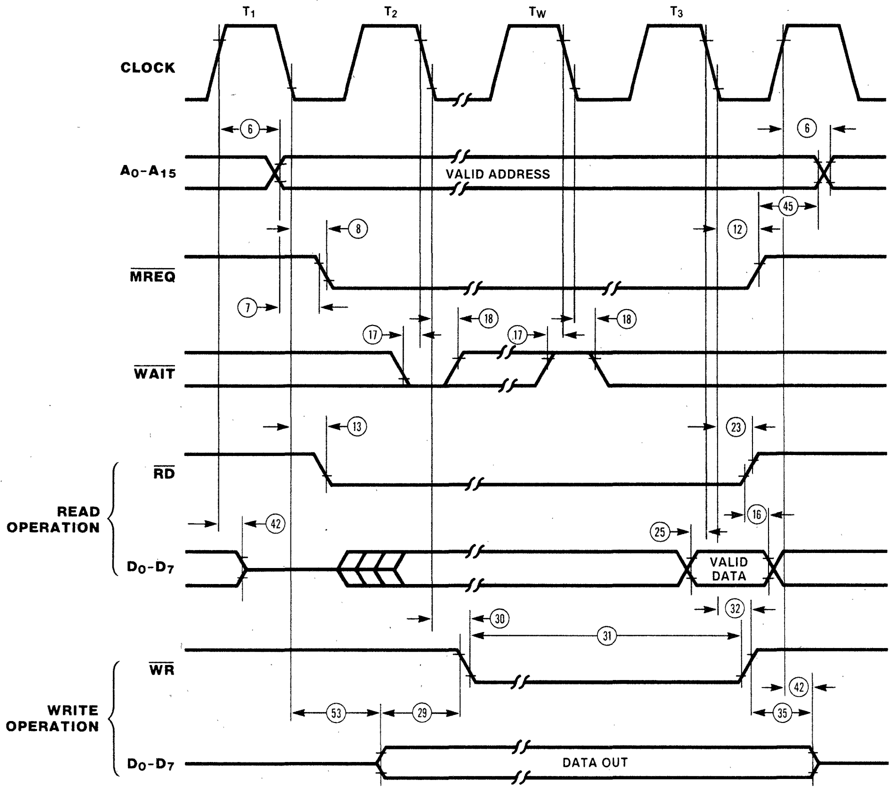
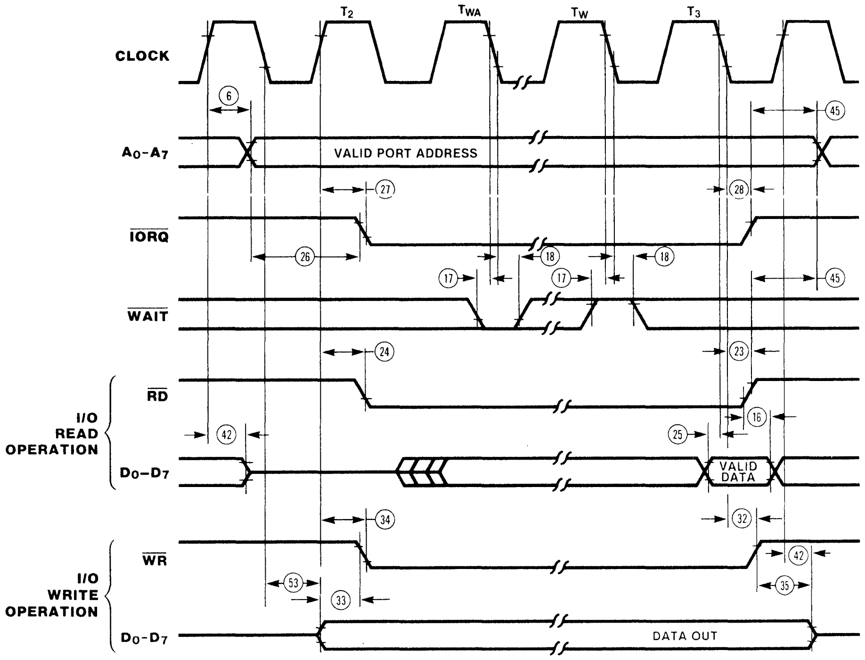


Figure 6. Memory Read or Write Cycles

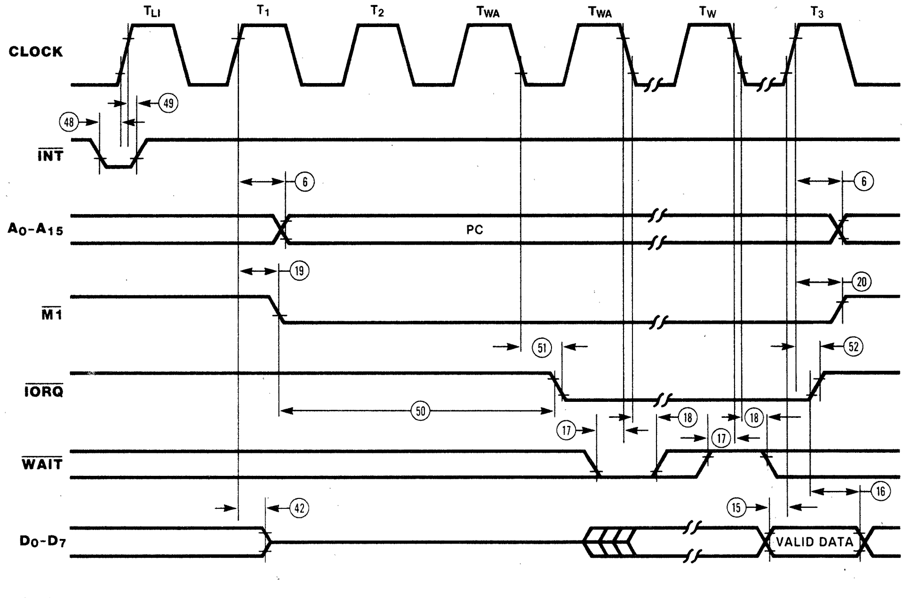
Input or Output Cycles. Figure 7 shows the timing for an I/O read or I/O write operation. During I/O operations, the CPU automatically inserts a single Wait state (T_{WA}). This extra Wait state allows sufficient time for an I/O port to decode the address from the port address lines.



T_{WA} = One wait cycle automatically inserted by CPU.

Figure 7. Input or Output Cycles

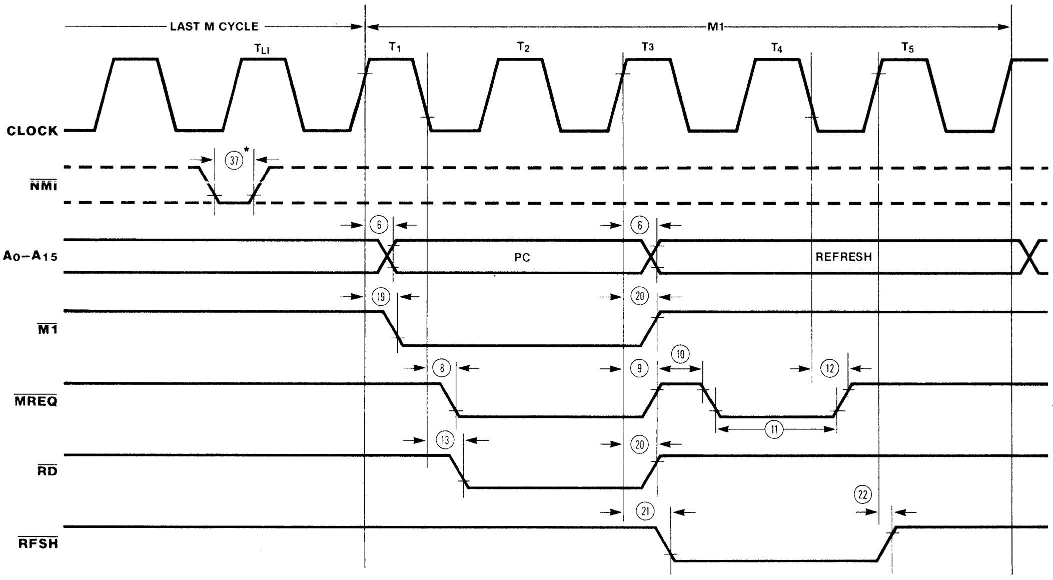
Interrupt Request/Acknowledge Cycle. The CPU samples the interrupt signal with the rising edge of the last clock cycle at the end of any instruction (Figure 8). When an interrupt is accepted, a special $\overline{M1}$ cycle is generated. During this $\overline{M1}$ cycle, $\overline{IORQ}$ becomes active (instead of $\overline{MREQ}$) to indicate that the interrupting device can place an 8-bit vector on the data bus. The CPU automatically adds two Wait states to this cycle.



NOTES: 1) T_{LI} = Last state of any instruction cycle.
2) T_{WA} = Wait cycle automatically inserted by CPU.

Figure 8. Interrupt Request/Acknowledge Cycle

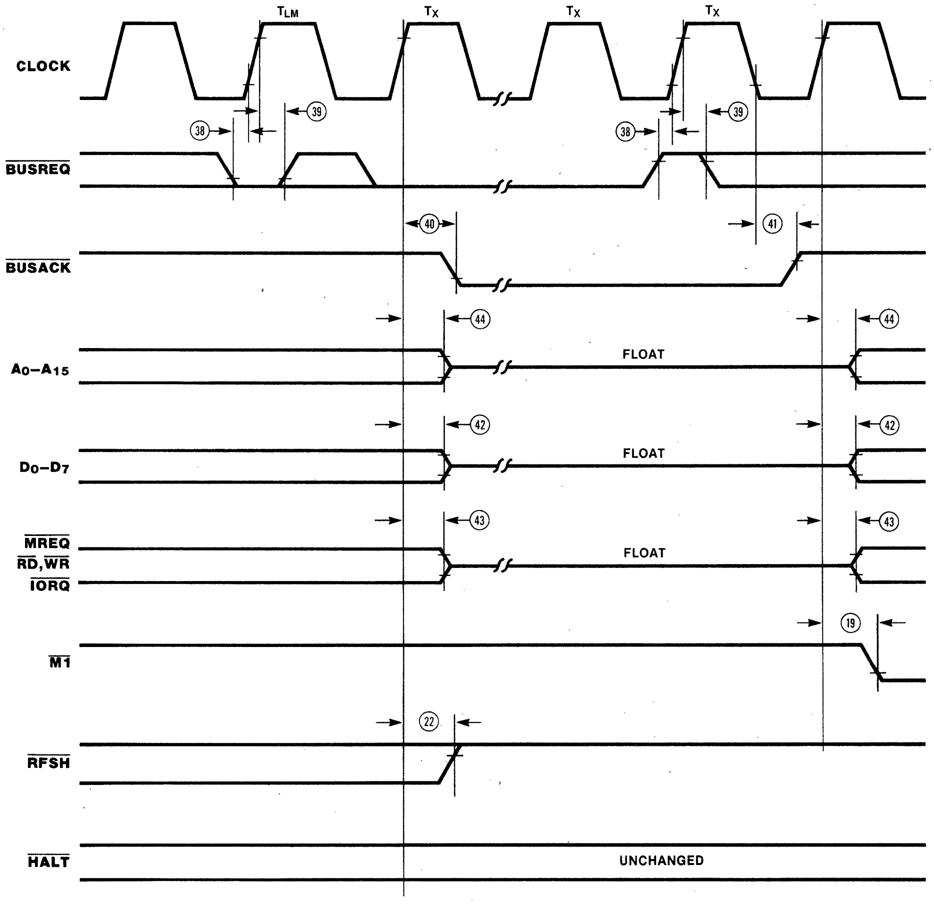
Non-Maskable Interrupt Request Cycle. $\overline{\text{NMI}}$ is sampled at the same time as the maskable interrupt input $\overline{\text{INT}}$ but has higher priority and cannot be disabled under software control. The subsequent timing is similar to that of a normal memory read operation except that data put on the bus by the memory is ignored. The CPU instead executes a restart (RST) operation and jumps to the $\overline{\text{NMI}}$ service routine located at address 0066H (Figure 9).



*Although $\overline{\text{NMI}}$ is an asynchronous input, to guarantee its being recognized on the following machine cycle, $\overline{\text{NMI}}$'s falling edge must occur no later than the rising edge of the clock cycle preceding the last state of any instruction cycle (T_{LI}).

Figure 9. Non-Maskable Interrupt Request Operation

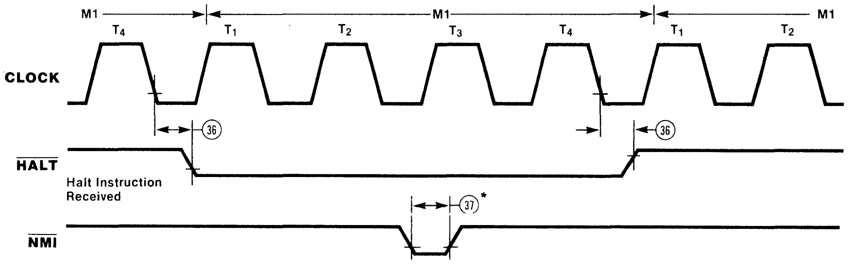
Bus Request/Acknowledge Cycle. The CPU samples $\overline{\text{BUSREQ}}$ with the rising edge of the last clock period of any machine cycle (Figure 10). If $\overline{\text{BUSREQ}}$ is active, the CPU sets its address, data, and $\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ lines to a high-impedance state with the rising edge of the next clock pulse. At that time, any external device can take control of these lines, usually to transfer data between memory and I/O devices.



NOTES: 1) T_{LM} = Last state of any M cycle.
2) T_X = An arbitrary clock cycle used by requesting device.

Figure 10. BUS Request/Acknowledge Cycle

Halt Acknowledge Cycle. When the CPU receives a $\overline{\text{HALT}}$ instruction, it executes NOP states until either an $\overline{\text{INT}}$ or $\overline{\text{NMI}}$ input is received. When in the Halt state, the $\overline{\text{HALT}}$ output is active and remains so until an interrupt is received (Figure 11). $\overline{\text{INT}}$ will also force a Halt exit.



*Although $\overline{\text{NMI}}$ is an asynchronous input, to guarantee its being recognized on the following machine cycle, $\overline{\text{NMI}}$'s falling edge must occur no later than the rising edge of the clock cycle preceding the last state of any instruction cycle (T_{LI}).

Figure 11. Halt Acknowledge

Reset Cycle. $\overline{\text{RESET}}$ must be active for at least three clock cycles for the CPU to properly accept it. As long as $\overline{\text{RESET}}$ remains active, the address and data buses float, and the control outputs are inactive. Once $\overline{\text{RESET}}$ goes inactive, two internal T cycles are consumed before the CPU resumes normal processing operation. $\overline{\text{RESET}}$ clears the PC register, so the first opcode fetch will be to location 0000H (Figure 12).

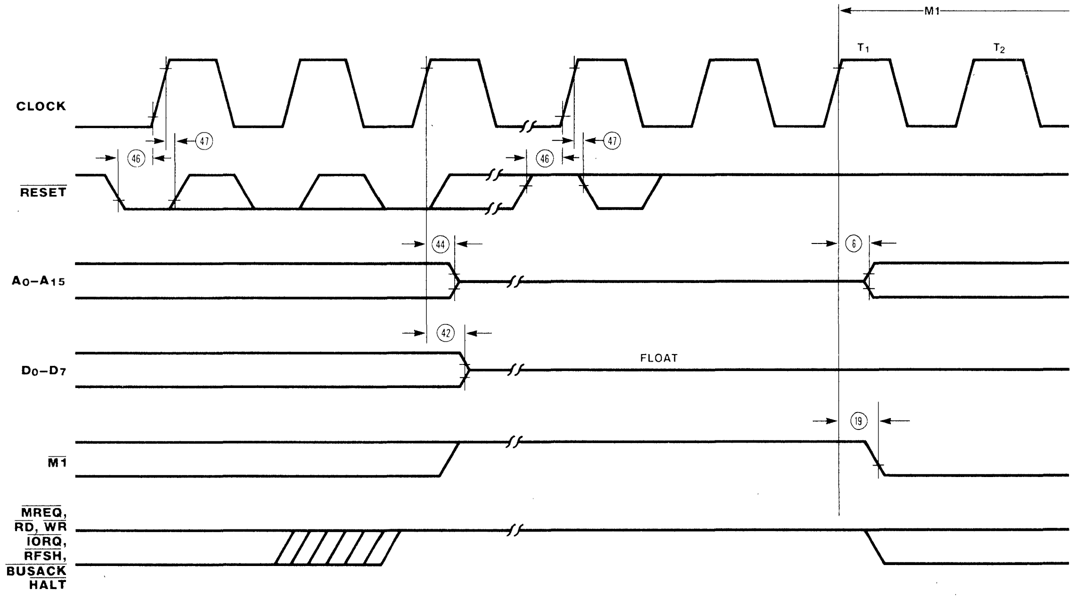


Figure 12. Reset Cycle

Power-Down mode of operation (Only applies to CMOS Z80 CPU).

CMOS Z80 CPU supports Power-Down mode of operation. **This mode is also referred to as the "standby mode", and supply current for the CPU goes down as low as 10 uA (Where specified as Icc_2).**

Power-Down Acknowledge Cycle. When the clock input to the CPU is stopped at either a High or Low level, the CPU stops its operation and maintains all registers and control signals. However, I_{CC2} (standby supply current) is guaranteed only when the system clock is stopped at a Low level during T_4 of the machine cycle following the execution of the HALT instruction. The timing diagram for the power-down function, when implemented with the HALT **instruction, is shown in Figure 13.**

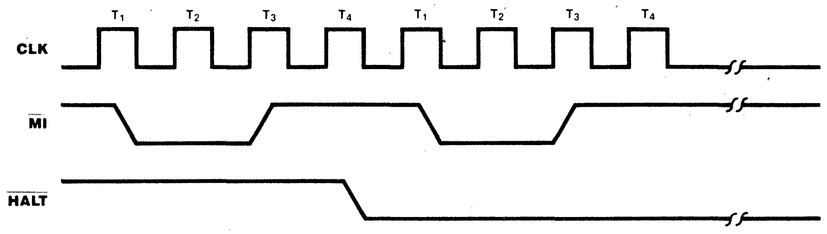


Figure 13. Power-Down Acknowledge

Power-Down Release Cycle. The system clock must be supplied to the CPU to release the power-down state. When the system clock is supplied to the CLK input, the CPU restarts operations from the point at which the power-down state was implemented.

The timing diagrams for the release from power-down mode **are shown in Figure 14.**

NOTES:

1) When the external oscillator has been stopped to enter the power-down state, some warm-up time may be required to obtain a stable clock for the release.
2) When the HALT instruction is executed to enter the power-down state, the CPU will also enter the Halt state. An interrupt signal (either $\overline{\text{NMI}}$ or $\overline{\text{INT}}$) or a $\overline{\text{RESET}}$ signal must be applied to the CPU after the system clock is supplied in order to release the power-down state.

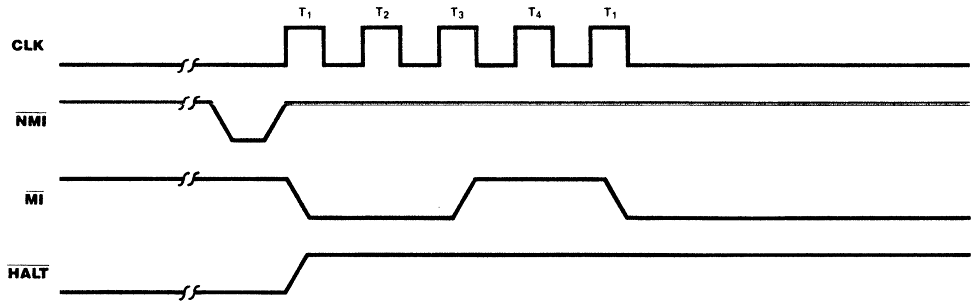


Figure 14a.

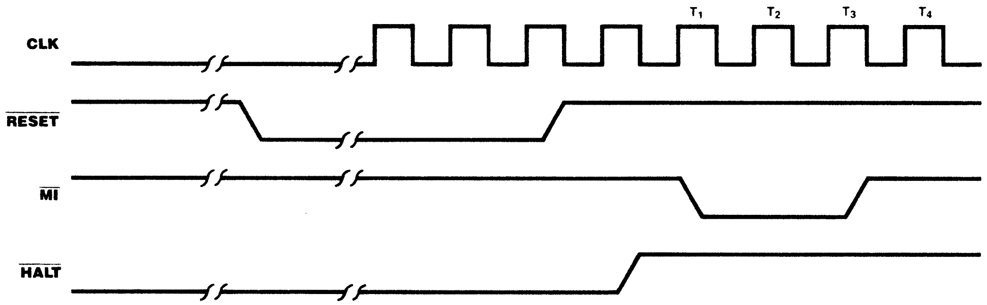


Figure 14b.

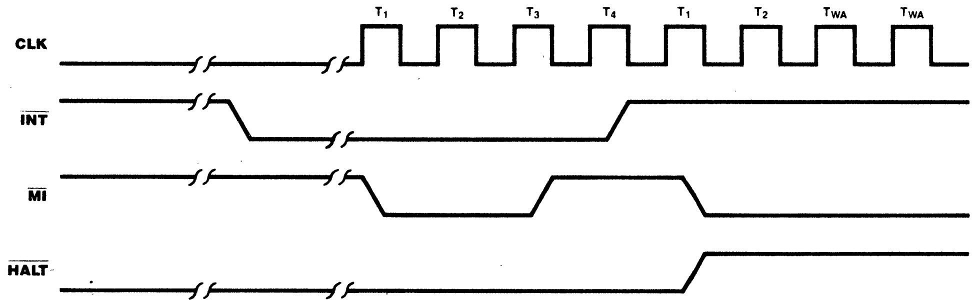


Figure 14c.

Figure 13. Power-Down Release

ABSOLUTE MAXIMUM RATINGS

Voltage on V_{CC} with respect to V_{SS} . . . . −0.3V to +7V
Voltages on all inputs with respect to V_{SS} . . . . −0.3V to V_{CC} + 0.3V
Operating Ambient Temperature . . . . .See Ordering Information
Storage Temperature . . . . −65°C to +150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

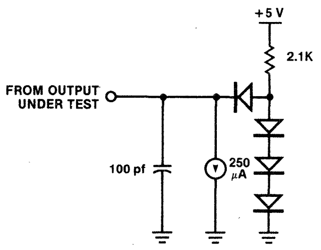
The DC Characteristics and capacitance sections below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND (0V). Positive current flows into the referenced pin.

Available operating temperature ranges are:

- **S = 0°C to +70°C**
  Voltage Supply Range:
  NMOS: +4.75V ≧ V_{cc} ≧ +5.25V
  CMOS: +4.50V ≧ V_{cc} ≧ 5.50V
- **E = -40°C to 100°C, +4.50V ≧ V_{cc} > +5.50V**

All ac parameters assume a load capacitance of 100 pf. Add 10 ns delay for each 50 pf increase in load up to a maximum of 200 pf for the data bus and 100 pf for address and control lines. AC timing measurements are referenced to 1.5 volts (except for clock, which is referenced to the 10% and 90% points).

The Ordering Information section lists temperature ranges and product numbers. Package drawings are in the Package Information section. Refer to the Literature List for additional documentation.



DC CHARACTERISTICS (Z84C00/CMOS Z80 CPU)

Symbol	Parameter	Min	Max	Unit	Condition
V_{ILC}	Clock Input Low Voltage	−0.3	0.45	V	
V_{IHC}	Clock Input High Voltage	V_{CC} − .6	V_{CC} + .3	V	
V_{IL}	Input Low Voltage	−0.3	0.8	V	
V_{IH}	Input High Voltage	2.2	V_{CC}	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.0 mA
V_{OH_1}	Output High Voltage	2.4		V	I_{OH} = −1.6 mA
V_{OH_2}	Output High Voltage	V_{CC} − 0.8		V	I_{OH} = −250 µA
I_{CC_1}	Power Supply Current **4 MHz**		**20**	**mA**	V_{CC} = 5V
	6 MHz		**30**	**mA**	V_{IH} = V_{CC} − 0.2V
	8 MHz		**40**	**mA**	V_{IL} = 0.2V
	10 MHz		**50**	**mA**	
I_{CC_2}	Standby Supply Current		10	µA	V_{CC} = 5V CLK = (0) V_{IH} = V_{CC} − 0.2V V_{IL} = 0.2V
I_{LI}	Input Leakage Current		10	µA	V_{IN} = 0.4 to V_{CC}
I_{LO}	3-State Output Leakage Current in Float	−10	10[2]	µA	V_{OUT} = 0.4 to V_{CC}

1. Measurements made with outputs floating.
2. A_{15}-A_0, D_7-D_0, $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, and $\overline{WR}$.
3. I_{CC_2} standby supply current is guaranteed only when the supplied clock is stopped at a low level during T_4 of the machine cycle immediately following the execution of a HALT instruction.

CAPACITANCE

Symbol	Parameter	Min	Max	Unit
C_{CLOCK}	Clock Capacitance		**10**	pf
C_{IN}	Input Capacitance		5	pf
C_{OUT}	Output Capacitance		15	pf

T_A = 25°C, f = 1 MHz.
Unmeasured pins returned to ground.

AC CHARACTERISTICS† (Z84C00/CMOS Z80 CPU)

			Z84C0004		Z84C0006		Z84C0008		Z84C0010	
Number	Symbol	Parameter	Min	Max	Min	Max	Min	Max	Min	Max
1	TcC	Clock Cycle Time	250*	DC	162*	DC	125	DC	100	DC
2	TwCh	Clock Pulse Width (High)	110	DC	65	DC	55	DC	42	DC
3	TwCl	Clock Pulse Width (Low)	110	DC	65	DC	55	DC	42	DC
4	TfC	Clock Fall Time		30		20		10		10
5	TrC	Clock Rise Time		30		20		10		10
6	TdCr(A)	Clock ↑ to Address Valid Delay		110		90		80		65
7	TdA(MREQf)	Address Valid to $\overline{\text{MREQ}}$ ↓ Delay	65*		35*		20*		22*	
8	TdCf(MREQf)	Clock ↓ to $\overline{\text{MREQ}}$ ↓ Delay		85		70		60		55
9	TdCr(MREQr)	Clock ↑ to $\overline{\text{MREQ}}$ ↑ Delay		85		70		60		55
10	TwMREQh	$\overline{\text{MREQ}}$ Pulse Width (High)	110*††		65*††		45*††		32*††	
11	TwMREQl	$\overline{\text{MREQ}}$ Pulse Width (Low)	220*††		135*††		100*††		75*††	
12	TdCf(MREQr)	Clock ↓ to $\overline{\text{MREQ}}$ ↑ Delay		85		70		60		55
13	TdCf(RDf)	Clock ↓ to $\overline{\text{RD}}$ ↓ Delay		95		80		70		65
14	TdCr(RDr)	Clock ↑ to $\overline{\text{RD}}$ ↑ Delay		85		70		60		55
15	TsD(Cr)	Data Setup Time to Clock ↑	35		30		30		25	
16	ThD(RDr)	Data Hold Time to $\overline{\text{RD}}$ ↑		0		0		0		0
17	TsWAIT(Cf)	$\overline{\text{WAIT}}$ Setup Time to Clock ↓	70		60		50		25	
18	ThWAIT (Cf)	$\overline{\text{WAIT}}$ Hold Time after Clock ↓	10		10		10		10	
19	TdCr(M1f)	Clock ↑ to $\overline{\text{M1}}$ ↓ Delay		100		80		70		65
20	TdCr(M1r)	Clock ↑ to $\overline{\text{M1}}$ ↑ Delay		100		80		70		65
21	TdCr(RRSHf)	Clock ↑ to $\overline{\text{RFSH}}$ ↓ Delay		130		110		95		80
22	TdCr(RFSHr)	Clock ↑ to $\overline{\text{RFSH}}$ ↑ Delay		120		100		85		80
23	TdCf(RDr)	Clock ↓ to $\overline{\text{RD}}$ ↑ Delay		85		70		60		55
24	TdCr(RDf)	Clock ↑ to $\overline{\text{RD}}$ ↓ Delay		85		70		60		55
25	TsD(Cf)	Data Setup to Clock ↓ during M_2, M_3, M_4, or M_5 Cycles	50		40		30		25*	
26	TdA(IORQf)	Address Stable prior to $\overline{\text{IORQ}}$ ↓	180*		110*		75*		70*	
27	TdCr(IORQf)	Clock ↑ to $\overline{\text{IORQ}}$ ↓ Delay		75		65		55		50
28	TdCf(IORQr)	Clock ↓ to $\overline{\text{IORQ}}$ ↑ Delay		85		70		60		55
29	TdD(WRf)Mw	Data Stable prior to $\overline{\text{WR}}$ ↓	80*		25*		5*		40*	

*For clock periods other than the minimums shown, calculate parameters using the table on the following page. Calculated values above assumed TrC = TfC = 20 ns.

†Units in nanoseconds (ns).

††**For loading ≥ 50 pf. Decrease width by 10 ns for each additional 50 pf.**

AC CHARACTERISTICS† (Z84C00/CMOS Z80 CPU; Continued)

			Z84C0004		Z84C0006		Z84C0008		Z84C0010	
Number	Symbol	General Parameter	Min	Max	Min	Max	Min	Max	Min	Max
30	TdCf(WRf)	Clock ↓ to $\overline{\text{WR}}$ ↓ Delay		80		70		60		55
31	TwWR	$\overline{\text{WR}}$ Pulse Width	220*		135*		100*		75*	
32	TdCf(WRr)	Clock ↓ to $\overline{\text{WR}}$ ↑ Delay		80		70		60		55
33	TdD(WRf)IO	Data Stable prior to $\overline{\text{WR}}$ ↓	−10*		−55*		−55*		-8*	
34	TdCr(WRf)	Clock ↑ to $\overline{\text{WR}}$ ↓ Delay		65		60		55		50
35	TdWRr(D)	Data Stable from $\overline{\text{WR}}$ ↑	60*		30*		15*		12*	
36	TdCf(HALT)	Clock ↓ to $\overline{\text{HALT}}$ ↑ or ↓		300		260		225		90
37	TwNMI	$\overline{\text{NMI}}$ Pulse Width	80		70		60		60	
38	TsBUSREQ(Cr)	$\overline{\text{BUSREQ}}$ Setup Time to Clock ↑	50		50		40		30	
39	ThBUSREQ(Cr)	$\overline{\text{BUSREQ}}$ Hold Time after Clock ↑	10		10		10		10	
40	TdCr(BUSACKf)	Clock ↑ to $\overline{\text{BUSACK}}$ ↓ Delay		100		90		80		75
41	TdCf(BUSACKr)	Clock ↓ to $\overline{\text{BUSACK}}$ ↑ Delay		100		90		80		75
42	TdCr(Dz)	Clock ↑ to Data Float Delay		90		80		70		65
43	TdCr(CTz)	Clock ↑ to Control Outputs Float Delay($\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$)		80		70		60		60
44	TdCr(Az)	Clock ↑ to Address Float Delay		90		80		70		65
45	TdCTr(A)	$\overline{\text{MREQ}}$ ↑, $\overline{\text{IORQ}}$ ↑, $\overline{\text{RD}}$ ↑, and $\overline{\text{WR}}$ ↑ to Address Hold Time	80*		35*		20*		32*	
46	TsRESET(Cr)	$\overline{\text{RESET}}$ to Clock ↑ Setup Time	60		60		45		40	
47	ThRESET(Cr)	$\overline{\text{RESET}}$ to Clock ↑ Hold Time	10		10		10		10	
48	TsINTf(Cr)	$\overline{\text{INT}}$ to Clock ↑ Setup Time	80		70		55		50	
49	ThINTr(Cr)	$\overline{\text{INT}}$ to Clock ↑ Hold Time	10		10		10		10	
50	TdM1f(IORQf)	$\overline{\text{M1}}$ ↓ to $\overline{\text{IORQ}}$ ↓ Delay	565*		365*		270*		222*	
51	TdCf(IORQf)	Clock ↓ to $\overline{\text{IORQ}}$ ↓ Delay		85		70		60		55
52	TdCf(IORQr)	Clock ↑ to $\overline{\text{IORQ}}$ ↑ Delay		85		70		60		55
53	TdCf(D)	Clock ↓ to Data Valid Delay		150		130		115		110

*For clock periods other than the minimums shown, calculate parameters using the following table. Calculated values above assumed TrC = TfC = 20 ns.
†Units in nanoseconds (ns).

FOOTNOTES TO AC CHARACTERISTICS

Number	Symbol	General Parameter	Z84C0004	Z84C0006	Z84C0008	Z84C0010
1	TcC	TwCh + TwCl + TrC + TfC				
7	TdA(MREQf)	TwCh + TfC	−65	−50	−45	-45
10	TwMREQh	TwCh + TfC	−20	−20	−20	-20
11	TwMREQl	TcC	−30	−30	−25	-25
26	TdA(IORQf)	TcC	−70	−55	−50	-50
29	TdD(WRf)	TcC	−170	−140	−120	-60
31	TwWR	TcC	−30	−30	−25	-25
33	TdD(WRf)	TwCl + TrC	−140	−140	−120	-60
35	TdWRr(D)	TwCl + TrC	−70	−55	−50	-40
45	TdCTr(A)	TwCl + TrC	−50	−50	−45	-30
50	TdM1f(IORQf)	2TcC + TwCh + TfC	−65	−50	−45	-30

AC Test Conditions: V_{IH} = 2.0 V V_{OH} = 1.5 V V_{IHC} = V_{CC} −0.6 V FLOAT = ±0.5 V
V_{IL} = 0.8 V V_{OL} = 1.5 V V_{ILC} = 0.45 V

DC CHARACTERISTICS (Z8400/NMOS Z80 CPU)

All parameters are tested unless otherwise noted.

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	−0.3	0.45	V	
V_{IHC}	Clock Input High Voltage	$V_{CC}-.6$	$V_{CC}+.3$	V	
V_{IL}	Input Low Voltage	−0.3	0.8	V	
V_{IH}	Input High Voltage	2.0[1]	V_{CC}	V	
V_{OL}	Output Low Voltage		0.4	V	$I_{OL} = 2.0$ mA
V_{OH}	Output High Voltage	2.4[1]		V	$I_{OH} = -250\ \mu A$
I_{CC}	Power Supply Current		200	mA	Note 3
I_{LI}	Input Leakage Current		10	μA	$V_{IN} = 0$ to V_{CC}
I_{LO}	3-State Output Leakage Current in Float	−10	10[2]	μA	$V_{OUT} = 0.4$ to V_{CC}

1. For military grade parts, refer to the Z80 Military Electrical Specification.
2. A_{15}-A_0, D_7-D_0, $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, and $\overline{WR}$.
3. Measurements made with outputs floating.

CAPACITANCE

Guaranteed by design and characterization.

Symbol	Parameter	Min	Max	Unit
C_{CLOCK}	Clock Capacitance		35	pf
C_{IN}	Input Capacitance		5	pf
C_{OUT}	Output Capacitance		15	pf

NOTES:
$T_A = 25°C$, f = 1 MHz.
Unmeasured pins returned to ground.

AC CHARACTERISTICS† (Z8400/NMOS Z80 CPU)

			Z0840004		Z0840006		Z0840008	
Number	Symbol	Parameter	Min	Max	Min	Max	Min	Max
1	TcC	Clock Cycle Time	250*		162*		125*	
2	TwCh	Clock Pulse Width (High)	110	2000	65	2000	55	2000
3	TwCl	Clock Pulse Width (Low)	110	2000	65	2000	55	2000
4	TfC	Clock Fall Time		30		20		10
5	TrC	Clock Rise Time		30		20		10
6	TdCr(A)	Clock ↑ to Address Valid Delay		110		90		80
7	TdA(MREQf)	Address Valid to $\overline{\text{MREQ}}$ ↓ Delay	65*		35*		20*	
8	TdCf(MREQf)	Clock ↓ to $\overline{\text{MREQ}}$ ↓ Delay		85		70		60
9	TdCr(MREQr)	Clock ↑ to $\overline{\text{MREQ}}$ ↑ Delay		85		70		60
10	TwMREQh	$\overline{\text{MREQ}}$ Pulse Width (High)	110*††		65*††		45*††	
11	TwMREQl	$\overline{\text{MREQ}}$ Pulse Width (Low)	220*††		135*††		100*††	
12	TdCf(MREQr)	Clock ↓ to $\overline{\text{MREQ}}$ ↑ Delay		85		70		60
13	TdCf(RDf)	Clock ↓ to $\overline{\text{RD}}$ ↓ Delay		95		80		70
14	TdCr(RDr)	Clock ↑ to $\overline{\text{RD}}$ ↑ Delay		85		70		60
15	TsD(Cr)	Data Setup Time to Clock ↑	35		30		30	
16	ThD(RDr)	Data Hold Time to $\overline{\text{RD}}$ ↑		0		0		0
17	TsWAIT(Cf)	$\overline{\text{WAIT}}$ Setup Time to Clock ↓	70		60		50	
18	ThWAIT(Cf)	$\overline{\text{WAIT}}$ Hold Time after Clock ↓		0		0		0
19	TdCr(M1f)	Clock ↑ to $\overline{\text{M1}}$ ↓ Delay		100		80		70
20	TdCr(M1r)	Clock ↑ to $\overline{\text{M1}}$ ↑ Delay		100		80		70
21	TdCr(RFSHf)	Clock ↑ to $\overline{\text{RFSH}}$ ↓ Delay		130		110		95
22	TdCr(RFSHr)	Clock ↑ to $\overline{\text{RFSH}}$ ↑ Delay		120		100		85
23	TdCf(RDr)	Clock ↓ to $\overline{\text{RD}}$ ↑ Delay		85		70		60
24	TdCr(RDf)	Clock ↑ to $\overline{\text{RD}}$ ↓ Delay		85		70		60
25	TsD(Cf)	Data Setup to Clock ↓ during M_2, M_3, M_4, or M_5 Cycles	50		40		30	
26	TdA(IORQf)	Address Stable prior to $\overline{\text{IORQ}}$ ↓	180*		110*		75*	
27	TdCr(IORQf)	Clock ↑ to $\overline{\text{IORQ}}$ ↓ Delay		75		65		55
28	TdCf(IORQr)	Clock ↓ to $\overline{\text{IORQ}}$ ↑ Delay		85		70		60
29	TdD(WRf)	Data Stable prior to $\overline{\text{WR}}$ ↓	80*		25*		5*	
30	TdCf(WRf)	Clock ↓ to $\overline{\text{WR}}$ ↓ Delay		80		70		60
31	TwWR	$\overline{\text{WR}}$ Pulse Width	220*		135*		100*	
32	TdCf(WRr)	Clock ↓ to $\overline{\text{WR}}$ ↑ Delay		80		70		60
33	TdD(WRf)	Data Stable prior to $\overline{\text{WR}}$ ↓	−10*		−55*		55*	
34	TdCr(WRf)	Clock ↑ to $\overline{\text{WR}}$ ↓ Delay		65		60		55
35	TdWRr(D)	Data Stable from $\overline{\text{WR}}$ ↑	60*		30*		15*	
36	TdCf(HALT)	Clock ↓ to $\overline{\text{HALT}}$ ↑ or ↓		300		260		225
37	TwNMI	$\overline{\text{NMI}}$ Pulse Width	80		70		60*	
38	TsBUSREQ(Cr)	$\overline{\text{BUSREQ}}$ Setup Time to Clock ↑	50		50		40	

*For clock periods other than the minimums shown, calculate parameters using the table on the following page. Calculated values above assumed TrC = TfC = 20 ns.

†Units in nanoseconds (ns).

†† For loading $\geq$ 50 pf., Decrease width by 10 ns for each additional 50 pf.

AC CHARACTERISTICS† (Z8400/NMOS Z80 CPU; Continued)

			Z0840004		Z0840006		Z0840008	
Number	Symbol	Parameter	Min	Max	Min	Max	Min	Max
39	ThBUSREQ(Cr)	$\overline{\text{BUSREQ}}$ Hold Time after Clock ↑	0		0		0	
40	TdCr(BUSACKf)	Clock ↑ to $\overline{\text{BUSACK}}$ ↓ Delay		100		90		80
41	TdCf(BUSACKr)	Clock ↓ to $\overline{\text{BUSACK}}$ ↑ Delay		100		90		80
42	TdCr(Dz)	Clock ↑ to Data Float Delay		90		80		70
43	TdCr(CTz)	Clock ↑ to Control Outputs Float Delay ($\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$)		80		70		60
44	TdCr(Az)	Clock ↑ to Address Float Delay		90		80		70
45	TdCTr(A)	$\overline{\text{MREQ}}$ ↑, $\overline{\text{IORQ}}$ ↑, $\overline{\text{RD}}$ ↑, and $\overline{\text{WR}}$ ↑ to Address Hold Time	80*		35*		20*	
46	TsRESET(Cr)	$\overline{\text{RESET}}$ to Clock ↑ Setup Time	60		60		45	
47	ThRESET(Cr)	$\overline{\text{RESET}}$ to Clock ↑ Hold Time		0		0		0
48	TsINTf(Cr)	$\overline{\text{INT}}$ to Clock ↑ Setup Time	80		70		55	
49	ThINTr(Cr)	$\overline{\text{INT}}$ to Clock ↑ Hold Time		0		0		0
50	TdM1f(IORQf)	$\overline{\text{M1}}$ ↓ to $\overline{\text{IORQ}}$ ↓ Delay	565*		365*		270*	
51	TdCf(IORQf)	Clock ↓ to $\overline{\text{IORQ}}$ ↓ Delay		85		70		60
52	TdCf(IORQr)	Clock ↑ $\overline{\text{IORQ}}$ ↑ Delay		85		70		60
53	TdCf(D)	Clock ↓ to Data Valid Delay		150		130		115

*For clock periods other than the minimums shown, calculate parameters using the following table. Calculated values above assumed TrC = TfC = 20 ns.
†Units in nanoseconds (ns).

FOOTNOTES TO AC CHARACTERISTICS

Number	Symbol	General Parameter	Z0840004	Z0840006	Z0840008
1	TcC	TwCh + TwCl + TrC + TfC			
7	TdA(MREQf)	TwCh + TfC	− 65	−50	−45
10	TwMREQh	TwCh + TfC	− 20	−20	−20
11	TwMREQl	TcC	− 30	−30	−25
26	TdA(IORQf)	TcC	− 70	−55	−50
29	TdD(WRf)	TcC	− 170	−140	−120
31	TwWR	TcC	− 30	−30	−25
33	TdD(WRf)	TwCl + TrC	− 140	−140	−120
35	TdWRr(D)	TwCl + TrC	− 70	−55	−50
45	TdCTr(A)	TwCl + TrC	− 50	−50	−45
50	TdM1f(IORQf)	2TcC + TwCh + TfC	− 65	−50	−45

AC Test Conditions:

V_{IH} = 2.0 V	V_{OH} = 1.5 V
V_{IL} = 0.8 V	V_{OL} = 1.5 V
V_{IHC} = V_{CC} −0.6 V	FLOAT = ±0.5 V
V_{ILC} = 0.45 V	

Zilog

PRELIMINARY
Product Specification

January 1989

Z84C01 Z80® CPU with Clock Generator/Controller

FEATURES:

- Commands compatible with the Zilog Z80 MPU
- Low power consumption
  - 40mA Typ (5V, 10 MHz under RUN mode)
  - 2mA Typ (5V, 10 MHz under IDLE1 mode)
  - 10mA Typ (5V, 10 MHz under IDLE2 mode)
  - .5μ A Typ (5V under STOP mode)
- DC to 10 MHz operation (at 5V±10%)
- Single 5V power supply (at 5V±10%)
- Operating temperature (0° C to 70° C)
- On-chip clock generator
- In the HALT state, the following 4 modes are selectable:
  - RUN mode
  - IDLE 1 mode
  - IDLE 2 mode
  - STOP mode
- Powerful set of 158 instructions
- Powerful interrupt function
  - Non-maskable interrupt terminal ($\overline{NMI}$)
  - Maskable interrrupt terminal ($\overline{INT}$)

  The following three modes are selectable:
  - 8080 compatible interrupt mode (interrupt by Non-Z80 family peripheral LSI) (Mode 0)
  - Restart interrupt (Mode 1)
  - Daisy-chain structure interrupt using Z80 family peripheral LSI (Mode 2)
- An auxiliary register provided to each of general purpose registers.
- 2 index registers
- 10 addressing modes
- Built-in refresh circuit for dynamic memory
- Molded in 44-pin PLCC package

GENERAL DESCRIPTION:

The Z84C01 is an 8-bit microprocessor (hereinafter referred to as MPU) with a built-in clock generator/controller, which provides low power operation and high performance.

Built into the Z84C01 is a control function and clock generator for the standby function in addition to: six paired general purpose registers, accumulator, flag registers, an arithmetic-and-logic unit, bus control, memory control and timing control circuits.

The Z84C01 is fabricated with Zilog CMOS technology and molded in a 44-pin PLCC package.

Further, in the following text and explanations for charts and tables, hexadecimal numbers are directly used without giving an identification to explanation of address, etc. so as not to cause confusions.

PIN CONNECTIONS AND PIN FUNCTIONS:

The pin connections and I/O pin names and brief functions of the Z84C01 are shown below.

Pin Connections. The pin connections of the Z84C01 are as shown in Fig. 1.

Pin Names and Functions. I/O pin names and functions are as shown in Table 1.

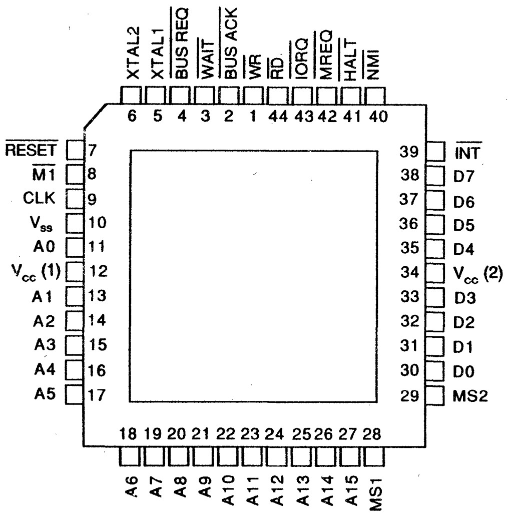


Figure 1. Pin Connections (Top View)

Table 1 Pin Names and Functions

Pin Name	Number of Pin	Input/Output 3-state	Function
A0 - A15	16	Output 3-state	16-bit address bus. Specify addresses of memories and I/O to be accessed. During the refresh period, addresses for refreshing are output.
MS1, MS2	2	Input	Mode selection input. One of 4 modes (Run, IDLE1/2, STOP) is selected according to the state of these 2 pins.
D0 - D7	8	I/O 3-state	8-bit bidirectional data bus.
$\overline{\text{INT}}$	1	Input	Maskable interrupt request signal. Interrupt is generated by peripheral LSI. This signal is accepted if the interrupt enable flip-flop (IFF) is set at "1". $\overline{\text{INT}}$ is normally wired-OR and requires an external pull up for these applications.

Table 1 Pin Names and Functions (continued)

Pin Name	Number of Pin	Input/Output 3-state	Function
$\overline{\text{NMI}}$	1	Input	Non-maskable interrupt request signal. This interrupt request has the higher priority than the maskable interrupt request and does not rely upon the state of the interrupt enable flip-flop (IFF).
$\overline{\text{HALT}}$	1	Output	Halt signal. Indicates that the CPU has executed a Halt instruction.
$\overline{\text{MREQ}}$	1	Output 3-state	Memory request signal. When an effective address for memory access is on the address bus, "0" is output.
$\overline{\text{IORQ}}$	1	Output 3-state	I/O request signal. When addresses for I/O are on lower 8 bits (A0 - A7) of the address bus in the I/O operation, "0" is output. In addition, $\overline{\text{IORQ}}$ signal is output together with $\overline{\text{M1}}$ signal at time of interrupt acknowledge cycle to inform peripheral LSI of the state that the interrupt response vector may be put on the data bus.
$\overline{\text{RD}}$	1	Output 3-state	Read signal. "0" signal is output for a period when MPU can receive data from a memory or peripheral LSI. It is possible to put data from a specified peripheral LSI or mamory on the MPU data bus after gating by this signal.
$\overline{\text{WR}}$	1	Output 3-state	Write signal. This signal is output when data to be stored in a specified memory or peripheral LSI is on the MPU data bus.
$\overline{\text{BUSACK}}$	1	Output	Bus acknowledge signal. In response to $\overline{\text{BUSREQ}}$ signal, this signal informs a peripheral LSI of the fact that the address bus, data bus, MREQ, IORQ, RD and WR signals have been placed in the high impedance state.
$\overline{\text{WAIT}}$	1	Input	Wait signal. $\overline{\text{WAIT}}$ signal is a signal to inform MPU of specified memory or peripheral LSI which is not ready for data transfer. As long as WAIT signal as at "0" level, MPU is continuously kept in the wait state.
$\overline{\text{BUSREQ}}$	1	Input	Bus request signal. BUSREQ signal is a signal requesting placement of the address bus, data bus, $\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals in the high impedance state. $\overline{\text{BUSREQ}}$ signal is normally wired-OR. In this case, a pull-up resistor is externally connected.

Table 1 Pin Names and Functions (continued)

Pin Name	Number of Pin	Input/Output 3-state	Function
$\overline{\text{RESET}}$	1	Input	Reset signal. $\overline{\text{RESET}}$ signal is used for initializing MPU and must be kept in active state ("0") for a period of at least 3 clocks.
$\overline{\text{M1}}$	1	Output	Signal showing machine cycle 1. "0" is output together with $\overline{\text{MREQ}}$ signal in the operation code fetch cycle. This signal is output for every opcode fetch when 2 byte opcode is executed. In the maskable interrupt acknowledge cycle, this signal is output together with IORQ signal.
XTAL 1 (XIN) XTAL 2 (XOUT)	2	Input Output	Crystal oscillator connecting terminal.
CLK	1	Output	Single-phase clock output. Clock polarity is in-phase with OSC-IN (XTAL 1) so that Z80 users could use OSC-IN as clock input without needing extra inverter on the board. When the HALT in struction in STOP Mode is executed, MPU stops its operation and holds clock output at "0" level.
VCC (1), (2)	2	Power supply	+5V Connect pin 34 and pin 12 externally.
VSS	1	Power supply	0V

FUNCTIONAL DESCRIPTION:

The system configuration, functions and basic operation of the Z84C01 are described here.

Block Diagram. The block diagram of the internal configuration is shown in Fig. 2.

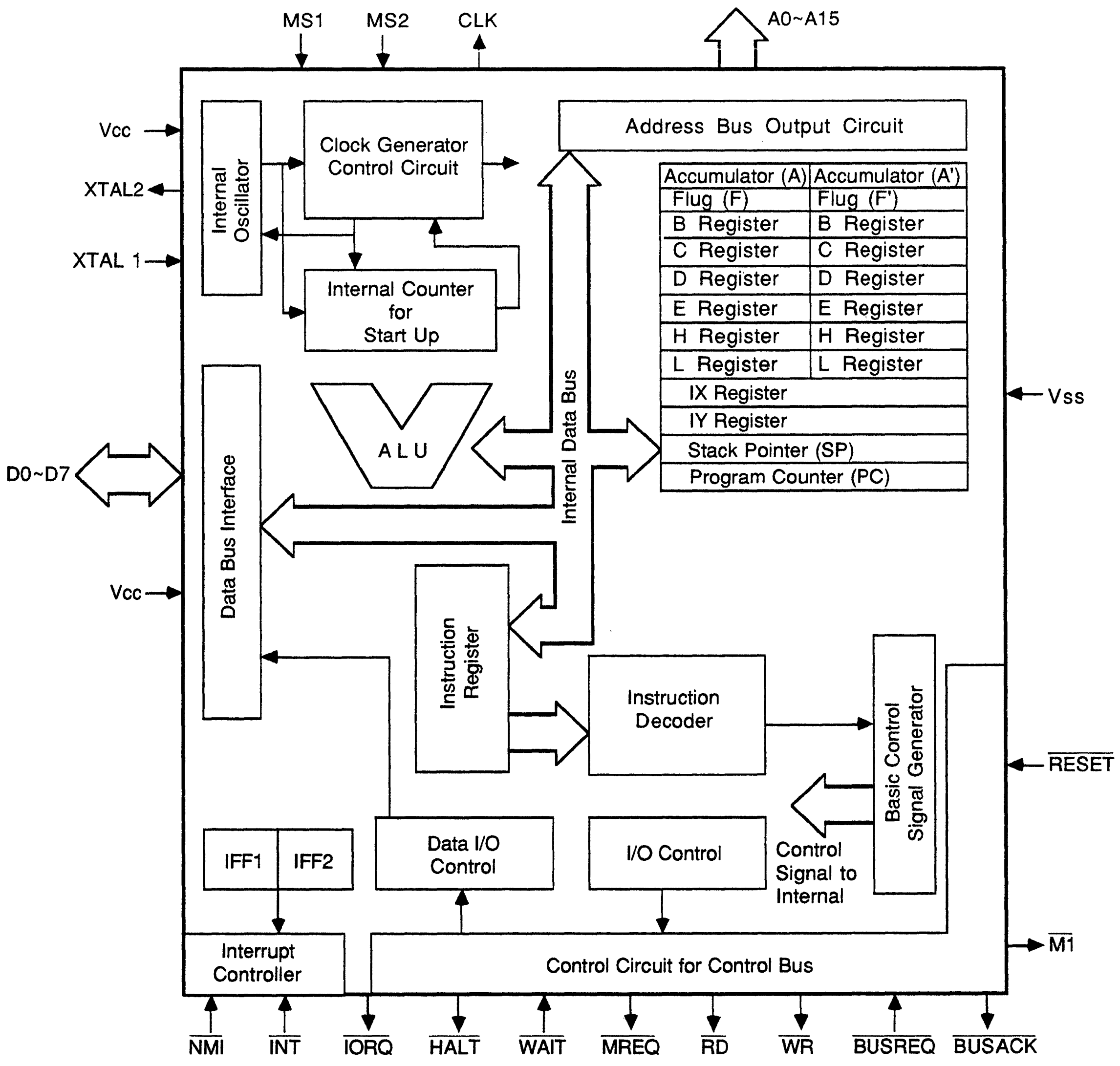


Fig. 2 Block Diagram

System Configuration. The Z84C01 has a built-in system clock generator for CMOS Z80 in addition to the standard functions of the Z84C00 MPU. The explanation is provided here with emphasis placed on the halt function relative to the clock generator, which is an additional function. The internal register group, reset and interrupt function are identical to those of the Z84C00. For details, please refer to the data sheet for the Z84C00.

In this section, the following principal components and functions will be described:

(1) Generation of clock
(2) Operation mode
(3) Start-up time at time of restart

Generating the System Clock. The Z84C01 has a built-in oscillation circuit and required clock can be easily generated by connecting an oscillator to the external terminals (XTAL1, XTAL2). Clock in the same frequency as input oscillation frequency is generated.
Examples of oscillator connection are shown in Fig. 0.0.

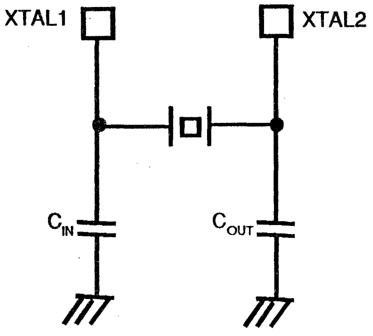


Figure 3a Example of Oscillator Connection and Constant

C_{IN}	C_{OUT}
22_{PF}	33_{PF}

Figure 3b Example of Oscillator Connection and Constant

Operation Modes. There are four kinds of operation modes available for the Z84C01 in connection with generation of clock; RUN Mode, IDLE1/2 Modes and STOP Mode. One of these modes is selected by the mode select inputs (MS1, MS2).

The operation mode is effective when the halt instruction is executed. Restart of MPU from the stopped state under IDLE1/2 Mode or STOP Mode is effected by inputting either RESET signal or interrupt signal (INT or NMI).

Operations of these modes in the halt state are shown in Table 2.

Table 2 Clock Generating Operation Mode

Operation Mode	MS1	MS2	Description at HALT State
RUN Mode	1	1	MPU continues the operation and supplies clock to the outside continuously.
IDLE 1 Mode	0	0	The internal oscillator's operation is continued. Clock (CLK) output as well as internal operations are stopped at "0" level of T4 state in the halt instruction operation code fetch cycle.
IDLE 2 Mode	0	1	The internal oscillator's operation and clock (CLK) output are continued but the internal operations are stopped at "0" level of T4 state in the halt instruction operation code fetch cycle.
STOP Mode	1	0	All operations of the internal oscillator, clock (CLK) output, and internal operation are stopped at "0" level of T4 state in the halt instruction operation code fetch cycle.

Start-up Time at Time of Restart (STOP Mode). When MPU is released from the halt state by accepting an interrupt request, MPU, then will execute an interrupt service routine. Therefore, when an interrupt request is accepted, MPU starts generation of internal system clock and clock output after a start-up time by the internal counter (2^{14}+2.5) TcC (TcC: Clock Cycle) to obtain a stabilized oscillation for MPU operation.

Further, in case of the restart by $\overline{\text{RESET}}$ signal, the internal counter does not operate for a quick operation at time of power ON.

Status Change Flowchart and Basic Timing. In this section, the status change and basic timing when the Z84C01 is operating are explained.

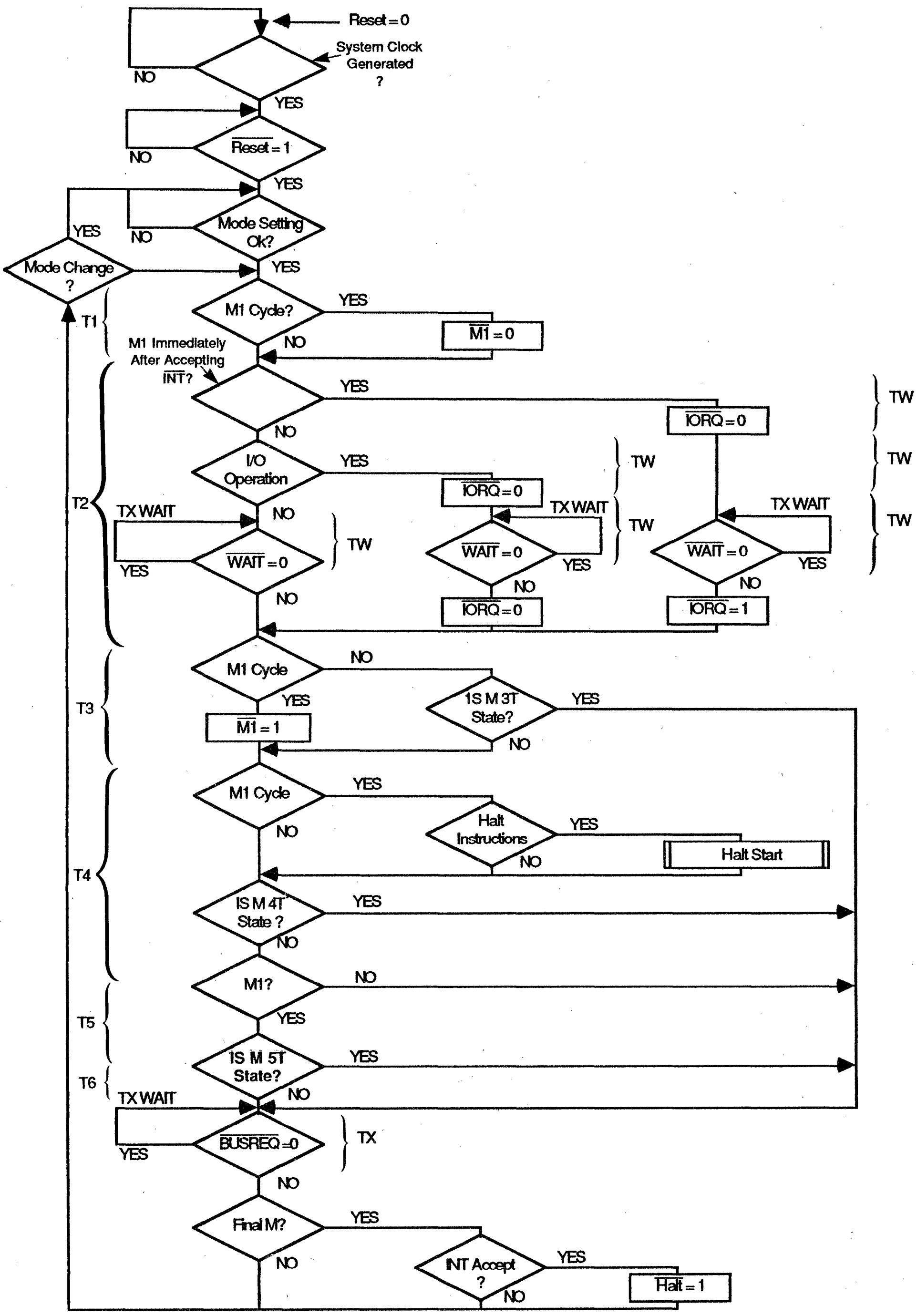


Figure 4 (a) Status Change Flowchart

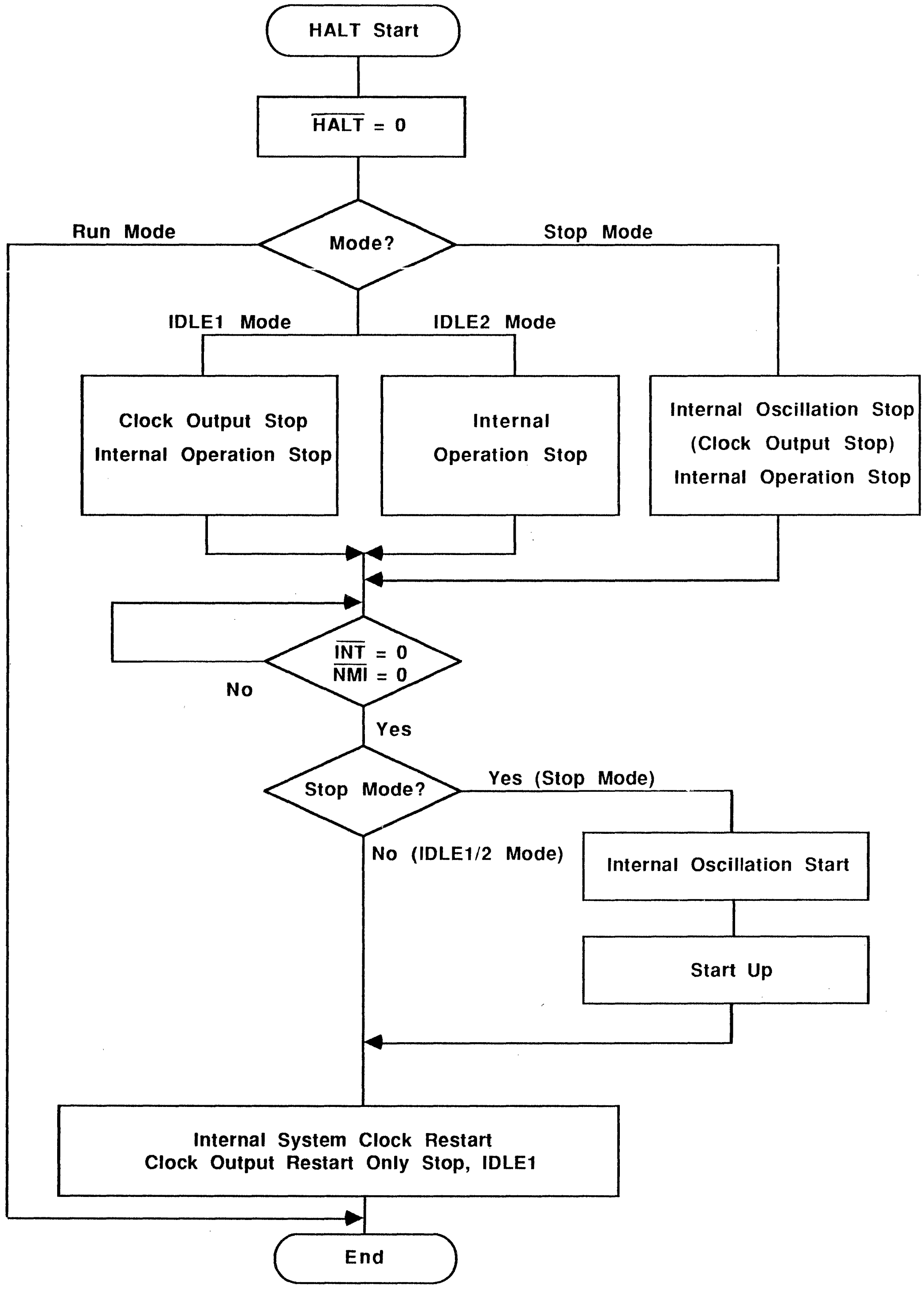


Figure 4 (b) Status Change Flowchart

Basic Timing. The basic timing is explained here with emphasis placed on the halt function relative to the clock generator. Except $\overline{RFSH}$ signal output, the following items are identical to those for the Z84C00. Refer to the data sheet for the Z84C00.

Operation code fetch cycle
Memory read/write operation
Input/output operation
Bus request/acknowledge operation
Maskable interrupt request operation
Non-maskable interrupt request operation
Reset operation

Note that the Z84C01 does not have the refresh terminal ($\overline{RFSH}$), but refresh address is output on the address bus in the operation code fetch cycle (M1) as in the Z84C00 since the on-chip refresh control circuit is available.

(1) Operation When HALT Instruction is Executed
When MPU fetches a halt instruction in the operation code fetch cycle, HALT signal goes active (low level) in synchronous with falling edge of T4 state for the peripheral LSI and MPU stops the operation. The system clock generating operation after this differs depending upon the operation mode (RUN Mode, IDLE1/2 Mode or STOP Mode). If the internal system clock is running, MPU continues to execute NOP instruction even in the halt state.

(a) RUN Mode (MS1=1, MS2=1)
Shown in Fig. 5 is the basic timing when the halt instruction is executed in RUN Mode.

In RUN Mode, system clock (ø) in MPU and clock output (CLK) are not stopped, even after the halt instruction is executed. Therefore, until the halt state is released by the interrupt signal ($\overline{NMI}$ or $\overline{INT}$) or $\overline{RESET}$ signal, MPU continues to execute NOP instruction.

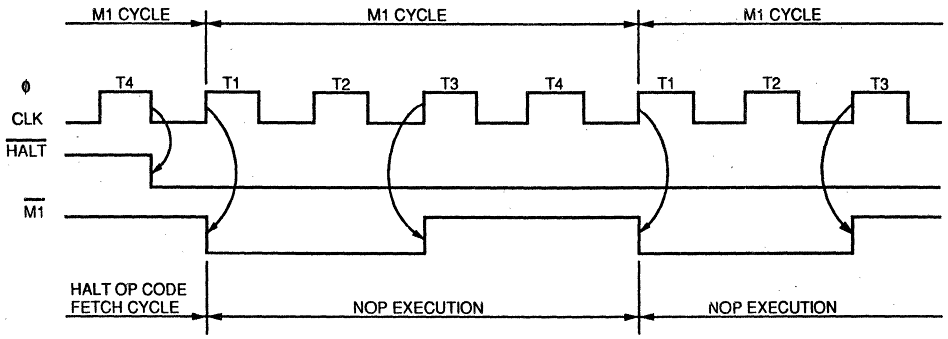
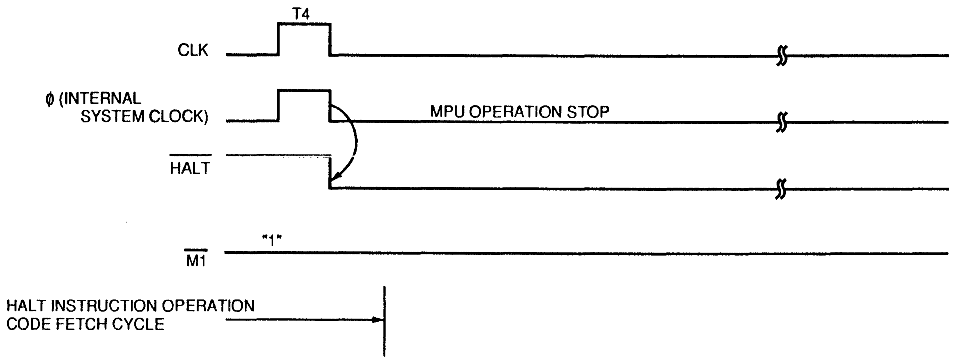


Figure 5 Timing of RUN Mode (at Halt Command Execution)

(b) IDLE1 Mode (MS1=0, MS2=0)
Shown in Fig. 6 is the basic timing when the halt instruction is executed in IDLE1 Mode.

In IDLE1 Mode, system clock (ø) in MPU and clock output (CLK) are stopped and MPU stops its operation after the halt instruction is executed. However, the internal oscillator continues to operate.

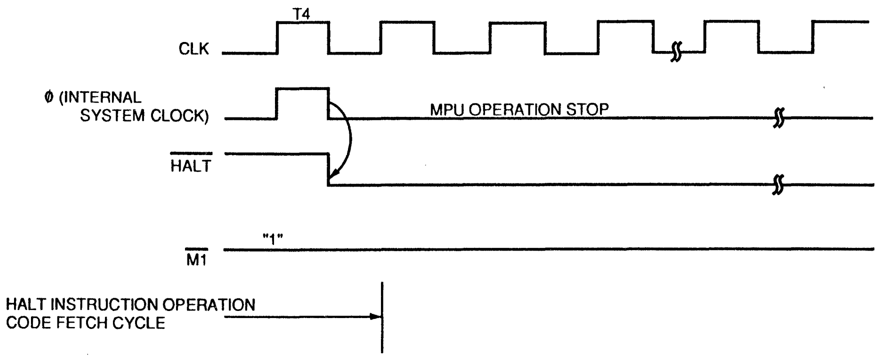


T4
CLK
ϕ (INTERNAL SYSTEM CLOCK)
MPU OPERATION STOP
HALT
"1"
M1
HALT INSTRUCTION OPERATION CODE FETCH CYCLE

Figure 6 IDLE1 Mode Timing (at Halt Instruction Execution)

(c) IDLE2 Mode (MS1=0, MS2=1)
Shown in Fig. 7 is the basic timing when the halt instruction is executed in IDLE2 Mode.

In IDLE2 Mode, system clock (ø) in MPU is stopped and MPU stops its operation after the halt instruction is executed. However, the internal oscillator and clock output (CLK) to the outside of MPU continues to operate.



T4
CLK
ϕ (INTERNAL SYSTEM CLOCK)
MPU OPERATION STOP
HALT
"1"
M1
HALT INSTRUCTION OPERATION CODE FETCH CYCLE

Figure 7 IDLE2 Mode Timing (at Halt Instruction Execution)

(d) STOP Mode (MS1=1, MS2=0)
Shown in Fig. 8 is the basic timing when the halt instruction is executed in STOP Mode.

In STOP Mode, internal operation and internal oscillator are stopped after the halt instruction is executed. Therefore, system clock (ϕ) in MPU and clock output (CLK) to the outside of MPU are stopped.

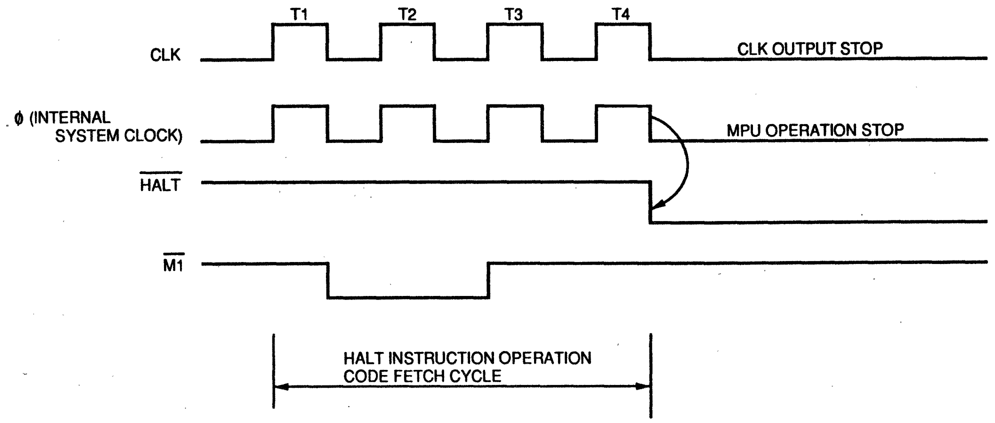


Figure 8 STOP Mode Timing (at Halt Instruction Execution)

(2) Release from Halt State
The halt state of MPU is released when "0" is input to $\overline{\text{RESET}}$ signal and MPU is reset or an interrupt request is accepted. An interrupt request signal is sampled at the leading edge of the last clock cycle (T4 state) of NOP instruction. In case of the maskable interrupt, interrupt will be accepted by an active INT signal ("0" level). Also the interrupt enable flip-flop must have been set to "1". The accepted interrupt process is started from next cycle.

Further, when the internal system clock is stopped (IDLE1/2 Mode, STOP Mode), it is necessary first to restart the internal system clock. The internal system clock is restarted when RESET or interrupt signal (NMI or INT) is input.

(a) RUN Mode (MS1, MS2=1)
The halt release operation by acceptance of interrupt request in RUN Mode is shown in Fig. 9.

In RUN Mode the internal system clock is not stopped, and therefore, if the interrupt signal is recognized at the rise of T4 state of the continued NOP instruction, MPU will execute the interrupt process from next cycle.

The halt release operation by resetting MPU in RUN Mode is shown in Fig. 10. After reset, MPU will execute an instruction starting from address 0000H. However, in order to reset MPU it is necessary to keep RESET signal at "0" for at least 3 clocks. In addition, if RESET signal becomes "1", after the dummy cycle for at least two T states, MPU executes an instruction from address 0000H.

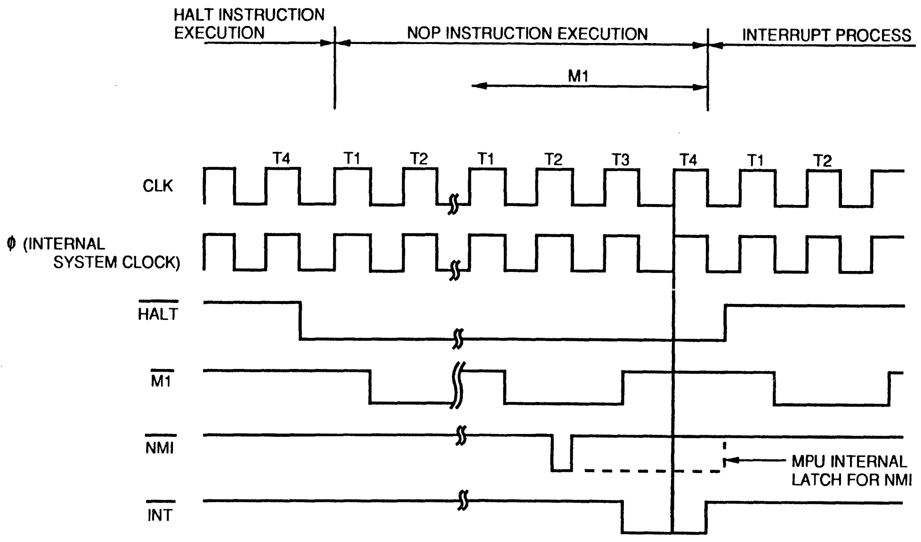


Figure 9 Halt Release Operation Timing by interrupt Request Signal in RUN Mode

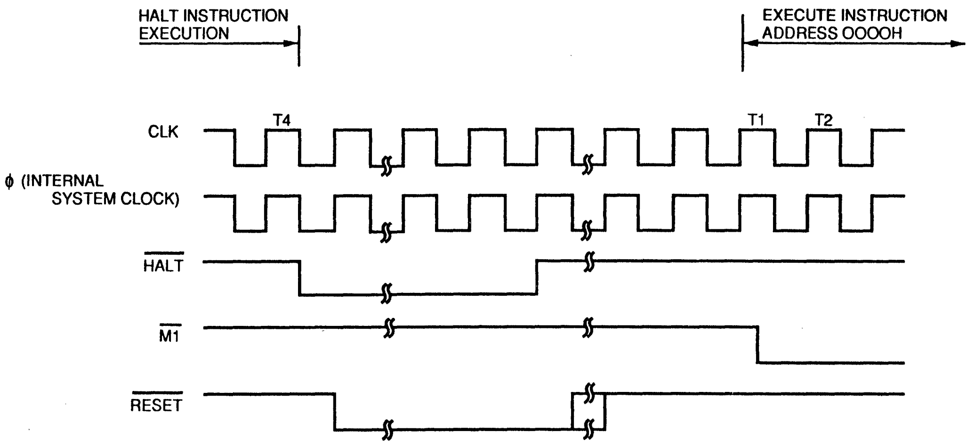

HALT INSTRUCTION EXECUTION
EXECUTE INSTRUCTION ADDRESS OOOOH
T4
T1
T2
CLK
ϕ (INTERNAL SYSTEM CLOCK)
$\overline{\text{HALT}}$
$\overline{\text{M1}}$
$\overline{\text{RESET}}$

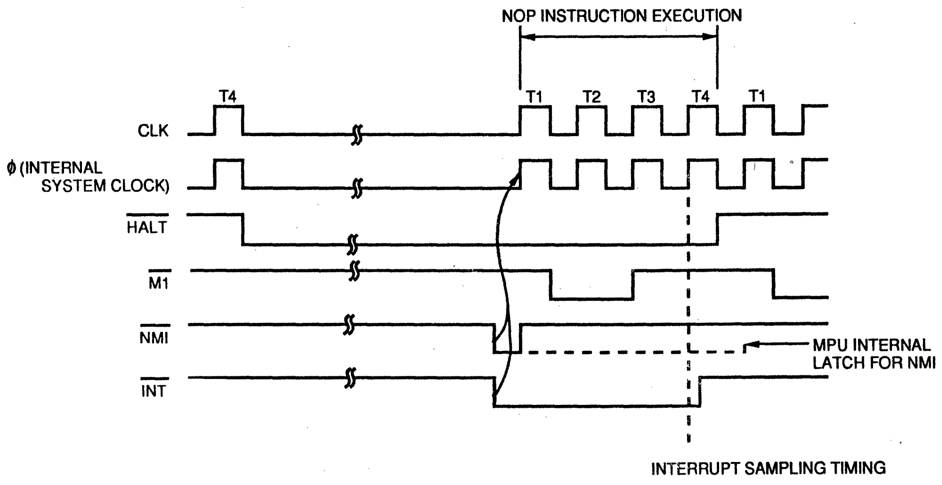
Figure 10 Halt Release Operation Timing by Reset in RUN Mode

(b) IDLE1 Mode (MS1=0, MS2=0), IDLE2 Mode (MS1=0, MS2=1)

The halt release operation by interrupt signal in IDLE1 Mode is shown in Fig. 11 (a) and in IDLE2 Mode in Fig. 11 (b).

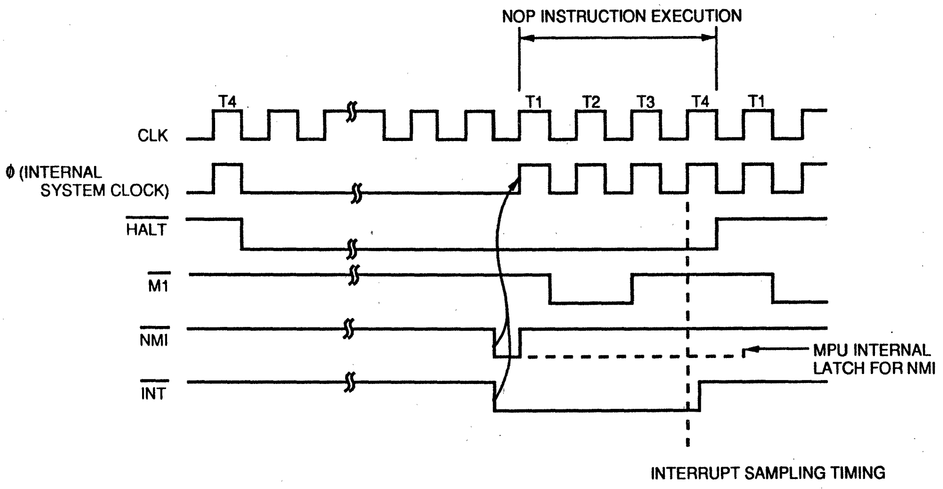
When receiving $\overline{NMI}$ or $\overline{INT}$ signal, MPU starts the internal system clock operation. In IDLE1 Mode, MPU starts clock output to the outside at the same time.

The operation stop of MPU in IDLE1/2 Mode is taking place at "0" level during T4 state in the halt instruction operation code fetch cycle. Therefore, after being re-started by the interruption signal, MPU executes one NOP instruction and samples an interrupt signal at the rise of T4 state during the execution of this NOP instruction, and executes the interrupt process from next cycle.



NOP INSTRUCTION EXECUTION
T4
T1
T2
T3
T4
T1
CLK
ϕ (INTERNAL SYSTEM CLOCK)
HALT
M1
NMI
INT
MPU INTERNAL LATCH FOR NMI
INTERRUPT SAMPLING TIMING

Figure 11 (a) IDLE1 Mode



NOP INSTRUCTION EXECUTION
T4
T1
T2
T3
T4
T1
CLK
ϕ (INTERNAL SYSTEM CLOCK)
HALT
M1
NMI
INT
MPU INTERNAL LATCH FOR NMI
INTERRUPT SAMPLING TIMING

Figure 11 (b) IDLE2 Mode

Figure 11 Halt Release Operation Timing by Interrupt Request Signal in IDLE1/2 Mode

If no interrupt signal is accepted during the execution of the first NOP instruction after the internal system clock is restarted, MPU is not released from the halt state and is placed in IDLE1/2 Mode again at "0" level during T4 state of the NOP instruction, stopping the internal system clock. If $\overline{\text{INT}}$ signal is not at "0" level at the rise of T4 state, no interrupt request is accepted.

The halt release operation by resetting MPU in IDLE1 Mode is shown in Fig. 12 (a) and that in IDLE2 Mode in Fig. 12 (b).

When $\overline{\text{RESET}}$ signal at "0" level is input into MPU, the internal system clock is restarted and MPU will execute an instruction stored in address 0000H.

At time of $\overline{\text{RESET}}$ signal input, it is necessary to take the same care as that in resetting MPU in RUN Mode.

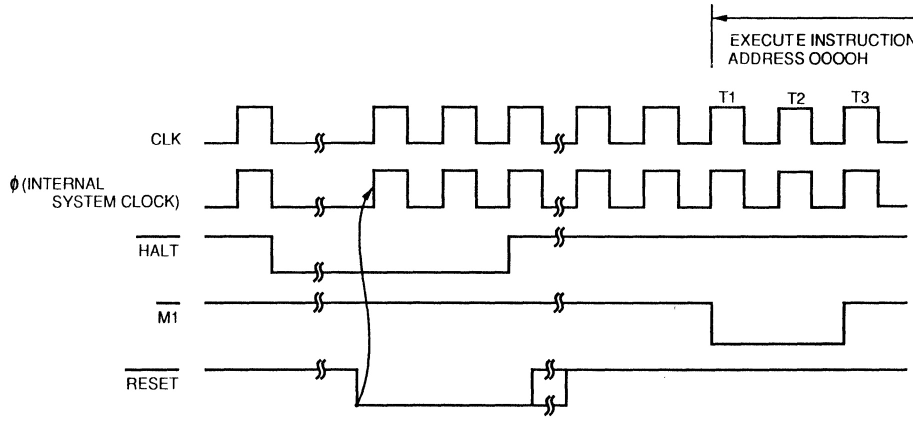

EXECUTE INSTRUCTION FROM ADDRESS 0000H
T1
T2
T3
CLK
ϕ (INTERNAL SYSTEM CLOCK)
HALT
M1
RESET

Figure 12 (a) IDLE1 Mode

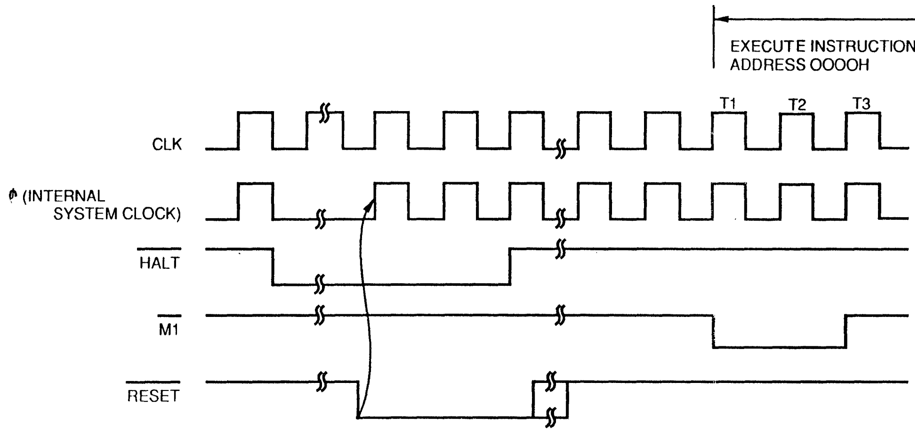

EXECUTE INSTRUCTION FROM ADDRESS 0000H
T1
T2
T3
CLK
ϕ (INTERNAL SYSTEM CLOCK)
HALT
M1
RESET

Figure 12 (b) IDLE2 Mode

Figure 12 Halt Release Operation Timing by Reset in IDLE1/2 Mode

(c) STOP Mode (MS1=1, MS2=0)

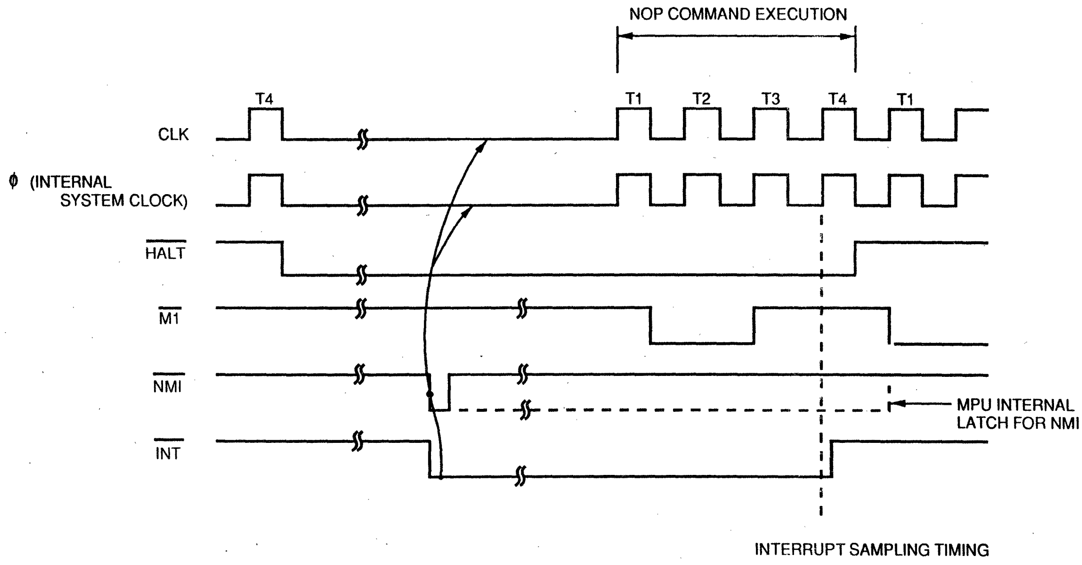
The halt release operation by interrupt signal in STOP Mode is shown in Fig. 13.

When MPU received an interrupt signal, the internal oscillator is restarted. In order to obtain stabilized oscillation, the internal system clock and clock output to the outside are started after a start-up time of (2^{14}+2.5) TcC (TcC: Clock Cycle) by the internal counter.

MPU executes one NOP instruction after the internal system clock is restarted and at the same time, sampling an interrupt signal at the rise of T4 state during the execution of this NOP instruction. If the interrupt signal is accepted, MPU executes the interrupt process operation from next cycle.

At time of interrupt signal input, it is necessary to take the same care as that in the interrupt signal input in IDLE1/2 Mode. The halt release operation by MPU resetting in STOP Mode is shown in Fig. 14.

When $\overline{\text{RESET}}$ signal at "0" level is input into MPU, the internal oscillator is restarted. However, since it performs a quick operation at time of power ON, the internal counter does not operate. Therefore, the operation may not be carried out properly due to unstable clock immediately after the signal in STOP Mode, it is necessary to hold $\overline{\text{RESET}}$ signal at "0" level for sufficient time. When $\overline{\text{RESET}}$ signal becomes "1", after the dummy cycle for at least 2T states, MPU starts to execute an execution from address 0000H.



NOP COMMAND EXECUTION
T4
T1
T2
T3
T4
T1
CLK
ϕ (INTERNAL SYSTEM CLOCK)
HALT
M1
NMI
INT
MPU INTERNAL LATCH FOR NMI
INTERRUPT SAMPLING TIMING

Figure 13 Halt Release Operation Timing by Interrupt Request Signal in STOP Mode

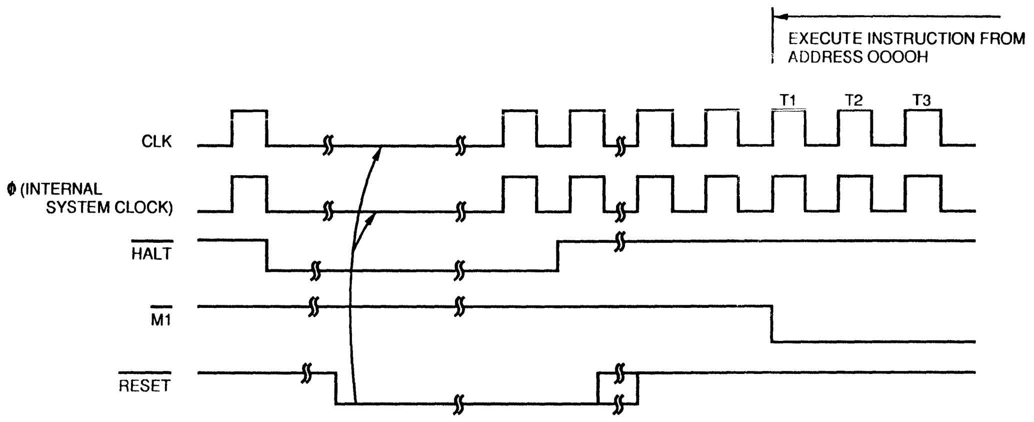

EXECUTE INSTRUCTION FROM ADDRESS OOOOH
T1
T2
T3
CLK
ɸ (INTERNAL SYSTEM CLOCK)
$\overline{HALT}$
$\overline{M1}$
$\overline{RESET}$

Figure 14 Halt Release Operation Timing by Reset in STOP Mode

Instruction Set. Instruction set of the Z84C01 is the same as that for the Z84C00. For details refer to the data sheet for the Z84C00.

Method of Use. An example of the Z84C01 with the Z80 family peripheral LSI's is shown in Fig. 15.

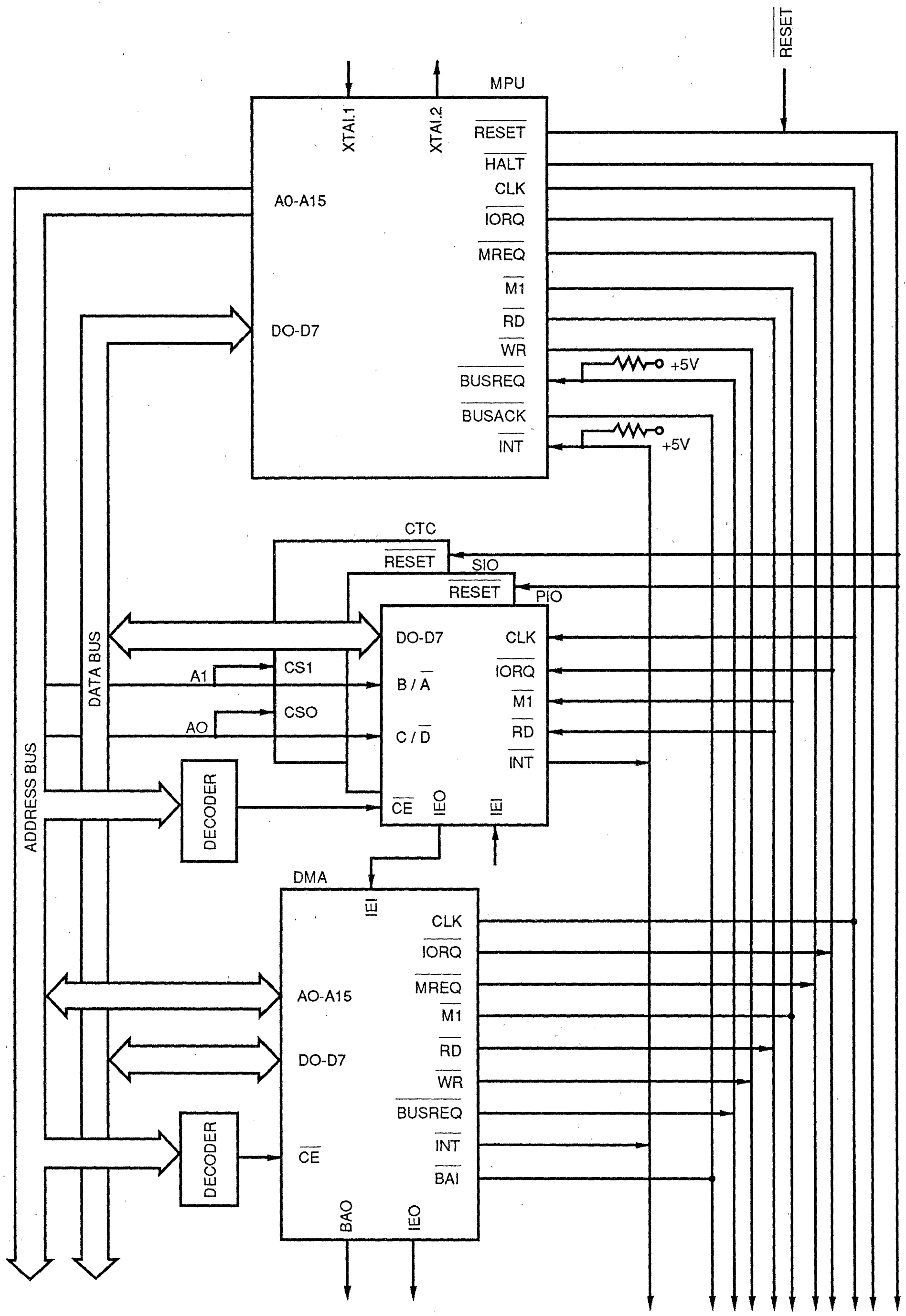


Figure 15 Example of Z80 Family Peripheral LSI

CPU TIMING

Timing Diagrams. The Z84C01 CPU executes instructions by proceeding through a specific sequence of operations:

- Memory read or write
- I/O device read or write
- Interrupt acknowledge

The basic clock period is referred to as a Time or Cycle, and three or more T cycles make up a machine cycle (M1, M2 or M3 for instance). Machine cycles can be extended either by the CPU automatically inserting one or more Wait states or by the insertion of one or more Wait states by the user.

Instruction Opcode Fetch. The CPU places the contents of the Program Counter (PC) on the address bus as the start of the cycle (Figure 16). Approximately one-half clock cycle later, $\overline{MREQ}$ goes active. When active, $\overline{RD}$ indicates that the memory data can be enabled onto the CPU data bus.

The CPU samples the $\overline{WAIT}$ input with the falling edge of clock state T2. During clock states T3 and T4 of an $\overline{M1}$ cycle, dynamic RAM refresh can occur while the CPU starts decoding and executing the instruction.

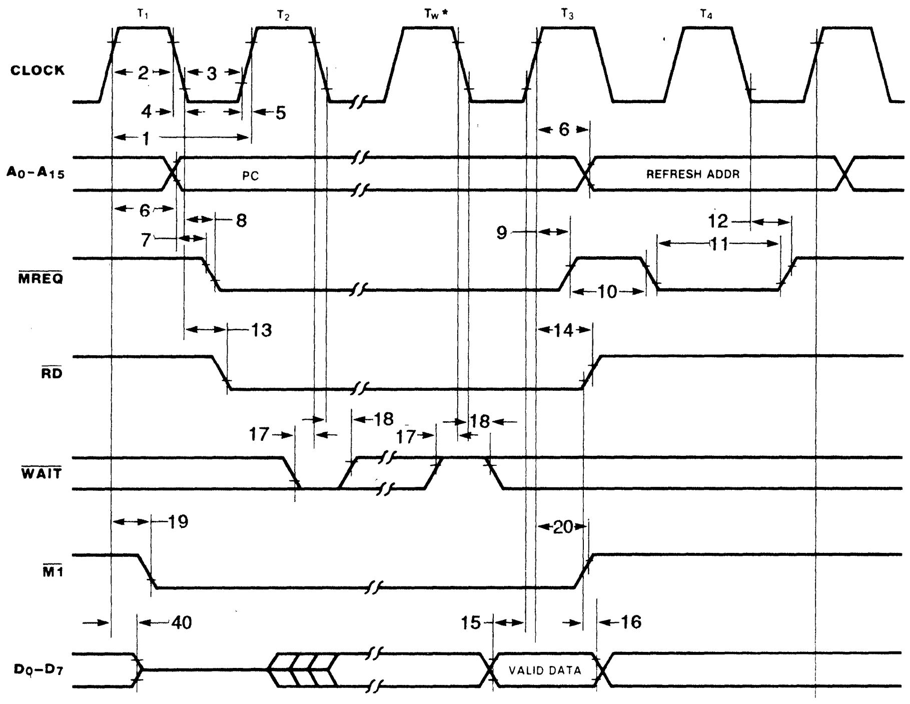


Figure 16 Instruction Opcode Fetch

Memory Read or Write Cycles. Figure 17 shows the timing of memory read or write cycles other than an opcode fetch ($\overline{M1}$) cycle. The $\overline{MREQ}$ and $\overline{RD}$ signals function exactly as in the fetch cycle.

In a memory write cycle, $\overline{MREQ}$ also becomes active when the address bus is stable. The $\overline{WR}$ line is active when the data bus is stable, so that it can be used directly as an R/$\overline{W}$ pulse to most semiconductor memories.

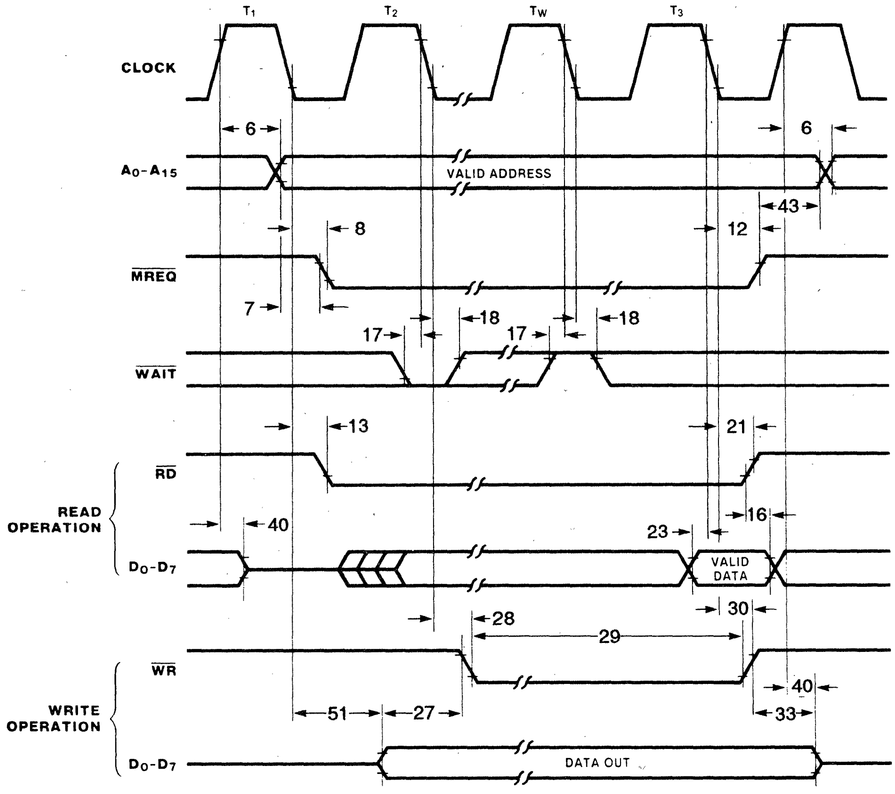


Figure 17 Memory Read or Write Cycles

Input or Output Cycles. Fig. 18 shows the timing for an I/O read or I/O write operation. During I/O operations, the CPU automatically inserts a single Wait state (T_{WA}). This extra Wait state allows sufficient time for an I/O port to decode the address from the port address lines.

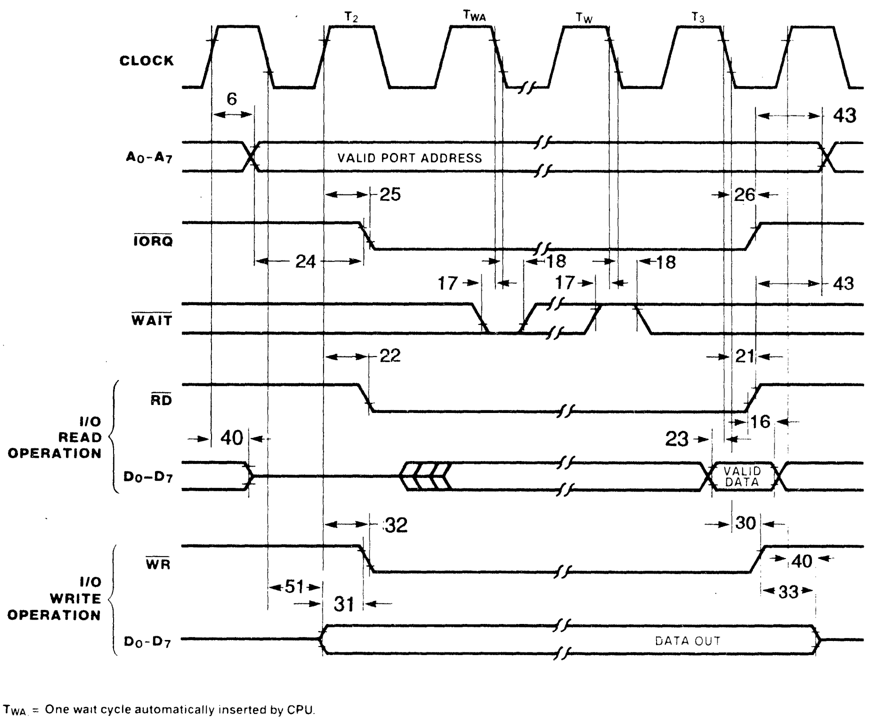


Figure 18 Input or Output Cycles

Interrupt Request/Acknowledge Cycle. The CPU samples the interrupt signal with the rising edge of the last clock cycle at the end of any instruction (Fig. 19). When an interrupt is accepted, a special $\overline{M1}$ cycle is generated. During this $\overline{M1}$ cycle, $\overline{IORQ}$ becomes active (instead of $\overline{MREQ}$) to indicate that the interrupting device can place an 8-bit vector on the data bus. The CPU automatically adds two Wait states to this cycle.

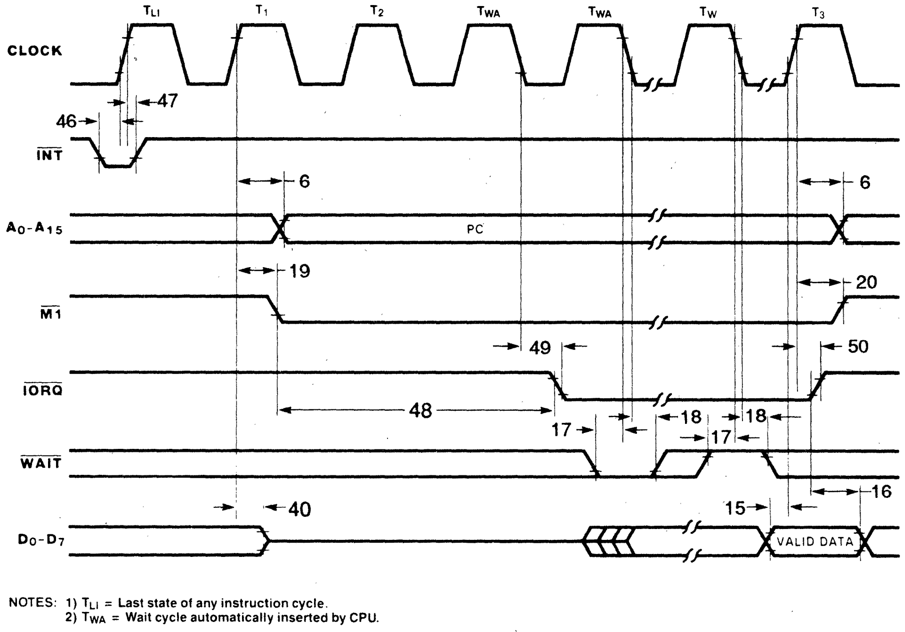
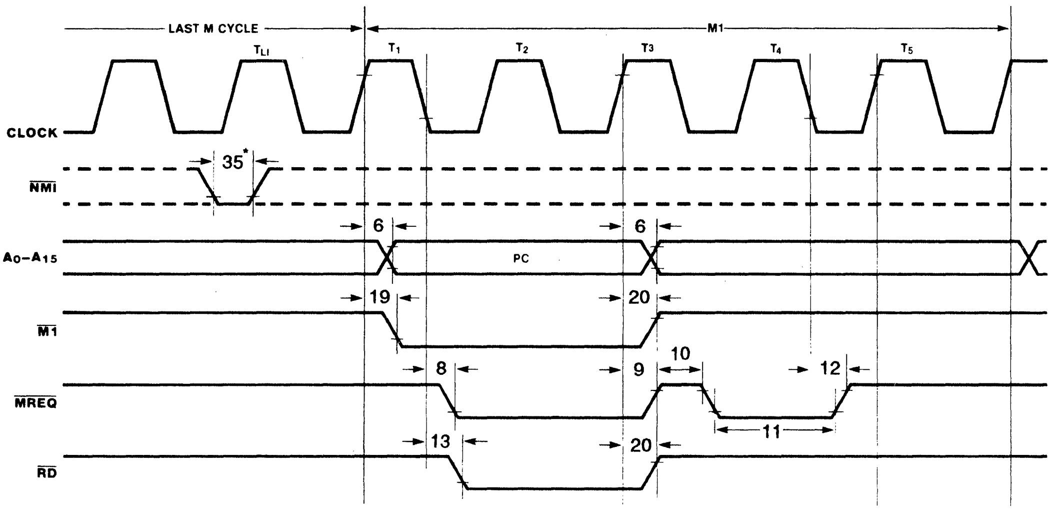


Figure 19 Interrupt Request/Acknowledge Cycle

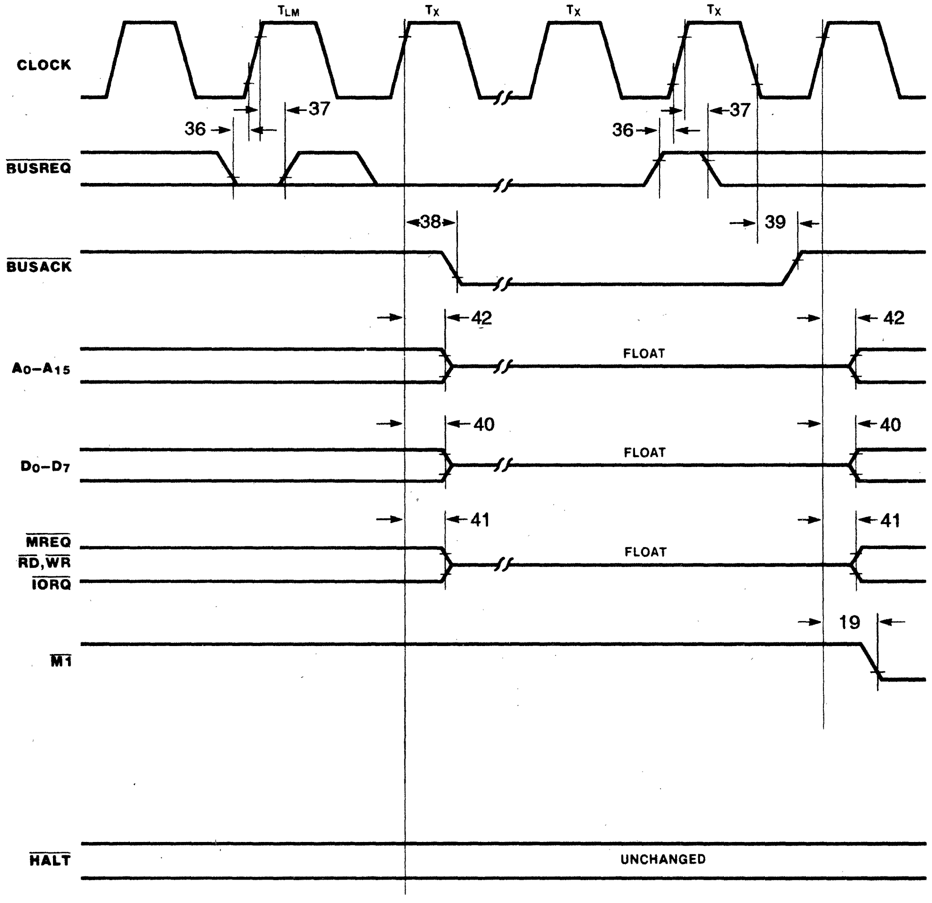
Non-Maskable Interrupt Request Cycle. $\overline{NMI}$ is sampled at the same time as the maskable interrupt input $\overline{INT}$, but has higher priority and cannot be disabled under software control. The subsequent timing is similar to that of a normal memory read operation except that data put on the bus by the memory is ignored. The CPU instead executes a restart (RST) operation and jumps to the $\overline{NMI}$ service routine located at address 0066H (Fig. 20).



*Although $\overline{NMI}$ is an asynchronous input, to guarantee its being recognized on the following machine cycle, $\overline{NMI}$'s falling edge must occur no later than the rising edge of the clock cycle preceding the last state of any instruction cycle (T_{LI}).

Figure 20 Non-Maskable Interrupt Request Operation

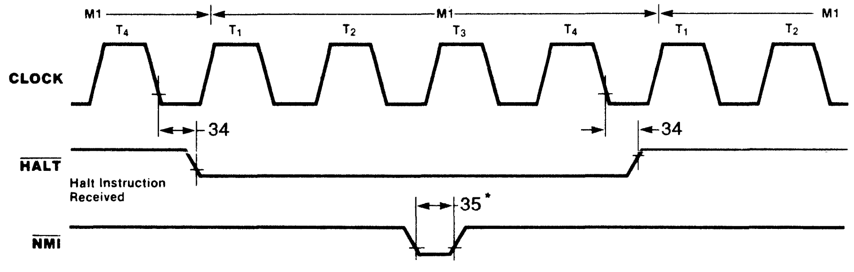
Bus Request/Acknowledge Cycle. The CPU samples $\overline{BUSREQ}$ with the rising edge of the last clock period of any machine cycle (Fig. 21). If $\overline{BUSREQ}$ is active, the CPU sets its address, data, and $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, and $\overline{WR}$ lines to a high-impedance state with the rising edge of the next clock pulse. At that time, any external device can take control of these lines, usually to transfer data between memory and I/O devices.



NOTES: 1) T_{LM} = Last state of any M cycle.
2) T_X = An arbitrary clock cycle used by requesting device.

Figure 21 BUS Request/Acknowledge Cycle

Halt Acknowledge Cycle.



*Although NMI is an asynchronous input, to guarantee its being recognized on the following machine cycle, NMI's falling edge must occur no later than the rising edge of the clock cycle preceding the last state of any instruction cycle (T_{LI}).

Figure 22 Halt Acknowledge

Reset Cycle. RESET must be active for at least three clock cycles for the CPU to properly accept it. As long as RESET remains active, the address and data buses float, and the control outputs are inactive.

Once RESET goes inactive, two internal T cycles are consumed before the CPU resumes normal processing operation. RESET clears the PC register, so the first opcode fetch will be location 0000H (Fig. 23).

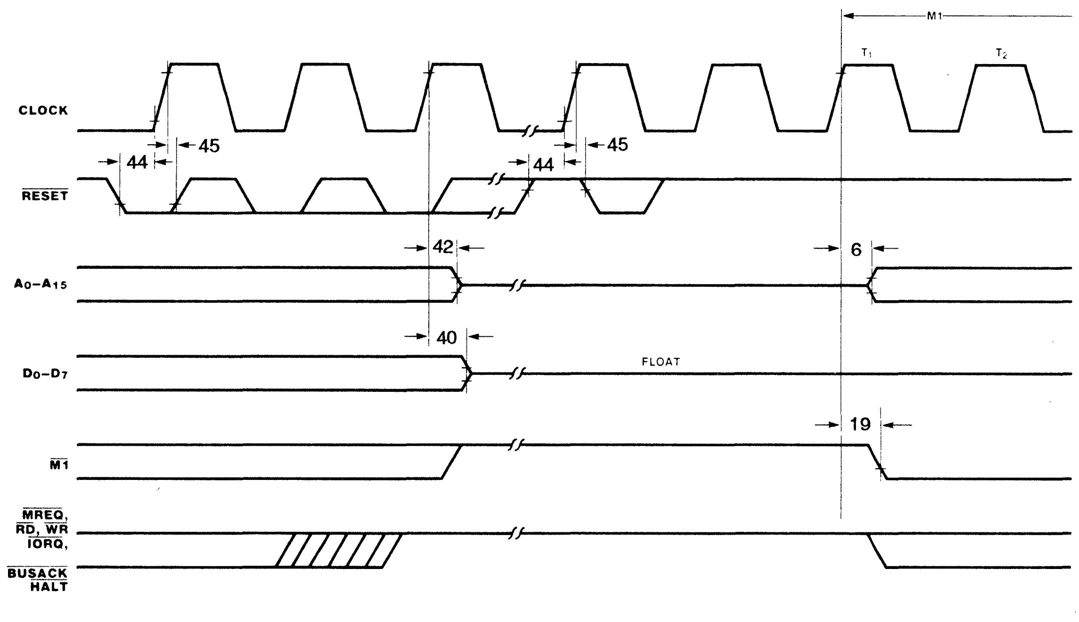


Figure 23 Reset Cycle

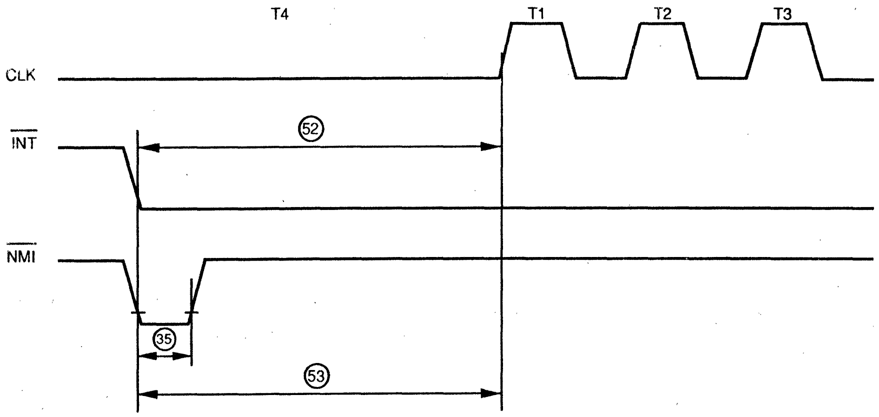


Figure 24 Clock Restart Timing (STOP Mode)

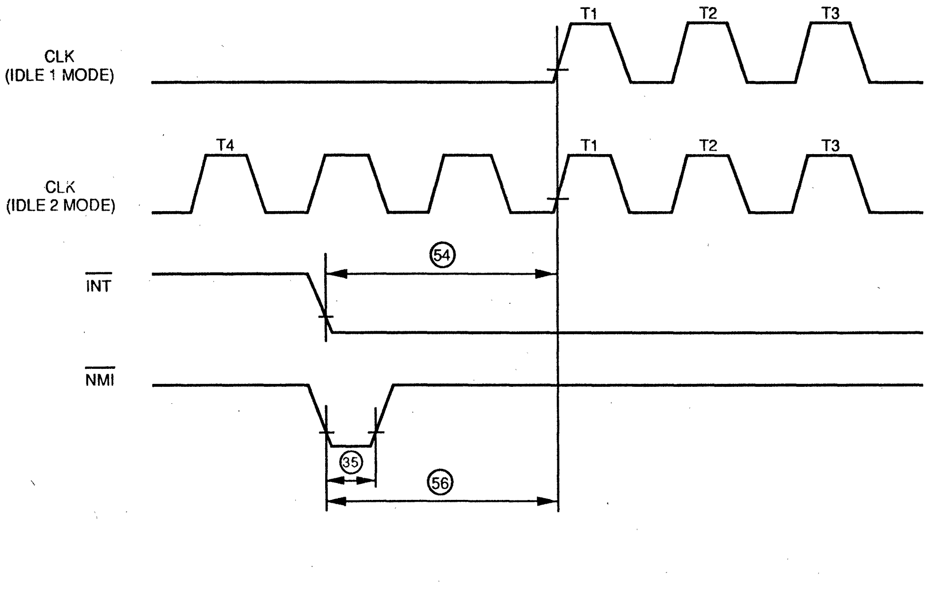


Figure 25 Clock Restart Timing (IDLE1/2 Mode)

PRECAUTIONS:

(1) To reset MPU, it is necessary to hold $\overline{\text{RESET}}$ signal input at "0" level for at least three clocks.

In particular, to release the HALT state by $\overline{\text{RESET}}$ signal in STOP Mode, hold $\overline{\text{RESET}}$ signal at "0" level for sufficient time in order to stabilize output from the internal oscillator.

(2) In releasing MPU from the HALT state by inter rupt signal in IDLE1/2 Mode and STOP Mode, MPU will not be released from the HALT state and the internal system clock will stop again unless an interrupt signal is accepted during the execution of NOP instruction even when the internal system clock is restarted by the interrupt signal input. In particular, care must be taken when $\overline{\text{INT}}$ is used.

Other precautions are identical to those for the Z84C00, except those for $\overline{\text{RFSH}}$ terminal. Refer to the data sheet for the Z84C00.

AC CHARACTERISTICS:

AC CHARACTERISTICS TA = 0° C to 70° C, VCC = 5V +/- 10%, VSS = 0V.

Number	Symbol	Parameter	Min	Max	Unit
1	TcC	Clock cycle time	100	DC	ns
2	TwCh	High clock pulse width	40	DC	ns
3	TwCl	Low clock pulse width	40	DC	ns
4	TfC	Clock falling time		10	ns
5	TrC	Clock rising time		10	ns
6	TdCr (A)	Effective address output delay from clock rise		60	ns
7	TdA (MREQf)	Address output definite time prior to $\overline{\text{MREQ}}$	**		ns
8	TdCf (MREQf)	Delay from clock fall to $\overline{\text{MREQ}}$ = "L"		40	ns
9	TdCf (MREQr)	Delay from clock rise to $\overline{\text{MREQ}}$ = "H"		40	ns
10	TwMREQh	$\overline{\text{MREQ}}$ high level pulse width	**		ns
11	TwMREQl	$\overline{\text{MREQ}}$ low level pulse width	**		ns
12	TdCf (MREQr)	Delay from clock fall to $\overline{\text{MREQ}}$ = "H"		40	ns
13	TdCf (RDf)	Delay from clock fall to $\overline{\text{RD}}$ = "L"		40	ns
14	TdCr (RDr)	Delay from clock rise to $\overline{\text{RD}}$ = "H"		40	ns
15	TsD (Cr)	Data set-up time for clock rise	25		ns
16	ThD (RDr)	Data hold time for $\overline{\text{RD}}$ rise	0		ns
17	TsWAIT (Cf)	$\overline{\text{WAIT}}$ signal set-up time for clock fall	30		ns
18*	ThWAIT (Cf)	$\overline{\text{WAIT}}$ hold time after clock fall	0		ns
19	TdCr (MIf)	Delay from clock rise to $\overline{\text{M1}}$ = "L"		40	ns
20	TdCr (MIr)	Delay from clock rise to $\overline{\text{M1}}$ = "H"		40	ns
21	TdCf (RDr)	Delay from clock fall to $\overline{\text{RD}}$ = "H"		40	ns
22	TdCr (RDF)	Delay from clock rise to $\overline{\text{RD}}$ = "L"		40	ns
23	TsD (Cf)	Data set-up time for clock fall	25		ns
24	TdA (IORQf)	Address definite time prior to $\overline{\text{IORQ}}$ fall	**		ns
25	TdCr (IORQf)	Delay from clock rise to $\overline{\text{IORQ}}$ = "L"		40	ns

AC CHARACTERISTICS (continued)

Number	Symbol	Parameter	Min	Max	Unit
26	TdCf (IORQr)	Delay from clock fall to $\overline{\text{IORQ}}$ "H"		40	ns
27	TdD (WRF)	Data definite time prior to $\overline{\text{WR}}$ fall	**		ns
28	TdCf (WRF)	Delay from clock fall to $\overline{\text{WR}}$ = "L"		40	ns
29	TwWR	$\overline{\text{WR}}$ pulse width	**		ns
30	TdCf (WRr)	Delay from clock fall to $\overline{\text{WR}}$ = "H"		40	ns
31	TdD (WRf)	Data definite time prior to $\overline{\text{WR}}$ fall	**		ns
32	TdCr (WRf)	Delay from clock rise to $\overline{\text{WR}}$ = "L"		40	ns
33	TdWRr (D)	Output data holding after $\overline{\text{WR}}$ = "H"	**		ns
34	TdCf (HALT)	Delay from clock fall to $\overline{\text{HALT}}$ = "L" or "H"		100	ns
35	TwNMI	$\overline{\text{NMI}}$ pulse width	60		ns
36	TsBUSREQ (Cr)	Set-up time for clock rise	35		ns
37*	ThBUSREQ (Cr)	$\overline{\text{BUSREQ}}$ hold time after clock rise	0		ns
38	TdCr (BUSACKf)	Time from clock rise to $\overline{\text{BUSACK}}$ = "L"		40	ns
39	TdCf (BUSACKr)	Time from clock fall to $\overline{\text{BUSACK}}$ = "H"		40	ns
40	TdCr (Dz)	Delay from clock rise to data bus float state		40	ns
41	TdCr (CTz)	Delay from clock rise to control output float state ($\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$)		40	ns
42	TdCr (Az)	Delay from clock rise to address bus float state		50	ns
43	TdCTr (A)	Address hold time from $\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$, $\overline{\text{RD}}$, or $\overline{\text{WR}}$	**		ns
44	TsRESET (Cr)	$\overline{\text{RESET}}$ set-up time for clock rise	30		ns
45*	ThRESET (Cr)	$\overline{\text{RESET}}$ hold time for clock rise	0		ns
46	TsINTf (Cr)	$\overline{\text{INT}}$ set-up time for clock rise	50		ns
47*	ThINTr (Cr)	$\overline{\text{INT}}$ hold time after clock rise	0		ns
49	TdCf (IORQf)	Delay from clock fall to $\overline{\text{IORQ}}$ = "L"		40	ns
50	TdCr (IORQr)	Delay from clock rise to $\overline{\text{IORQ}}$ = "H"		40	ns
51	TdCf (D)	Delay from clock fall to data output		80	ns

AC CHARACTERISTICS (continued)

Number	Symbol	Parameter	Min	Max	Unit
52	TRST1S	Clock (CLK) restart time by $\overline{INT}$ (STOP mode)		(typ) (2^{14}+2.5)xTcC	ns
53	TRST2S	Clock (CLK) restart time by $\overline{NMI}$ (STOP mode)		(typ) (2^{14}+2.5)xTcC	ns
54	TRST1I	Clock (CLK) restart time by $\overline{INT}$ (IDLE1/2 mode)		(typ) 2.5 TcC ns	
55	TRST2I	Clock (CLK) restart time by $\overline{NMI}$ (IDLE1/2 mode)		(typ) 2.5 TcC ns	

* Test conditions are: CL = 100 pf

** **NOTES: AC Characteristics** (per line item number).

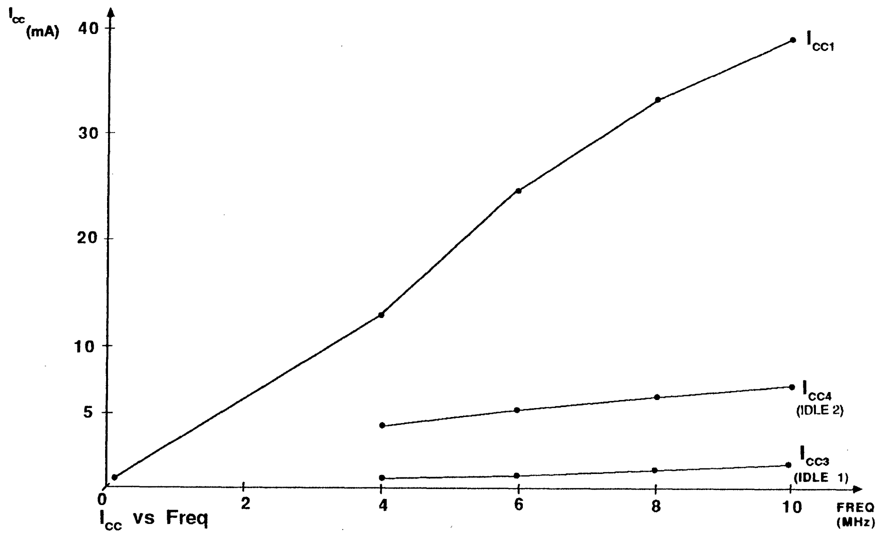
Number	Symbol	General Parameter
1	TcC	TwCh + TwCl + TrC + TfC
7	TdA(MREQf)	TwCh + TfC - 45
10	TwMREQh	TwCh + TfC - 25
11	TwMREQl	TcC - 30
24	TdA(IORQf)	TcC - 50
27	TdD(WRf)	TcC - 100
29	TwWR	TcC - 25
31	TdD (WRf)	TwCl + TrC - 100
33	TdWRr (D)	TwCl + TrC - 50
43	TdCTr (A)	TwCl + TrC - 45
48	TdM1f (IORQf)	2TcC +TwCh + TfC - 45

AC Test Conditions: V_{IH} = 3V V_{OH} = 2V VI_{HC} = V_{CC} -0.6V FLOAT = +-0.5V
V_{IL} = .5V V_{OL} = .8V V_{ILC} = .5V V_{CC} = 4.5 to 5.5V

DC CHARACTERISTICS VCC = 5.0 V +-10%

Symbol	Parameter	Min	Max	Unit	Condition
V_{OHC}	Clock Output High Voltage	VCC-0.6			-2mA
V_{OLC}	Clock Output Low Voltage		0.4	V	+2mA
V_{IH}	Input High Voltage	2.2	VCC	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	
V_{OL}	Output Low Voltage		0.4[5]	V	I_{OL} = 2.0mA
V_{OH1}	Output High Voltage	2.4[5]		V	I_{OH} = -1.6mA
V_{OH2}	Output High Voltage	VCC-0.8[5]		V	I_{OH} = -250uA
I_{CC1}[1]	Power Supply Current 10 MHz		50	mA	V_{CC} = 5V $V_{IIH} = V_{CC}$ - 0.2V V_{IIL} = 0.2V XTALIN = 10 MHz
I_{CC2}[1,3]	Power Supply Current (STOP Mode)		10	uA	V_{CC} = 5V
I_{CC3}[1]	Power Supply Current (IDLE1 Mode)		4	mA	$V_{IH} = V_{CC}$ - 0.2V V_{IL} = 0.2V XTALIN = 10 MHz
I_{CC4}[1]	Power Supply Current (IDLE2 Mode)		15	mA	$V_{IH} = V_{CC}$ - 0.2V V_{IL} = 0.2V XTALIN = 10 MHz
I_{LI}	Input Leakage Current	-10	10[4]	uA	V_{IN} = 0.4 to V_{CC}
I_{LO}	3-State Output Leakage Current in Float	-10	10[2]	uA	V_{OUT} = 0.4 to V_{CC}

1. Measurements made with outputs floating.
2. A_{15}-A_0, D_7-D_0, $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, and $\overline{WR}$.
3. I_{CC2} standby current is guaranteed when the halt pin is low in STOP mode.
4. All pins except XTAL1, where I_{LI} = +/- 25 uA.
5. A_{15}- A_0, D_7-D_0, $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, $\overline{WR}$, $\overline{HALT}$, $\overline{M1}$, and $\overline{BUSACK}$.



CONDITIONS: $V_{IHC}=V_{IH}=V_{CC}$-0.2V V_{CC} = 5V, TEMP= 0°C to 70°C $V_{ILC}=V_{IL}$ = 0.2V

Figure 26 Z84C01 Typical I_{CC} vs Freq

ELECTRICAL CHARACTERISTICS:

ABSOLUTE MAXIMUM RATINGS

Voltage on Vcc with respect to Vss..........-0.3V to + 7V
Voltages on all inputs with respect to Vss..-0.3V to Vcc + 0.3V
Operating Ambient
Temperature.......................See Ordering Information
Storage Temperature...............-65°C to + 150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Standard Test Conditions

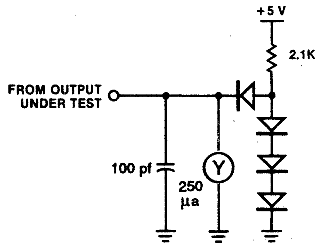
The DC Characteristics and capacitance sections below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND (OV). Positive current flows into the referenced pin.

Available operating temperature ranges are:

S = 0°C to 70°C
Voltage Supply Range: +4.50V < Vcc < + 5.50V

All AC parameters assume a load capacitance of 100 pf. Add 10 ns delay for each 50 pf increase in load up to a maximum of 150 pf for the data bus and 100 pf for address and control lines. AC timing measurements are referenced to 1.5 volts (except for clock, which is referenced to the 10% and 90% points). Maximum capacitive load for CLK is 125 pf.

The Ordering Information section lists temperature ranges and product numbers. Package drawings are in the Package Information section. Refer to the Literature List for additional documentation.



Zilog **Product Specification**

January 1989

Z8410/Z84C10 NMOS/CMOS Z80® DMA Direct Memory Access Controller

FEATURES

- Transfers, searches, and search/transfers in Byte-at-a-Time, Burst, or Continuous modes. Cycle length and edge timing can be programmed to match the speed of any port.
- Dual port addresses (source and destination) generated for memory-to-I/O, memory-to-memory, or I/O-to-I/O operations. Addresses may be fixed or automatically incremented/decremented.
- Next-operation loading without disturbing current operations via buffered starting-address registers. An entire previous sequence can be repeated automatically.
- Extensive programmability of functions. CPU can read complete channel status.
- **NMOS version for high cost performance solutions**
- **CMOS version for the designs requires low power consumption**
- **NMOS Z0841004 - 4MHz**
- **CMOS Z84C1006 - DC 4 MHz to 6.17 MHz, Z84C1008 - DC to 8 MHz**
- **6 MHz version supports 6.144 MHz CPU clock operation clock.**
- Standard Z80 Family bus-request and prioritized interrupt-request daisy chains implemented without external logic. Sophisticated, internally modifiable interrupt vectoring.
- Direct interfacing to system buses without external logic.

GENERAL DESCRIPTION

The Z80 DMA (Direct Memory Access), hereafter referred to as Z80 DMA or DMA, is a powerful and versatile device for controlling and processing transfers of data. Its basic function of managing CPU-independent transfers between two ports is augmented by an array of features that optimize transfer speed and control with little or no external logic in systems using an 8- or 16-bit data bus and a 16-bit address bus.

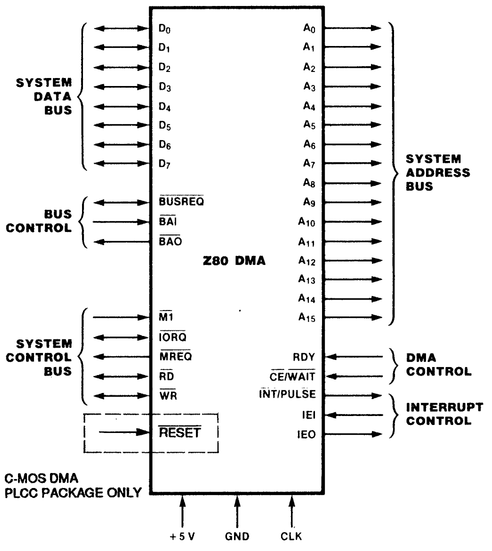


Figure 1. Pin Functions

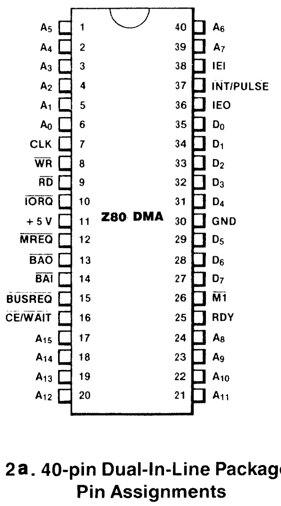


Figure 2a. 40-pin Dual-In-Line Package (DIP), Pin Assignments

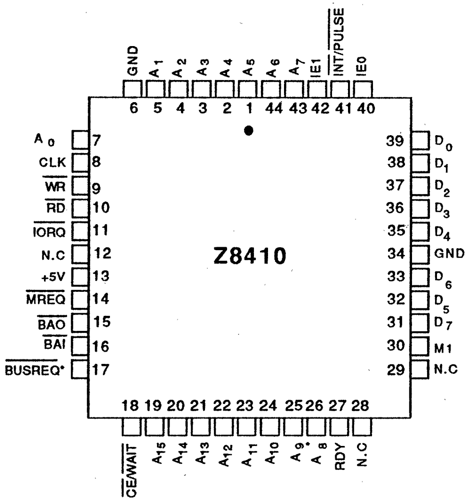


Figure 2b. Z8410 NMOS Z80 DMA
44-Pin PLCC Pinout

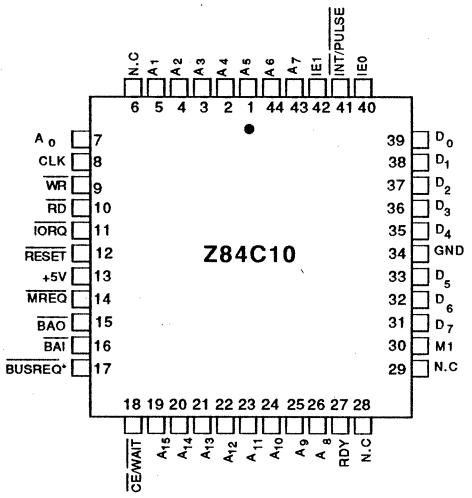


Figure 2c. Z84C10 CMOS Z80 DMA
PLCC Pinout

Transfers can be done between any two ports (source and destination), including memory-to-I/O, memory-to-memory, and I/O-to-I/O. Dual port addresses are automatically generated for each transaction and may be either fixed or incrementing/decrementing. In addition, bit-maskable byte searches can be performed either concurrently with transfers or as an operation in itself.

The Z80 DMA contains direct interfacing to, and independent control of, system buses, as well as sophisticated bus and interrupt controls. Many programmable features, including variable cycle timing and auto-restart, minimize CPU software overhead. They are especially useful in adapting this special-purpose transfer processor to a broad variety of memory, I/O and CPU environments.

The Z80 DMA is packaged in a 40-pin plastic or Cerdip DIP, or 44-pin PCC. It uses a single +5V power supply and the standard Z80 Family single-phase clock.

FUNCTIONAL DESCRIPTION

Classes of Operation. The Z80 DMA has three basic classes of operation:

- Transfers of data between two ports (memory or I/O peripheral)
- Searches for a particular 8-bit maskable byte at a single port in memory or an I/O peripheral
- Combined transfers with simultaneous search between two ports

Figure 4 illustrates the basic functions served by these classes of operation.

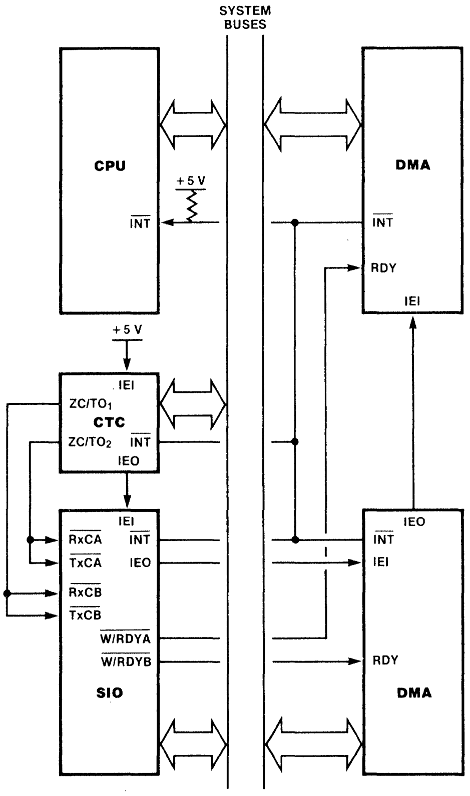


Figure 3. Typical Z80 Environment

During a transfer, the DMA assumes control of the system address and data buses. Byte by byte, data is read from one addressable port and written to the other addressable port. The ports may be programmed to be either system main memory or peripheral I/O devices. Thus, a block of data may be written from one peripheral to another, from one area of main memory to another, or from a peripheral to main memory and vice versa.

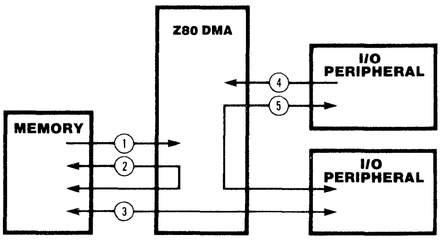
During a search-only operation, data is read from the source port and compared byte by byte with a DMA-internal register containing a programmable match byte. This match byte may optionally be masked so that only certain bits within the match byte are compared. Search rates up to 2M bytes per second can be obtained with the 4 MHz Z80 DMA.

In combined searches and transfers, data is transferred between two ports while simultaneously searching for a bit-maskable byte match.

Data transfers or searches can be programmed to stop, or interrupt, under various conditions. In addition, CPU-readable status bits can be programmed to reflect the condition.

Modes of Operation. The Z80 DMA can be programmed to operate in one of three transfer and/or search modes:

- *Byte-at-a-Time:* data operations are performed one byte at a time. Between each byte operation the system buses are released to the CPU. The buses are requested again for each succeeding byte operation.



1. Search memory
2. Transfer memory-to-memory (optional search)
3. Transfer memory-to-I/O (optional search)
4. Search I/O
5. Transfer I/O-to-I/O (optional search)

Figure 4. Basic Functions of the Z80 DMA

- *Burst:* data operations continue until a port's Ready line to the DMA goes inactive. The DMA then stops and releases the system buses after completing its current byte operation.
- *Continuous:* data operations continue until the end of the programmed block of data is reached before the system buses are released. If a port's Ready line goes inactive before this occurs, the DMA simply pauses until the Ready line comes active again.

In all modes, once a byte of data is read into the DMA, the operation on the byte will be completed in an orderly fashion, regardless of the state of other signals (including a port's Ready line).

Due to the DMA's high-speed buffered method of reading data, operations on one byte are not completed until the next byte is read in. This means that total transfer or search block lengths must be two or more bytes, and that block lengths programmed into the DMA must be one byte less than the desired block length (count is N-1 where N is the block length).

Commands and Status. The Z80 DMA has several writable control registers and readable status registers available to the CPU. Control bytes can be written to the DMA whenever the DMA is not controlling the system buses, but the act of writing a control byte to the DMA disables the DMA until it is again enabled by a specific command. Status bytes can also be read at any such time, but writing the Read Status Byte command or the Initiate Read Sequence command disables the DMA.

Control bytes to the DMA include those which affect immediate command actions such as enable, disable, reset, load starting-address buffers, continue, clear counters, and clear status bits. In addition, many mode-setting control bytes can be written, including mode and class of operation, port configuration, starting addresses, block length, address counting rule, match and match-mask byte, interrupt conditions, interrupt vector, status-affects-vector condition, pulse counting, auto restart, Ready-line and Wait-line rules, and read mask.

Readable status registers include a general status byte reflecting Ready-line, end-of-block, byte-match, and interrupt conditions, as well as 2-byte registers for the current byte count, Port A address, and Port B address.

Variable Cycle. The Z80 DMA has the unique feature of programmable operation-cycle length. This is valuable in tailoring the DMA to the particular requirements of other system components (fast or slow) and maximizes the data-transfer rate. It also eliminates external logic for signal conditioning.

There are two aspects to the variable cycle feature. First, the entire read and write cycles (periods) associated with the source and destination ports can be independently programmed as 2, 3, or 4 T-cycles long (more if Wait cycles are used), thereby increasing or decreasing the speed with which all DMA signals change (Figure 5).

Second, the four signals in each port specifically associated with transfers of data (I/O Request, Memory Request, Read and Write) can each have its active trailing edge terminated one-half T-cycle early. This adds a further dimension of flexibility and speed, allowing such things as shorter-than-normal Read or Write signals that go inactive before data starts to change.

Address Generation. Two 16-bit addresses are generated by the Z80 DMA for every transfer operation, one address for the source port and another for the destination port. Each address can be either variable or fixed. Variable addresses can increment or decrement from the programmed starting address. The fixed-address capability eliminates the need for separate enabling wires to I/O ports.

Port addresses are multiplexed onto the system address bus, depending on whether the DMA is reading the source port or writing to the destination port. Two readable address counters (2 bytes each) keep the current address of each port.

Auto Restart. The starting addresses of either port can be reloaded automatically at the end of a block. This option is selected by the Auto Restart control bit. The byte counter is cleared when the addresses are reloaded.

The Auto Restart feature relieves the CPU of software overhead for repetitive operations such as CRT refresh and many others. Moreover, when the CPU has access to the buses during byte-at-a-time or burst transfers, different starting addresses can be written into buffer registers during transfers, causing the Auto Restart to begin at a new location.

Interrupts. The Z80 DMA can be programmed to interrupt the CPU on four conditions:

- Interrupt on Ready (before requesting bus)
- Interrupt on Match
- Interrupt on End of Block
- Interrupt on Match and End of Block

Any of these interrupts causes an interrupt-pending status bit to be set, and each of them can optionally alter the DMA's interrupt vector. Due to the buffered constraint mentioned under "Modes of Operation," interrupts on Match at End of Block are caused by matches to the byte just prior to the last byte in the block.

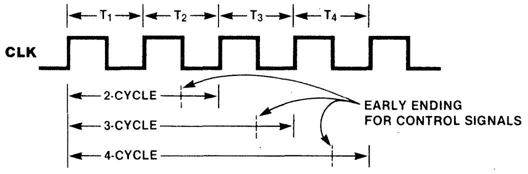


Figure 5. Variable Cycle Length

The DMA shares the Z80 Family's elaborate interrupt scheme, which provides fast interrupt service in real-time applications. In a Z80 CPU environment, the DMA passes its internally modifiable 8-bit interrupt vector to the CPU, which adds an additional eight bits to form the memory address of the interrupt-routine table. This table contains the address of the beginning of the interrupt routine itself. In this process, CPU control is transferred directly to the interrupt routine, so that the next instruction executed after an interrupt acknowledge is the first instruction of the interrupt routine itself.

Pulse Generation. External devices can keep track of how many bytes have been transferred by using the DMA's pulse output, which provides a signal at 256-byte intervals. The interval sequence may be offset at the beginning by 1 to 255 bytes.

The Interrupt line outputs the pulse signal in a manner that prevents misinterpretation by the CPU as an interrupt request, since it only appears when the Bus Request and Bus Acknowledge lines are both active.

PIN DESCRIPTION

A_0-A_{15}. *System Address Bus* (output, 3-state). Addresses generated by the DMA are sent to both source and destination ports (main memory or I/O peripherals) on these lines.

$\overline{\text{BAI}}$. *Bus Acknowledge In* (input, active Low). Signals that the system buses have been released for DMA control. In multiple-DMA configurations, the $\overline{\text{BAI}}$ pin of the highest priority DMA is normally connected to the Bus Acknowledge pin of the CPU. Lower-priority DMAs have their $\overline{\text{BAI}}$ connected to the $\overline{\text{BAO}}$ of a higher-priority DMA.

$\overline{\text{BAO}}$. *Bus Acknowledge Out* (output, active Low). In a multiple-DMA configuration, this pin signals that no other higher-priority DMA has requested the system buses. $\overline{\text{BAI}}$ and $\overline{\text{BAO}}$ form a daisy chain for multiple-DMA priority resolution over bus control.

$\overline{\text{BUSREQ}}$. *Bus Request* (bidirectional, active Low, open-drain). As an output, it sends requests for control of the system address bus, data bus, and control bus to the CPU. As an input when multiple DMAs are strung together in a priority daisy chain via $\overline{\text{BAI}}$ and $\overline{\text{BAO}}$, it senses when another DMA has requested the buses and causes this DMA to refrain from bus requesting until the other DMA is finished. Because it is a bidirectional pin, there cannot be any buffers between this DMA and any other DMA. It can, however, have a buffer between it and the CPU because it is unidirectional into the CPU. A pull-up resistor is connected to this pin.

$\overline{\text{CE}}/\overline{\text{WAIT}}$. *Chip Enable and Wait* (input, active Low). Normally this functions only as a $\overline{\text{CE}}$ line, but it can also be programmed to serve a $\overline{\text{WAIT}}$ function. As a $\overline{\text{CE}}$ line from the CPU, it becomes active when $\overline{\text{WR}}$ or $\overline{\text{RD}}$ and $\overline{\text{IORQ}}$ are active and the I/O port address on the system address bus is the DMA's address, thereby allowing a transfer of control, command bytes from the CPU to the DMA, or status bytes from the DMA to the CPU. As a $\overline{\text{WAIT}}$ line from memory or I/O devices, after the DMA has received a bus-request acknowledge from the CPU, it causes wait states to be inserted in the DMA's operation cycles thereby slowing the DMA to a speed that matches the memory or I/O device.

CLK. *System Clock* (input). Standard Z80 single-phase clock.

D_0-D_7. *System Data Bus* (bidirectional, 3-state). Commands from the CPU, DMA status, and data from memory or I/O peripherals are transferred on these lines.

IEI. *Interrupt Enable In* (input, active High). This is used with IEO to form a priority daisy chain when there is more than one interrupt-driven device. A High on this line indicates that no other device of higher priority is being serviced by a CPU interrupt service routine.

IEO. *Interrupt Enable Out* (output, active High). IEO is High only if IEI is High and the CPU is not servicing an interrupt from this DMA. Thus, this signal blocks lower-priority devices from interrupting while a higher-priority device is being serviced by its CPU interrupt service routine.

$\overline{\text{INT}}/\overline{\text{PULSE}}$. *Interrupt Request* (output, active Low, open-drain). While the CPU is the bus master, this output requests a CPU interrupt. The CPU acknowledges the interrupt by pulling its $\overline{\text{IORQ}}$ output Low during an $\overline{\text{M1}}$ cycle. It is typically connected to the $\overline{\text{INT}}$ pin of the CPU with a pullup resistor and tied to all other $\overline{\text{INT}}$ pins in the system. This pin can also be used to generate periodic pulses to an external device when the DMA is bus master (i.e., the CPU's $\overline{\text{BUSREQ}}$ and $\overline{\text{BUSACK}}$ lines are both Low and the CPU cannot see interrupts). While the DMA is the bus master, this output can be programmed to pulse each time 256 transfers have occurred.

$\overline{\text{IORQ}}$. *Input/Output Request* (bidirectional, active Low, 3-state). As an input, this indicates that the lower half of the address bus holds a valid I/O port address for transfer of control or status bytes from or to the CPU, respectively; this DMA is the addressed port if its $\overline{\text{CE}}$ pin and its $\overline{\text{WR}}$ or $\overline{\text{RD}}$ pins are simultaneously active. As an output, after the DMA has taken control of the system buses, it indicates that the lower half of the address bus holds a valid port address for another I/O device involved in a DMA transfer of data. When $\overline{\text{IORQ}}$ and $\overline{\text{M1}}$ are both active simultaneously, an interrupt acknowledge is indicated.

$\overline{\text{M1}}$. *Machine Cycle One* (input, active Low). Indicates that the current CPU machine cycle is an instruction fetch. It is used by the DMA to decode the return-from-interrupt instruction (RETI, ED-4D) sent by the CPU. During two-byte instruction fetches, $\overline{\text{M1}}$ is active as each opcode byte is fetched. An interrupt acknowledge is indicated when both **$\overline{\text{M1}}$ and $\overline{\text{IORQ}}$ are active. On CMOS DMA, $\overline{\text{M1}}$ signal has another function. When $\overline{\text{M1}}$ occurs without an active $\overline{\text{RD}}$ or $\overline{\text{IORQ}}$ for at least two clock cycles, the DMA is reset.**

$\overline{\text{MREQ}}$. *Memory Request* (output, active Low, 3-state). This indicates that the address bus holds a valid address for a memory read or write operation. After the DMA has taken control of the system buses, it indicates a DMA transfer request from or to memory.

$\overline{\text{RD}}$. *Read* (bidirectional, active Low, 3-state). As an input, this indicates that the CPU wants to read status bytes from the DMA's read registers. As an output, after the DMA has taken control of the system buses, it indicates a DMA-controlled read from a memory or I/O port address.

$\overline{\text{RESET}}$. *Reset* (CMOS PLCC version only: input, active Low). A low on this line resets the DMA.

RDY. *Ready* (input, programmable active Low or High). This is monitored by the DMA to determine when a peripheral device associated with a DMA port is ready for a read or write operation. Depending on the mode of DMA operation (Byte, Burst, or Continuous), the RDY line indirectly controls DMA activity by causing the $\overline{\text{BUSREQ}}$ line to go Low or High.

$\overline{\text{WR}}$. *Write* (bidirectional, active Low, 3-state). As an input, this indicates that the CPU wants to write control or command bytes to the DMA write registers. As an output, after the DMA has taken control of the system buses, it indicates a DMA-controlled write to a memory or I/O port address.

INTERNAL STRUCTURE

The internal structure of the Z80 DMA includes driver and receiver circuitry for interfacing with an 8-bit system data bus, a 16-bit system address bus, and system control lines (Figure 6). In a Z80 CPU environment, the DMA can be tied directly to the analogous pins on the CPU (Figure 7) with no additional buffering, except for the $\overline{\text{CE}}/\overline{\text{WAIT}}$ line.

The DMA's internal data bus interfaces with the system data bus and services all internal logic and registers. Addresses generated from this logic for Ports A and B (source and destination) of the DMA's single transfer channel are multiplexed onto the system address bus.

Specialized logic circuits in the DMA are dedicated to the various functions of external bus interfacing, internal bus control, byte matching, byte counting, periodic pulse generation, CPU interrupts, bus requests, and address generation. A set of 21 writable control registers and seven readable status registers provides the means by which the CPU governs and monitors the activities of these logic circuits. All registers are eight bits wide, with double-byte information stored in adjacent registers. The two address counters (two bytes each) for Ports A and B are buffered by the two starting addresses.

The 21 writable control registers are organized into seven base-register groups, most of which have multiple registers. The base registers in each writable group contain both control/command bits and pointer bits that can be set to address other registers within the group. The seven readable status registers have no analogous second-level registers.

The registers are designated as follows, according to their base-register groups:

WR0-WR6—Write Register groups 0 through 6 (7 base registers plus 14 associated registers)

RR0-RR6—Read Registers 0 through 6

Writing to a register within a write-register group involves first writing to the base register, with the appropriate pointer bits set, then writing to one or more of the other registers within the group. All seven of the readable status registers are accessed sequentially according to a programmable mask contained in one of the writable registers. The section entitled Programming explains this in more detail.

A pipelining scheme is used for reading data in. The programmed block length is the number of bytes compared to the byte counter, which increments at the end of each cycle. In searches, data byte comparisons with the match byte are made during the read cycle of the next byte. Matches are, therefore, discovered only after the next byte is read in.

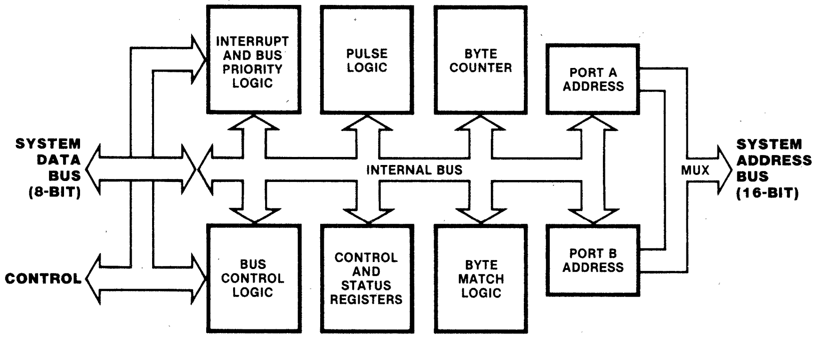


Figure 6. Block Diagram

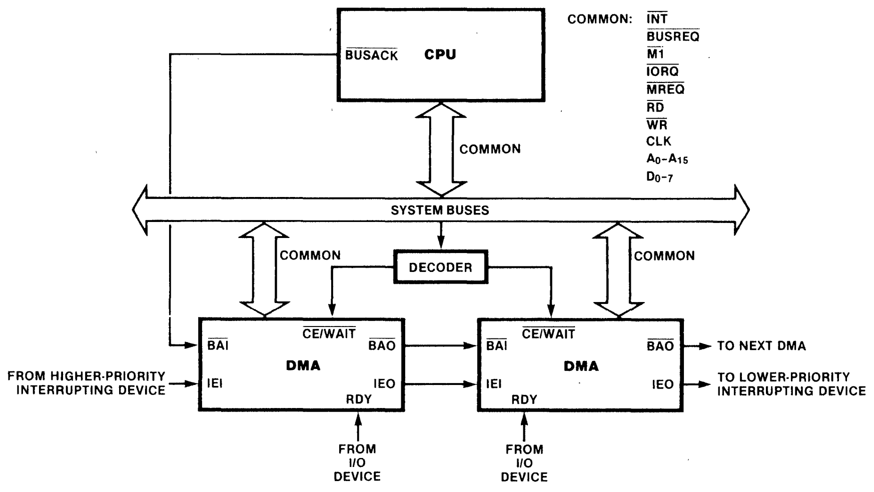


Figure 7. Multiple-DMA Interconnection to the Z80 CPU

In multiple-DMA configurations, interrupt-request daisy chains are prioritized by the order in which their IEI and IEO lines are connected. The system bus, however, may not be pre-empted. Any DMA that gains access to the system buses keeps them until it is finished.

Read Registers

RR0	Status byte
RR1	Byte counter (low byte)
RR2	Byte counter (high byte)
RR3	Port A address counter (low byte)
RR4	Port A address counter (high byte)
RR5	Port B address counter (low byte)
RR6	Port B address counter (high byte)

Write Registers

WR0	Base register byte
	Port A starting address (low byte)
	Port A starting address (high byte)
	Block length (low byte)
	Block length (high byte)
WR1	Base register byte
	Port A variable-timing byte
WR2	Base register byte
	Port B variable-timing byte
WR3	Base register byte
	Mask byte
	Match byte
WR4	Base register byte
	Port B starting address (low byte)
	Port B starting address (high byte)
	Interrupt control byte
	Pulse control byte
	Interrupt vector
WR5	Base register byte
WR6	Base register byte
	Read mask

PROGRAMMING

The Z80 DMA has two programmable fundamental states: (1) an enabled state, in which it can gain control of the system buses and direct the transfer of data between ports, and (2) a disabled state, in which it can initiate neither bus requests nor data transfers. When the DMA is powered up or reset by any means, it is automatically placed into the disabled state. Program commands can be written to it by the CPU in either state, but this automatically puts the DMA in the disabled state, which is maintained until an enable command is issued by the CPU. The CPU must program the DMA in advance of any data search or transfer by addressing it as an I/O port and sending a sequence of control bytes using an Output instruction (such as OTIR for the Z80 CPU).

Reading. (Figure 8a) The Read Registers (RR0-RR6) are read by the CPU by addressing the DMA as an I/O port using an Input instruction (such as INIR for the Z80 CPU). The readable bytes contain DMA status, byte-counter values, and port addresses since the last DMA reset. The registers are always read in a fixed sequence beginning with RR0 and ending with RR6. However, the register read in this sequence is determined by programming the Read Mask in WR6. The sequence of reading is initialized by writing an Initiate Read Sequence or Set Read Status command to WR6. After a Reset DMA, the sequence must be initialized with the Initiate Read Sequence command or a Read Status command. The sequence of reading all registers that are not excluded by the Read Mask register must be completed before a new Initiate Read Sequence or Read Status command.

Writing. Control or command bytes are written into one or more of the Write Register groups (WR0-WR6) by first writing to the base register byte in that group. All groups have base registers and most groups have additional associated registers. The associated registers in a group are sequentially accessed by first writing a byte to the base register containing register-group identification and pointer bits (1's) to one or more of that base register's associated registers.

This is illustrated in Figure 8b. In this figure, the sequence in which associated registers within a group can be written to is shown by the vertical position of the associated registers. For example, if a byte written to the DMA contains the bits that identify WR0 (bits D0, D1 and D7), and also contains 1's in the bit positions that point to the associated "Port A Starting Address (low byte)" and "Port A Starting Address (high byte)," then the next two bytes written to the DMA will be stored in that order in these two registers.

Fixed-Address Programming. A special circumstance arises when programming a destination port to have a fixed address. The load command in WR6 only loads a fixed address to a port selected as the source, not to a port selected as the destination. Therefore, a fixed destination address must be loaded by temporarily declaring it a fixed-source address and subsequently declaring the true source as such, thereby implicitly making the other a destination.

The following example illustrates the steps in this procedure, assuming that transfers are to occur from a variable-address source (Port A) to a fixed-address destination (Port B):

1. Temporarily declare Port B as source in WR0.
2. Load Port B address with LOAD command to WR6.
3. Declare Port A as a source in WR0.
4. Load Port A address with LOAD command to WR6.
5. Enable DMA in WR6.

Figure 9 illustrates a program to transfer data from memory (Port A) to a peripheral device (Port B). In this example, the Port A memory starting address is 1050_H and the Port B peripheral fixed address is 05_H. Note that the data flow is 1001_H bytes—one more than specified by the block length. The table of DMA commands may be stored in consecutive memory locations and transferred to the DMA with an output instruction such as the Z80 CPU's OTIR instruction.

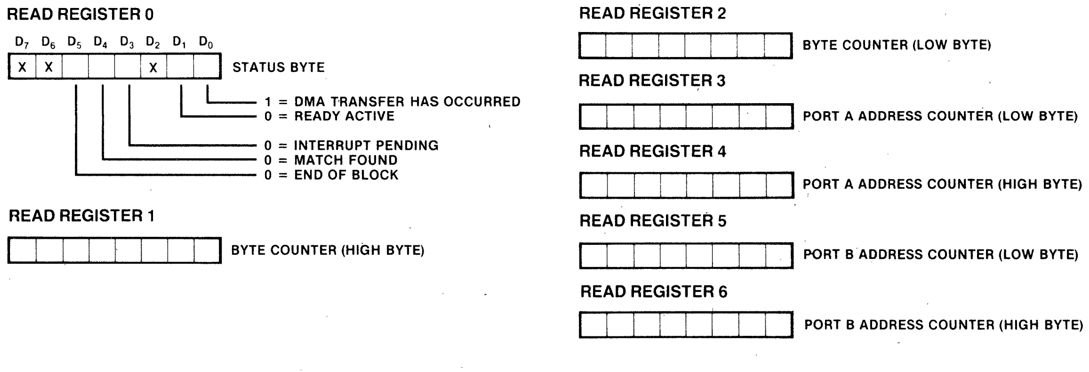


Figure 8a. Read Registers

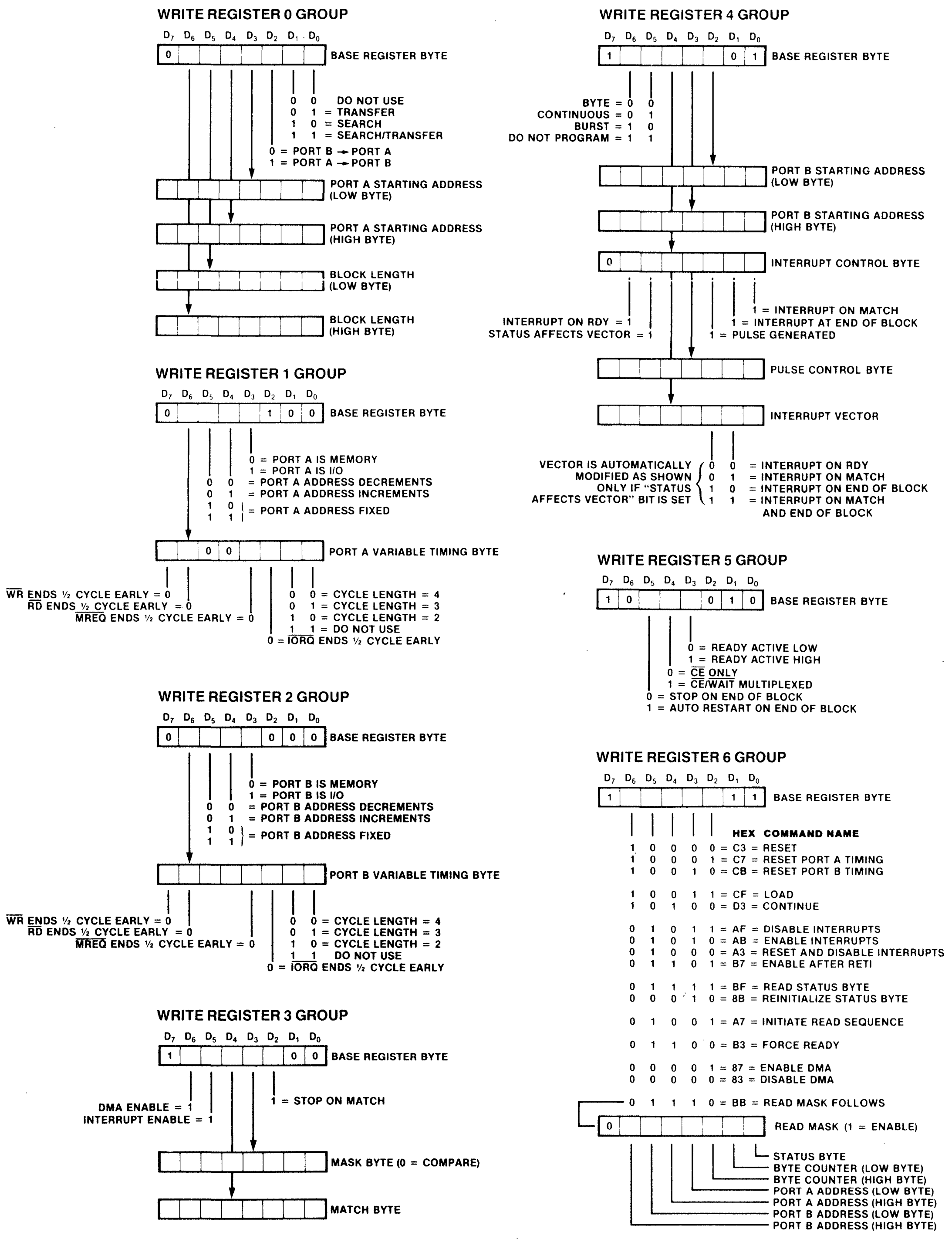


Figure 8b. Write Registers

Comments	D7	D6	D5	D4	D3	D2	D1	D0	HEX
WR0 sets DMA to receive block length, Port A starting address and temporarily sets Port B as source.	0	1 Block Length Upper Follows	1 Block Length Lower Follows	1 Port A Upper Address Follows	1 Port A Lower Address Follows	0 B ⟶ A Temporary for Loading B Address*	0 Transfer, No Search	1	79
Port A address (lower)	0	1	0	1	0	0	0	0	50
Port A address (upper)	0	0	0	1	0	0	0	0	10
Block length (lower)	0	0	0	0	0	0	0	0	00
Block length (upper)	0	0	0	1	0	0	0	0	10
WR1 defines Port A as memory with fixed incrementing address.	0	0 No Timing Follows	0 Address Changes	1 Address Increments	0 Port is Memory	1	0	0	14
WR2 defines Port B as peripheral with fixed address.	0	0 No Timing Follows	1 Fixed Address	0	1 Port is I/O	0	0	0	28
WR4 sets mode to Burst, sets DMA to expect Port B address.	1	1 Burst Mode	0	0 No Interrupt Control Byte Follows	0 No Upper Address	1 Port B Lower Address Follows	0	1	C5
Port B address (lower)	0	0	0	0	0	1	0	1	05
WR5 sets Ready active High.	1	0	0 No Auto Restart	0 No Wait States	1 RDY Active High	0	1	0	8A
WR6 loads Port B address and resets block counter.*	1	1	0	0	1	1	1	1	CF
WR0 sets Port A as source.*	0	0	0 No Address or Block Length Bytes	0	0	1 A ⟶ B	0 Transfer, No Search	1	05
WR6 loads Port A address and resets block counter.	1	1	0	0	1	1	1	1	CF
WR6 enables DMA to start operation.	1	0	0	0	0	1	1	1	87

NOTE: The actual number of bytes transferred is one more than specified by the block length.
*These entries are necessary only in the case of a fixed destination address.

Figure 9. Sample DMA Program

INACTIVE STATE TIMING

(DMA as CPU Peripheral)

In its disabled or inactive state, the DMA is addressed by the CPU as an I/O peripheral for write and read (control and status) operations. Write timing is illustrated in Figure 10.

Reading of the DMA's status byte, byte counter, or port address counters is illustrated in Figure 11. These operations require less than three T-cycles. The $\overline{CE}$, $\overline{IORQ}$, and $\overline{RD}$ lines are made active over two rising edges of CLK, and data appears on the bus approximately one T-cycle after they become active.

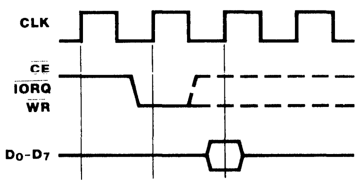


Figure 10. CPU-to-DMA Write Cycle

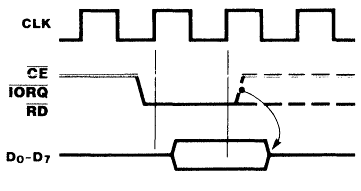


Figure 11. CPU-to-DMA Read Cycle

ACTIVE STATE TIMING

(DMA as Bus Controller)

Default Read and Write Cycles. By default, and after reset, the DMA's timing of read and write operations is exactly the same as the Z80 CPU's timing of read and write cycles for memory and I/O peripherals, with one exception: during a read cycle, data is latched on the falling edge of T_3 and held on the data bus across the boundary between read and write cycles, through the end of the following write cycle.

Figure 12 illustrates the timing for memory-to-I/O port transfers and Figure 13 illustrates I/O-to-memory transfers. Memory-to-memory and I/O-to-I/O transfer timings are simply permutations of these diagrams.

The default timing uses three T-cycles for memory transactions and four T-cycles for I/O transactions, which include one automatically inserted wait cycle (T_{WA}) between T_2 and T_3. If the $\overline{CE}/\overline{WAIT}$ line is programmed to act as a $\overline{WAIT}$ line during the DMA's active state, it is sampled on the falling edge of T_2 for memory transactions and the falling edge of T_{WA} for I/O transactions. If $\overline{CE}/\overline{WAIT}$ is Low during this time, another T-cycle is added, during which the

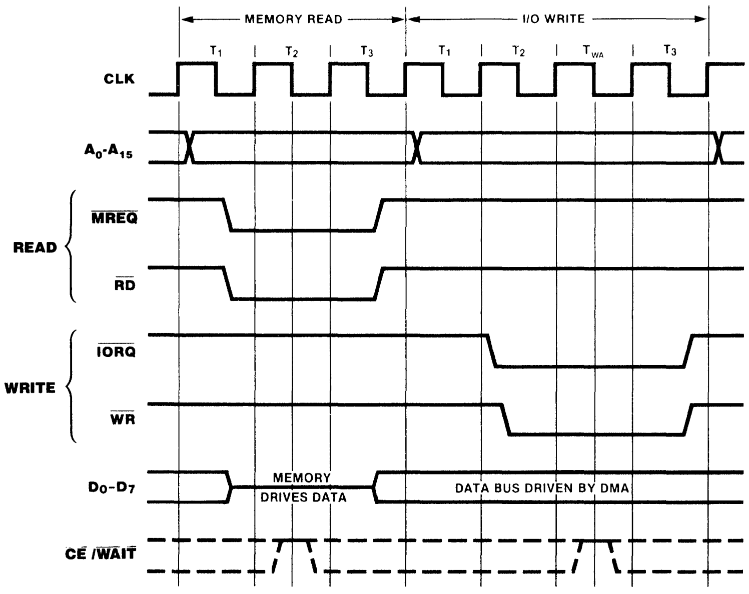


Figure 12. Memory-to-I/O Transfer

$\overline{CE}/\overline{WAIT}$ line will again be sampled. The duration of transactions can thus be indefinitely extended.

Variable Cycle and Edge Timing. The Z80 DMA's default operation-cycle length for the source (read) port and destination (write) port can be independently programmed. This variable-cycle feature allows read or write cycles consisting of two, three, or four T-cycles (more if Wait cycles are inserted), thereby increasing or decreasing the speed of all signals generated by the DMA. In addition, the trailing edges of the $\overline{IORQ}$, $\overline{MREQ}$, $\overline{RD}$, and $\overline{WR}$ signals can be independently terminated one-half cycle early. Figure 14 illustrates this.

In the variable-cycle mode, unlike default timing, $\overline{IORQ}$ comes active one-half cycle before $\overline{MREQ}$, $\overline{RD}$, and $\overline{WR}$. $\overline{CE}/\overline{WAIT}$ can be used to extend only the 3 or 4 T-cycle variable memory cycles and only the 4-cycle variable I/O cycle. The $\overline{CE}/\overline{WAIT}$ line is sampled at the falling edge of T_2 for 3- or 4-cycle memory cycles, and at the falling edge of T_3 for 4-cycle I/O cycles.

During transfers, data is latched on the clock edge causing the rising edge of $\overline{RD}$ and held until the end of the write cycle.

Bus Requests. Figure 15 illustrates the bus request and acceptance timing. The RDY line, which may be programmed active High or Low, is sampled on every rising edge of CLK. If it is found to be active and if the bus is not in use by any other device, the following rising edge of CLK drives $\overline{BUSREQ}$ Low. After receiving $\overline{BUSREQ}$, the CPU acknowledges on the $\overline{BAI}$ input either directly or through a multiple-DMA daisy chain. When a Low is detected on $\overline{BAI}$ for two consecutive rising edges of CLK, the DMA will begin transferring data on the next rising edge of CLK.

Bus Release Byte-at-a-Time. In Byte-at-a-Time mode, $\overline{BUSREQ}$ is brought High on the rising edge of CLK prior to the end of each read cycle (search-only) or write cycle (transfer and transfer/search) as illustrated in Figure 16. This is done regardless of the state of RDY. There is no possibility of confusion when a Z80 CPU is used since the CPU cannot

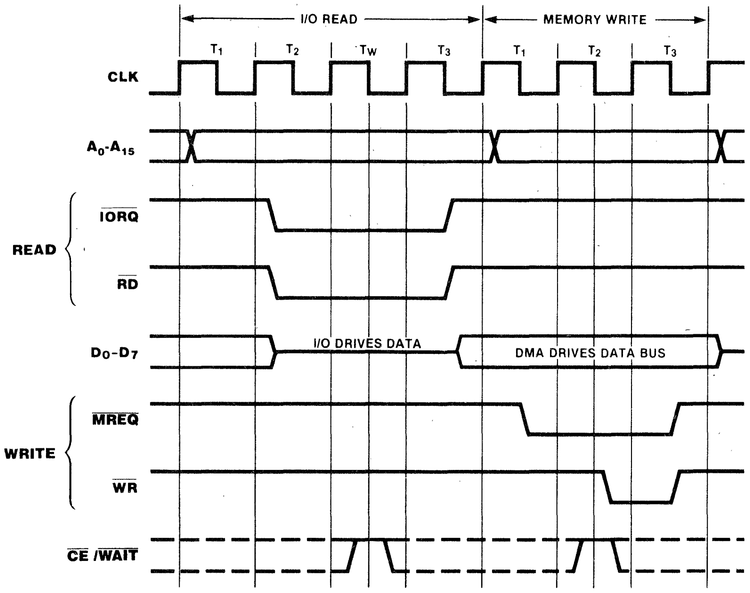


Figure 13. I/O-to-Memory Transfer

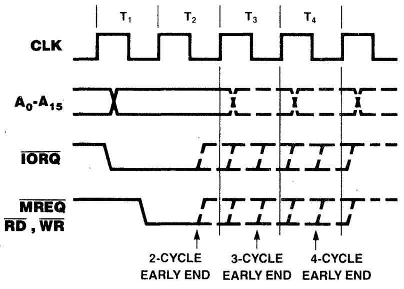


Figure 14. Variable-Cycle and Edge Timing

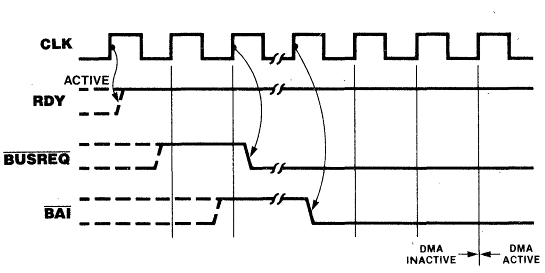


Figure 15. Bus Request and Acceptance

begin an operation until the following T-cycle. Most other CPUs are not bothered by this either, although note should be taken of it. The next bus request for the next byte will come after both $\overline{\text{BUSREQ}}$ and $\overline{\text{BAI}}$ have returned High.

Bus Release at End of Block. In Burst and Continuous modes, an end of block causes $\overline{\text{BUSREQ}}$ to go High, usually on the same rising edge of CLK in which the DMA completes the transfer of the data block (Figure 17). The last byte in the block is transferred even if RDY goes inactive before completion of the last byte transfer.

Bus Release on Not Ready. In Burst mode, when RDY goes inactive it causes $\overline{\text{BUSREQ}}$ to go High on the next rising edge of CLK after the completion of its current byte operation (Figure 18). The action on $\overline{\text{BUSREQ}}$ is thus somewhat delayed from action on the RDY line. The DMA always completes its current byte operation in an orderly fashion before releasing the bus.

By contrast, $\overline{\text{BUSREQ}}$ is not released in Continuous mode when RDY goes inactive. Instead, the DMA idles after completing the current byte operation, awaiting an active RDY again.

Bus Release on Match. If the DMA is programmed to stop on match in Burst or Continuous modes, a match causes $\overline{\text{BUSREQ}}$ to go inactive on the next DMA operation, i.e., at the end of the next read in a search or at the end of the following write in a transfer (Figure 19). Due to the pipelining scheme, matches are determined while the next DMA read or write is being performed.

The RDY line can go inactive after the matching operation begins without affecting this bus-release timing.

Interrupts. Timings for interrupt acknowledge and return from interrupt are the same as for the other Z80 peripherals.

Interrupt on RDY (interrupt before requesting bus) does not directly affect the $\overline{\text{BUSREQ}}$ line. Instead, the interrupt service routine must handle this by issuing the following commands to WR6:

1. Enable after Return From Interrupt (RETI) Command—Hex B7
2. Enable DMA—Hex 87
3. An RETI instruction that resets the Interrupt Under Service latch in the Z80 DMA.

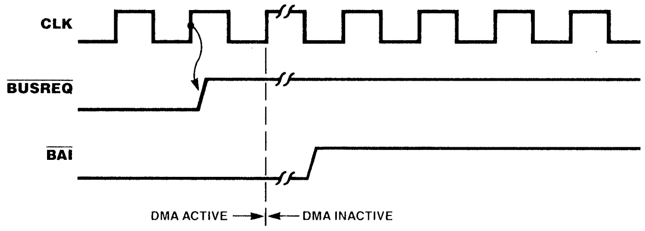


Figure 16. Bus Release (Byte-at-a-Time Mode)

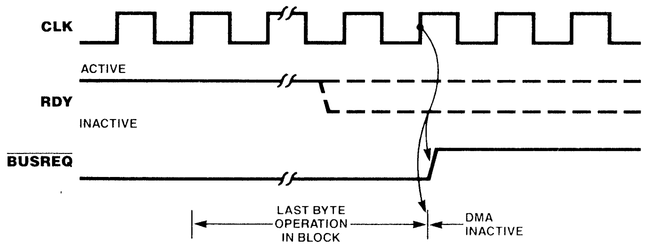


Figure 17. Bus Release at End of Block (Burst and Continuous Modes)

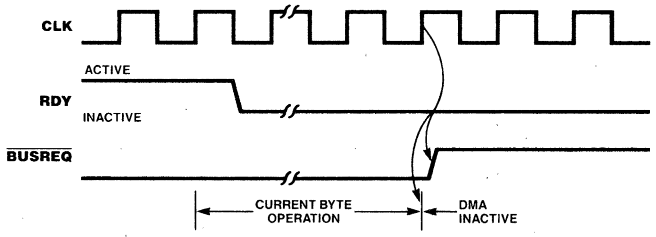


Figure 18. Bus Release When Not Ready (Burst Mode)

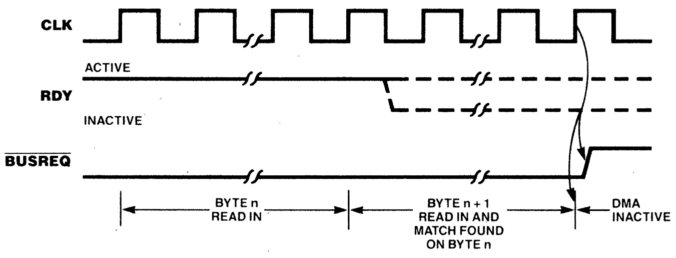


Figure 19. Bus Release on Match (Burst and Continuous Modes)

ABSOLUTE MAXIMUM RATINGS

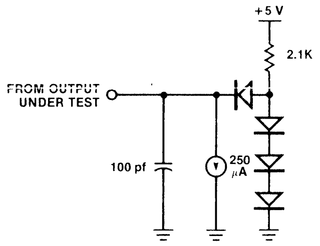
Voltages on V_{CC} with respect to V_{SS} . . . . . −0.3V to +7.0V
Voltages on all inputs with respect
to V_{SS} . . . . −0.3V to V_{CC} + 0.3V
Storage Temperature . . . . −65°C to +150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The characteristics below apply for the following test conditions, unless otherwise noted. All voltages are referenced to GND (0V). Positive current flows into the referenced pin. Available operating temperature range is:

- **S = 0°C to +70°C, V_{cc} Range**
  NMOS: =4.75V ≤ V_{cc} ≤+5.25V
  CMOS: +4.50V ≤ V_{cc}≤+5.50V
- **E = -40°C to 100°C, +4.50V ≤ V_{cc} ≤+5.50V**



DC CHARACTERISTICS (Z84C10 / CMOS Z80 DMA)

Symbol	Parameter	Min	Max	Typ	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	−0.3	+0.45		V	
V_{IHC}	Clock Input High Voltage	V_{CC}−0.6	V_{CC}+0.3		V	
V_{IL}	Input Low Voltage	−0.3	+0.8		V	
V_{IH}	Input High Voltage	+2.2	V_{CC}		V	
V_{OL}	Output Low Voltage		+0.4		V	I_{OL} = 2.0 mA
V_{OH1}	Output High Voltage	+2.4			V	I_{OH} = −1.6 mA
V_{OH2}	Output High Voltage	V_{CC}−0.8			V	I_{OH} = −250 μA
I_{LI}	Input Leakage Current		±10		μA	V_{IN} = 0.4 to V_{CC}
I_{LO}	3-State Output Leakage Current in Float		±10		μA	V_{OUT} = 0.4 to V_{CC}
ICC_1	Power Supply Current		**25/35**		mA	V_{CC} = 5V **CLK = 6/8MHz** V_{IHC} = V_{IH} = V_{CC} − 0.2V V_{ILC} = 0.2V
ICC_2	Standby Supply Current		10	0.5	μA	V_{CC} = 5V CLK = (0) V_{IHC} = V_{IH} = V_{CC} − 0.2V V_{ILC} = V_{IL} = 0.2V

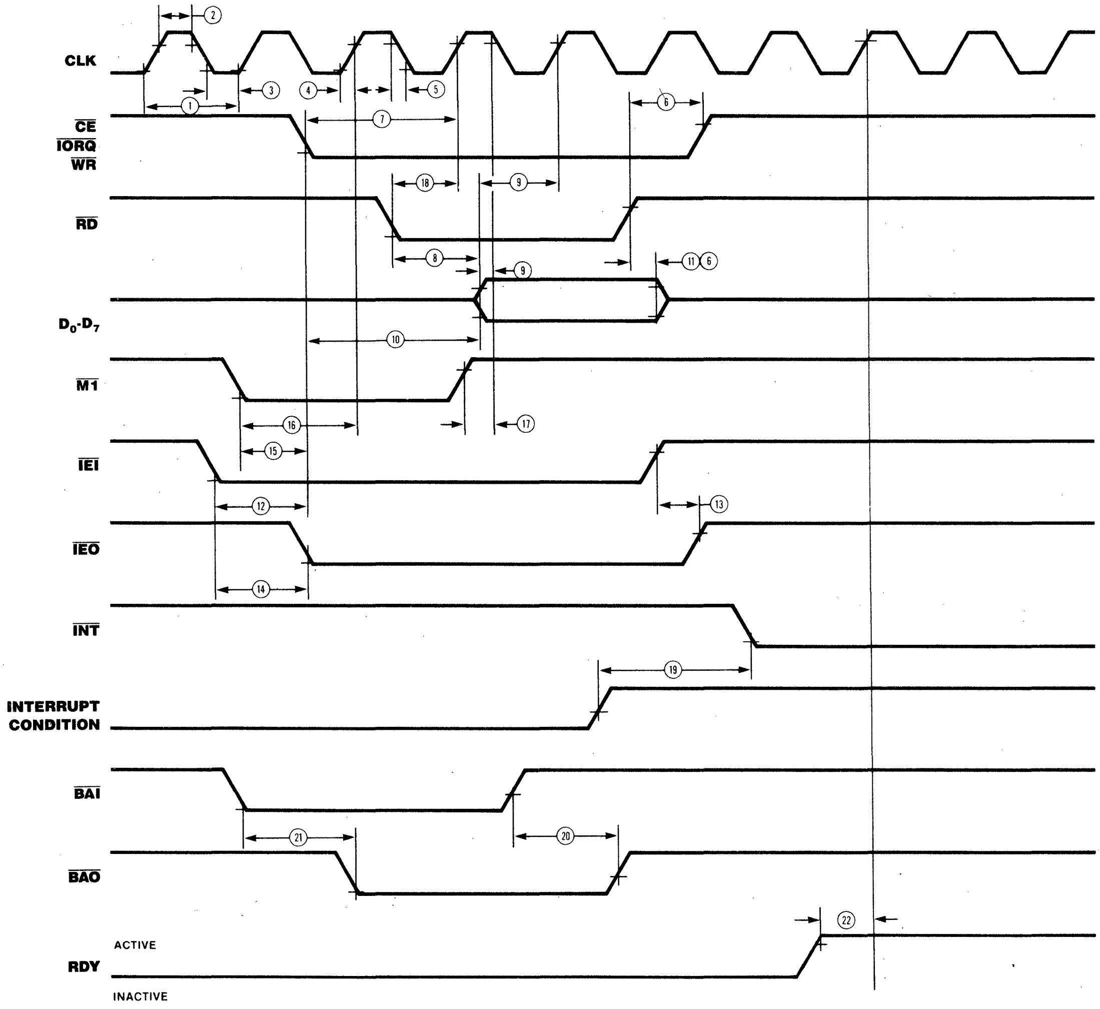
Over specified temperature and voltage range.

CAPACITANCE

Symbol	Parameter	Min	Max	Unit
C	Clock Capacitance		5	pf
C_{IN}	Input Capacitance		5	pf
C_{OUT}	Output Capacitance		10	pf

NOTES: Over specified temperature range; f = MHz.
Unmeasured pins returned to ground.

AC CHARACTERISTICS (Z84C10 / CMOS Z80 DMA)
(Inactive State)



NOTE: Signals in this diagram bear no relation to one another unless specifically noted as a numbered item.

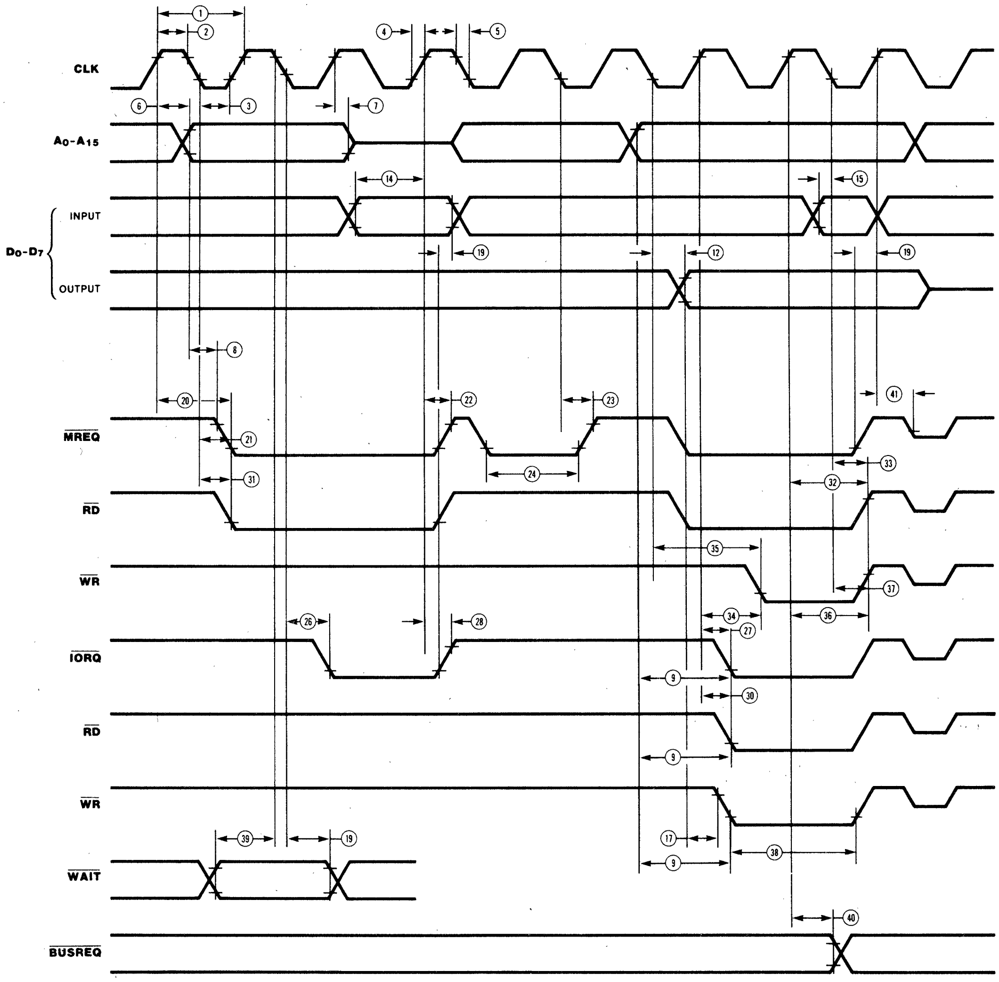
AC CHARACTERISTICS (Z8410 / NMOS Z80 DMA)
(Inactive State)

Number	Symbol	Parameter	Z0841004		Unit
			Min	Max	
1	TcC	Clock Cycle Time	250	4000	ns
2	TwCh	Clock Width (High)	110	2000	ns
3	TwCl	Clock Width (Low)	110	2000	ns
4	TrC	Clock Rise Time		30	ns
5	TfC	Clock Fall Time		30	ns
6	Th	Hold Time for Any Specified Setup Time	0		ns
7	TsC(Cr)	$\overline{\text{IORQ}}$, $\overline{\text{WR}}$, $\overline{\text{CE}}$ ↓ to Clock ↑ Setup	145		ns
8	TdDO(RDf)	$\overline{\text{RD}}$ ↓ to Data Output Delay		380	ns
9	TsDI(Cr)	Data In to Clock ↑ Setup ($\overline{\text{WR}}$ or $\overline{\text{M1}}$)	50		ns
10	TdDO(IOf)	$\overline{\text{IORQ}}$ ↓ to Data Out Delay (INTA Cycle)		160	ns
11	TdRDr(Dz)	$\overline{\text{RD}}$ ↑ to Data Float Delay (output buffer disable)		110	ns
12	TsIEI(IORQf)	IEI to $\overline{\text{IORQ}}$ ↓ Setup (INTA Cycle)	140		ns
13	TdIEOr(IEIr)	IEI ↑ to IEO ↑ Delay		160	ns
14	TdIEOf(IEIf)	IEI ↓ to IEO ↓ Delay		130	ns
15	TdM1f(IEOf)	$\overline{\text{M1}}$ ↓ to IEO ↓ Delay (interrupt just prior to $\overline{\text{M1}}$ ↓)		190	ns
16	TsM1f(Cr)	$\overline{\text{M1}}$ ↓ to Clock ↑ Setup	90		ns
17	TsM1r(Cf)	$\overline{\text{M1}}$ ↑ to Clock Setup	-10		ns
18	TsRDf(Cr)	$\overline{\text{RD}}$ ↓ to Clock ↑ Setup ($\overline{\text{M1}}$ Cycle)	115		ns
19	TdI(INTf)	Interrupt Cause to $\overline{\text{INT}}$ ↓ Delay ($\overline{\text{INT}}$ generated only when DMA is inactive)		500	ns
20	TdBAIr(BAOr)	$\overline{\text{BAI}}$ ↑ to $\overline{\text{BAO}}$ ↑ Delay		150	ns
21	TdBAIf(BAOf)	$\overline{\text{BAI}}$ ↓ to $\overline{\text{BAO}}$ ↓ Delay		150	ns
22	TsRDY(Cr)	RDY Active to Clock ↑ Setup	100		ns

NOTE: Negative minimum setup values mean that the first-mentioned event can come after the second-mentioned event.

AC CHARACTERISTICS (Z84C10 / CMOS Z80 DMA)

(Active State)



NOTE: Signals in this diagram bear no relation to one another unless specifically noted as a numbered item.

Number	Symbol	Parameter	Z84C1006°‡† Min(ns)	Max(ns)	Z84C1008°‡† Min(ns)	Max(ns)
1	TcC	Clock Cycle Time	162	DC	125	DC
2	TwCh	Clock Width (High)	65	DC	55	DC
3	TwCl	Clock Width (Low)	65	DC	55	DC
4	TrC	Clock Rise Time		20		10
5	TfC	Clock Fall Time		20		10

NOTES:

° **For clock periods other than the minimums shown, calculate parameters using the following table.**

‡ **Calculated values above assumed TrC = TfC = 20ns (6 MHz version) or 10ns (8 MHz version).**

† Data must be enabled onto data bus when $\overline{RD}$ is active.

* Parameter is not illustrated in the AC Timing Diagrams.

* **Z84C10 timing parameters are preliminary and subject to change.**

AC CHARACTERISTICS (Z84C10 / CMOS Z80 DMA)
(Active State)

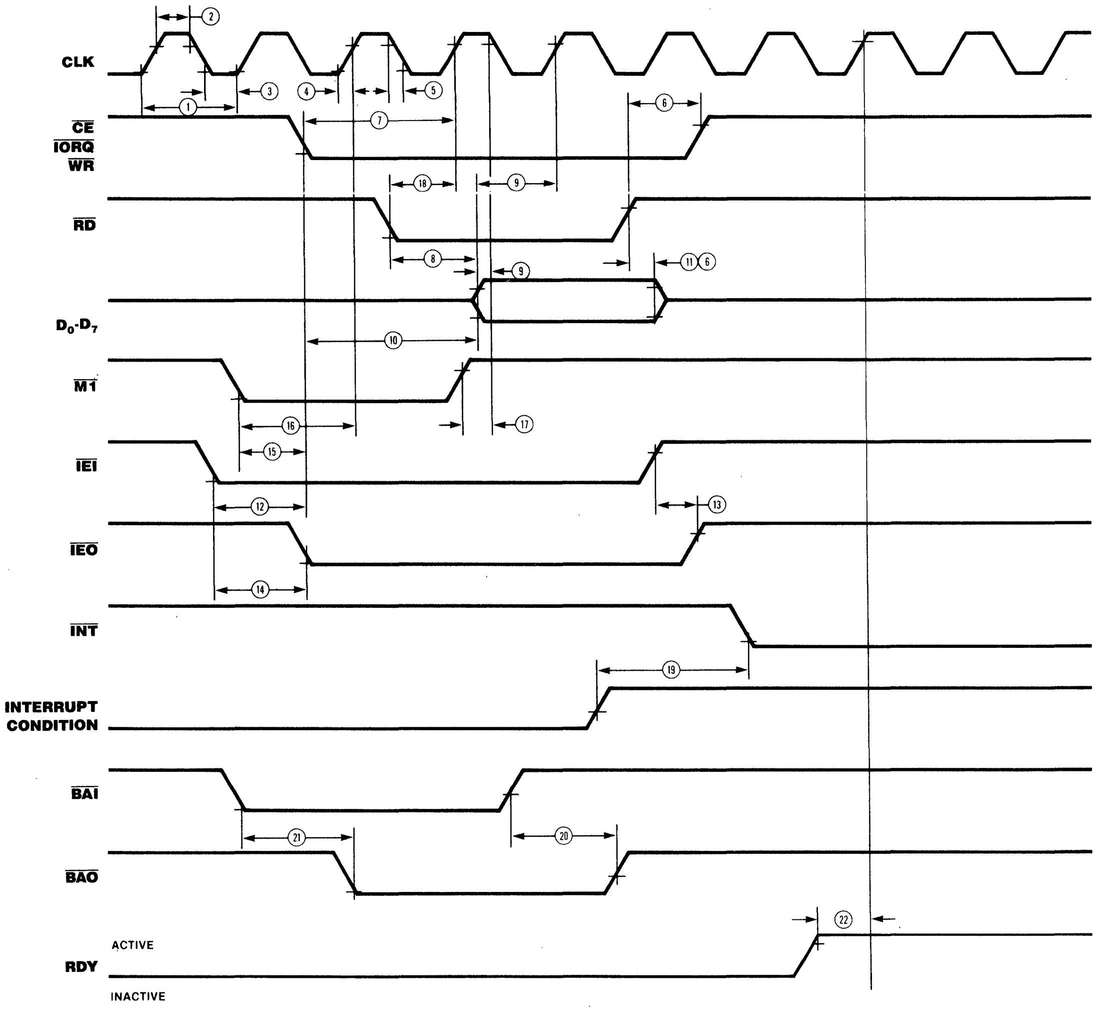
			Z84C1006		Z84C1008	
Number	Symbol	Parameter	Min(ns)	Max(ns)	Min(ns)	Max(ns)
6	TdA	Address Output Delay		90		70
7	TdC(Az)	Clock ↑ to Address Float Delay		80		70
8	TsA(MREQ)	Address to $\overline{\text{MREQ}}$ ↓ Setup (Memory Cycle)	35‡		35‡	
9	TsA(IRW)	Address Stable to $\overline{\text{IORQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$ ↓ Setup (I/O Cycle)	110‡		70‡	
*10	TdRW(A)	$\overline{\text{RD}}$, $\overline{\text{WR}}$ ↑ to Addr. Stable Delay	35‡		15‡	
*11	TdRW(Az)	$\overline{\text{RD}}$, $\overline{\text{WR}}$ ↑ to Addr. Float	60‡		45‡	
12	TdCf(DO)	Clock ↓ to Data Out Delay		130		110
*13	TdCr(Dz)	Clock ↑ to Data Float Delay (Write Cycle)		70		65
14	TsDI(Cr)	Data In to Clock ↑ Setup (Read cycle when rising edge ends read)	30		25	
15	TsDI(Cf)	Data In to Clock ↓ Setup (Read cycle when falling edge ends read)	40		30	
*16	TsDO(WfM)	Data Out to $\overline{\text{WR}}$ ↓ Setup (Memory Cycle)	25‡		5‡	
17	TsDO(WfI)	Data Out to $\overline{\text{WR}}$ ↓ Setup (I/O cycle)	55		40	
*18	TdWr(DO)	$\overline{\text{WR}}$ ↑ to Data Out Delay	30‡		10‡	
19	Th	Hold Time for Any Specified Setup Time	0		0	
20	TdCr(Mf)	Clock ↑ to $\overline{\text{MREQ}}$ ↓ Delay		70		60
21	TdCf(Mf)	Clock ↓ to $\overline{\text{MREQ}}$ ↓ Delay		70		60
22	TdCr(Mr)	Clock ↑ to $\overline{\text{MREQ}}$ ↑ Delay		70		60
23	TdCf(Mr)	Clock ↓ to $\overline{\text{MREQ}}$ ↑ Delay		70		60
24	TwM1	$\overline{\text{MREQ}}$ Low Pulse Width	135‡		95‡	
*25	TwMh	$\overline{\text{MREQ}}$ High Pulse Width	65‡		45‡	
26	TdCf(If)	Clock ↓ to $\overline{\text{IORQ}}$ ↓ Delay		70		60
27	TdCr(If)	Clock ↑ to $\overline{\text{IORQ}}$ ↓ Delay		65		55
28	TdCr(Ir)	Clock ↑ to $\overline{\text{IORQ}}$ ↑ Delay		70		60
*29	TdCf(Ir)	Clock ↓ to $\overline{\text{IORQ}}$ ↑ Delay		70		60
30	TdCr(Rf)	Clock ↑ to $\overline{\text{RD}}$ ↓ Delay		70		60
31	TdCf(Rf)	Clock ↓ to $\overline{\text{RD}}$ ↓ Delay		80		70
32	TdCr(Rr)	Clock ↑ to $\overline{\text{RD}}$ ↑ Delay		70		60
33	TdCf(Rr)	Clock ↓ to $\overline{\text{RD}}$ ↑ Delay		70		60
34	TdCr(Wf)	Clock ↑ to $\overline{\text{WR}}$ ↓ Delay		60		55
35	TdCf(Wf)	Clock ↓ to $\overline{\text{WR}}$ ↓ Delay		70		60
36	TdCr(Wr)	Clock ↑ to $\overline{\text{WR}}$ ↑ Delay		70		60
37	TdCf(Wr)	Clock ↓ to $\overline{\text{WR}}$ ↑ Delay		70		60
38	TwWl	$\overline{\text{WR}}$ Low Pulse Width	135‡		95‡	
39	TsWA(Cf)	$\overline{\text{WAIT}}$ to Clock ↓ Setup	60		50	
40	TdCr(B)	Clock ↑ to $\overline{\text{BUSREQ}}$ Delay		90		80
41	TdCr(Iz)	Clock ↑ to $\overline{\text{IORQ}}$, $\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$ Float Delay		70		70

NOTES:
‡ **All AC equations imply DMA default (standard) timing.**
† **Data must be enabled onto data bus when RD is active.**
* **Parameter is not illustrated in the AC Timing Diagrams.**
* **Numbers in parentheses are other parameter - numbers in this table;**
• **their values should be substituted in equations.**

FOOTNOTES TO AC CHARACTERISTICS

Number	Symbol	General Parameter	Z84C1006	Z84C1008
8	TsA(MREQ)	TwCh - TfC	-35	-30
9	TsA(IRW)	TcC	-55	-55
10	TdRW(A)	TwCl - TrC	-50	-50
11	TdRW(Z)	TwCl - TrC	-25	-20
16	TsDO(WfM)	TcC	-140	-120
18	TdWr(DO)	TwCl - TrC	-55	-55
24	TwM1	TcC	-30	-30
25	TwMh	TwCh - TfC	-20	-20
38	TwWl	TcC	-30	-30

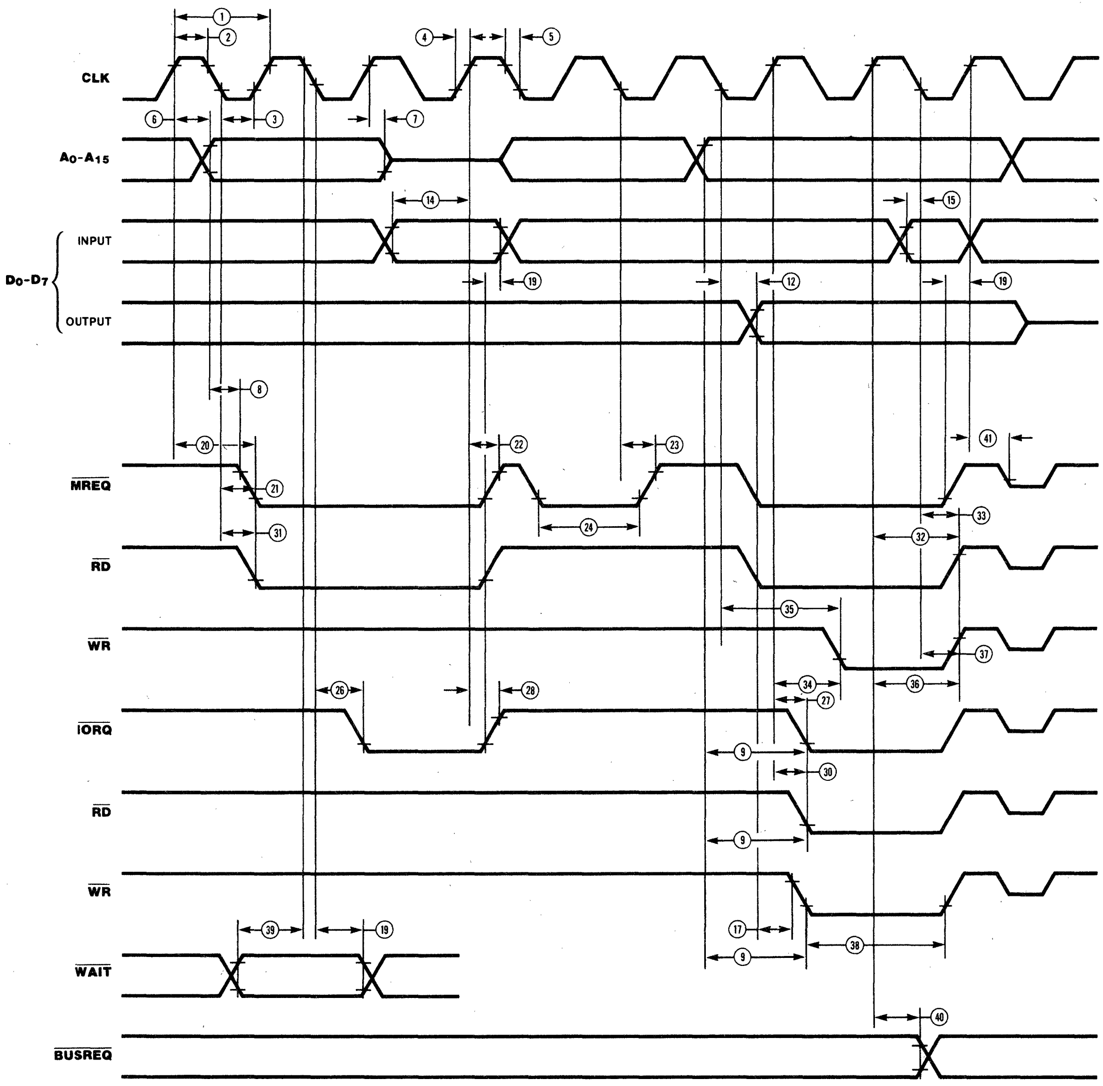
AC CHARACTERISTICS (Z8410 / NMOS Z80 DMA)
(Inactive State)



NOTE: Signals in this diagram bear no relation to one another unless specifically noted as a numbered item.

AC CHARACTERISTICS (Z8410 / NMOS Z80 DMA)

(Active State)



NOTE: Signals in this diagram bear no relation to one another unless specifically noted as a numbered item.

			Z0841004°‡†	
Number	**Symbol**	**Parameter**	**Min(ns)**	**Max(ns)**
1	TcC	Clock Cycle Time	250	
2	TwCh	Clock Width (High)	110	2000
3	TwCl	Clock Width (Low)	110	2000
4	TrC	Clock Rise Time		30
5	TfC	Clock Fall Time		30

NOTES:
° Numbers in parentheses are other parameter-numbers in this table; their values should be substituted in equations.
‡ All equations imply DMA default (standard) timing.
† Data must be enabled onto data bus when RD is active.
* Parameter is not illustrated in the AC Timing Diagrams.

AC CHARACTERISTICS (Z8410 / NMOS Z80 DMA)
(Active State)

Number	Symbol	Parameter	Z0841004 °‡† Min(ns)	Max(ns)
6	TdA	Address Output Delay		110
7	TdC(Az)	Clock ↑ to Address Float Delay		90
8	TsA(MREQ)	Address to $\overline{\text{MREQ}}$ ↓ Setup (Memory Cycle)	(2) + (5) − 75	
9	TsA(IRW)	Address Stable to $\overline{\text{IORQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$ ↓ Setup (I/O Cycle)	(1) − 70	
*10	TdRW(A)	$\overline{\text{RD}}$, $\overline{\text{WR}}$ ↑ to Addr. Stable Delay	(3) + (4) − 50	
*11	TdRW(Az)	$\overline{\text{RD}}$, $\overline{\text{WR}}$ ↑ to Addr. Float	(3) + (4) − 45	
12	TdCf(DO)	Clock ↓ to Data Out Delay		150
*13	TdCr(Dz)	Clock ↑ to Data Float Delay (Write Cycle)		90
14	TsDI(Cr)	Data In to Clock ↑ Setup (Read cycle when rising edge ends read)	35	
15	TsDI(Cf)	Data In to Clock ↓ Setup (Read cycle when falling edge ends read)	50	
*16	TsDO(WfM)	Data Out to $\overline{\text{WR}}$ ↓ Setup (Memory Cycle)	(1) − 170	
17	TsDO(WfI)	Data Out to $\overline{\text{WR}}$ ↓ Setup (I/O cycle)	100	
*18	TdWr(DO)	$\overline{\text{WR}}$ ↑ to Data Out Delay	(3) + (4) − 70	
19	Th	Hold Time for Any Specified Setup Time	0	
20	TdCr(Mf)	Clock ↑ to $\overline{\text{MREQ}}$ ↓ Delay		85
21	TdCf(Mf)	Clock ↓ to $\overline{\text{MREQ}}$ ↓ Delay		85
22	TdCr(Mr)	Clock ↑ to $\overline{\text{MREQ}}$ ↑ Delay		85
23	TdCf(Mr)	Clock ↓ to $\overline{\text{MREQ}}$ ↑ Delay		85
24	TwM1	$\overline{\text{MREQ}}$ Low Pulse Width	(1) − 30	
*25	TwMh	$\overline{\text{MREQ}}$ High Pulse Width	(2) + (5) − 20	
26	TdCf(If)	Clock ↓ to $\overline{\text{IORQ}}$ ↓ Delay		85
27	TdCr(If)	Clock ↑ to $\overline{\text{IORQ}}$ ↓ Delay		75
28	TdCr(Ir)	Clock ↑ to $\overline{\text{IORQ}}$ ↑ Delay		85
*29	TdCf(Ir)	Clock ↓ to $\overline{\text{IORQ}}$ ↑ Delay		85
30	TdCr(Rf)	Clock ↑ to $\overline{\text{RD}}$ ↓ Delay		85
31	TdCf(Rf)	Clock ↓ to $\overline{\text{RD}}$ ↓ Delay		95
32	TdCr(Rr)	Clock ↑ to $\overline{\text{RD}}$ ↑ Delay		85
33	TdCf(Rr)	Clock ↓ to $\overline{\text{RD}}$ ↑ Delay		85
34	TdCr(Wf)	Clock ↑ to $\overline{\text{WR}}$ ↓ Delay		65
35	TdCf(Wf)	Clock ↓ to $\overline{\text{WR}}$ ↓ Delay		80
36	TdCr(Wr)	Clock ↑ to $\overline{\text{WR}}$ ↑ Delay		80
37	TdCf(Wr)	Clock ↓ to $\overline{\text{WR}}$ ↑ Delay		80
38	TwWl	$\overline{\text{WR}}$ Low Pulse Width	(1) − 30	
39	TsWA(Cf)	$\overline{\text{WAIT}}$ to Clock ↓ Setup	70	
40	TdCr(B)	Clock ↑ to $\overline{\text{BUSREQ}}$ Delay		100
41	TdCr(Iz)	Clock ↑ to $\overline{\text{IORQ}}$, $\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$ Float Delay		80

NOTES:

‡ **All AC equations imply DMA default (standard) timing.**
† Data must be enabled onto data bus when $\overline{\text{RD}}$ is active.
* Parameter is not illustrated in the AC Timing Diagrams.

° **Numbers in parentheses are other parameter - numbers in this table; their values should be substituted in equations.**

AC CHARACTERISTICS (Z84C10 / CMOS Z80 DMA)

(Inactive State)

Number	Symbol	Parameter	Z84C1006 Min	Z84C1006 Max	Z84C1008 Min	Z84C1008 Max	Unit
1	TcC	Clock Cycle Time	162	DC	125	DC	
2	TwCh	Clock Width (High)	65	DC	55	DC	
3	TwCl	Clock Width (Low)	65	DC	55	DC	
4	TrC	Clock Rise Time		20		10	
5	TfC	Clock Fall Time		20		10	
6	Th	Hold Time for Any Specified Setup Time	0		0		ns
7	TsC(Cr)	$\overline{IORQ}$, $\overline{WR}$, $\overline{CE}$ ↓ to Clock ↑ Setup	60		45		ns
8	TdDO(RDf)	$\overline{RD}$ ↓ to Data Output Delay		300		220	ns
9	TsDI(Cr)	Data In to Clock ↑ Setup ($\overline{WR}$ or $\overline{M1}$)	30		20		ns
10	TdDO(IOf)	$\overline{IORQ}$ ↓ to Data Out Delay (INTA Cycle)		110		85	ns
11	TdRDr(Dz)	$\overline{RD}$ ↑ to Data Float Delay (output buffer disable)		70		50	ns
12	TsIEI(IORQf)	IEI to $\overline{IORQ}$ ↓ Setup (INTA Cycle)	100		80		ns
13	TdIEOr(IEIr)	IEI ↑ to IEO ↑ Delay		100		70	ns
14	TdIEOf(IEIf)	IEI ↓ to IEO ↓ Delay		100		70	ns
15	TdM1f(IEOf)	$\overline{M1}$ ↓ to IEO ↓ Delay (interrupt just prior to $\overline{M1}$ ↓)		100		80	ns
16	TsM1f(Cr)	$\overline{M1}$ ↓ to Clock ↑ Setup	70		45		ns
17	TsM1r(Cf)	$\overline{M1}$ ↑ to Clock Setup	-15		-15		ns
18	TsRDf(Cr)	$\overline{RD}$ ↓ to Clock ↑ Setup ($\overline{M1}$ Cycle)	60		45		ns
19	TdI(INTf)	Interrupt Cause to $\overline{INT}$ ↓ Delay ($\overline{INT}$ generated only when DMA is inactive)		450		400	ns
20	TdBAIr(BAOr)	$\overline{BAI}$ ↑ to $\overline{BAO}$ ↑ Delay		100		70	ns
21	TdBAIf(BAOf)	$\overline{BAI}$ ↓ to $\overline{BAO}$ ↓ Delay		100		70	ns
22	TsRDY(Cr)	RDY Active to Clock ↑ Setup	50		50		ns

NOTE: Negative minimum setup values mean that the first-mentioned event can come after the second-mentioned event.

* **Z84C10 Timing parameters are preliminary and subject to change.**

* **M1 must be active for a minimum of two clock cycles to reset the DMA (This feature is only with C-MOS Z80 DMA).**

Zilog **Product Specification**

January 1989

Z8420/Z84C20 NMOS/CMOS Z80 ® PIO Parallel Input/Output

FEATURES

- Provides a direct interface between Z80 microcomputer systems and peripheral devices.
- Two ports with interrupt-driven handshake for fast response.
- Four programmable operating modes: Output, Input, Bidirectional (Port A only), and Bit Control
- Programmable interrupts on peripheral status conditions.
- **NMOS version for high cost performance solutions.**
- **CMOS version for the designs requiring low power consumption.**
- **NMOS Z0842004 - 4 MHz, Z0842006 - 6.17 MHz.**
- **CMOS Z84C2004 - DC to 4 MHz, Z84C2006 - DC to 6.17 MHz, Z84C2008 - DC to 8 MHz.**
- Standard Z80 Family bus-request and prioritized interrupt-request daisy chains implemented without external logic.
- The eight Port B outputs can drive Darlington transistors (1.5 mA at 1.5V).
- **6 MHz version supports 6.144 MHz CPU clock operation.**

GENERAL DESCRIPTION

The Z80 PIO Parallel I/O Circuit (hereinafter referred to as the Z80 PIO or PIO) is a programmable, dual-port device that provides a TTL-compatible interface between peripheral devices and the Z80 CPU (Figures 1 and 2). Note the QFP package is only available in CMOS version. The CPU configures the Z80 PIO to interface with a wide range of peripheral devices that are compatible with the Z80 PIO include most keyboards, paper tape readers and punches, printers, and PROM programmers.

One characteristic of the Z80 peripheral controllers that separates them from other interface controllers is that all

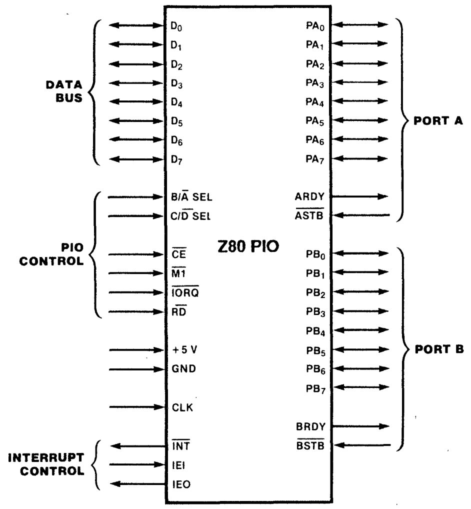
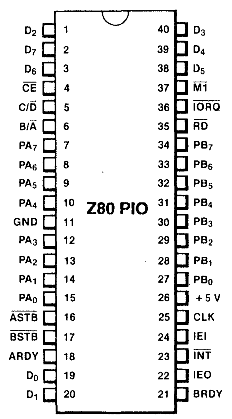


Figure 1. Pin Functions



Z80 PIO
D2 1, D7 2, D6 3, CE 4, C/D 5, B/A 6, PA7 7, PA6 8, PA5 9, PA4 10, GND 11, PA3 12, PA2 13, PA1 14, PA0 15, ASTB 16, BSTB 17, ARDY 18, D0 19, D1 20
40 D3, 39 D4, 38 D5, 37 M1, 36 IORQ, 35 RD, 34 PB7, 33 PB6, 32 PB5, 31 PB4, 30 PB3, 29 PB2, 28 PB1, 27 PB0, 26 +5 V, 25 CLK, 24 IEI, 23 INT, 22 IEO, 21 BRDY

Figure 2a. 40-pin Dual-In-Line Package (DIP), Pin Assignments

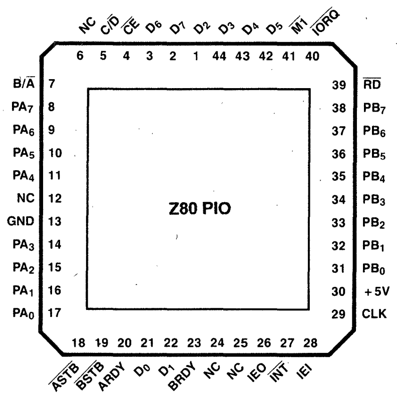


Figure 2b. 44-pin Chip Carrier, Pin Assignments

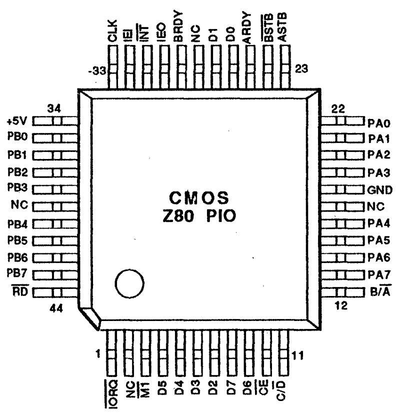


Figure 2c. 44-pin Quad Flat Pack Pin Assignments.

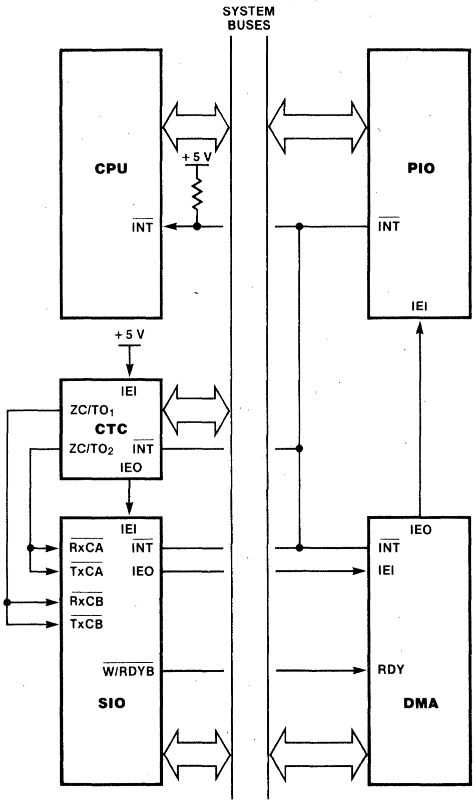


Figure 3. PIO in a Typical Z80 Family Environment

data transfer between the peripheral device and the CPU is accomplished under interrupt control. Thus, the interrupt logic of the PIO permits full use of the efficient interrupt capabilities of the Z80 CPU during I/O transfers. All logic necessary to implement a fully nested interrupt structure is included in the PIO (Figure 3).

Another feature of the PIO is the ability to interrupt the CPU upon occurrence of specified status conditions in the peripheral device. For example, the PIO can be programmed to interrupt if any specified peripheral alarm conditions should occur. This interrupt capability reduces the time the processor must spend in polling peripheral status.

The Z80 PIO interfaces to peripherals via two independent general-purpose I/O ports, designated Port A and Port B. Each port has eight data bits and two handshake signals, Ready and Strobe, which control data transfer. The Ready output indicates to the peripheral that the port is ready for a data transfer. Strobe is an input from the peripheral that indicates when a data transfer has occurred.

Operating Modes. The Z80 PIO ports can be programmed to operate in four modes: Output (Mode 0), Input (Mode 1), Bidirectional (Mode 2) and Bit Control (Mode 3).

Either Port A or Port B can be programmed to output data in Mode 0. Both ports have output registers that are individually addressed by the CPU; data can be written to either port at any time. When data is written to a port, an active Ready output indicates to the external device that data is available at the associated port and is ready for transfer to the external device. After the data transfer, the external device responds with an active Strobe input, which generates an interrupt, if enabled.

Either Port A or Port B can be programmed to input data in Mode 1. Each port has an input register addressed by the

CPU. When the CPU reads data from a port, the PIO sets the Ready signal, which is detected by the external device. The external device then places data on the I/O lines and strobes the I/O port, which latches the data into the Port Input Register, resets Ready, and triggers the Interrupt Request, if enabled. The CPU can read the input data at any time, which again sets Ready.

Mode 2 is bidirectional and uses only Port A, plus the interrupts and handshake signals from both ports. Port B must be set to Mode 3 and masked off from generating interrupts. In operation, Port A is used for both data input and output. Output operation is similar to Mode 0 except that data is allowed out onto the Port A bus only when $\overline{\text{ASTB}}$ is Low. For input, operation is similar to Mode 1, except that the data input uses the Port B handshake signals and the Port B interrupt, if enabled.

Both ports can be used in Mode 3. In this mode, the individual bits are defined as either input or output bits. This provides up to eight separate, individually defined bits for each port. During operation, Ready and Strobe are not used. Instead, an interrupt is generated if the condition of one input changes, or if all inputs change. The requirements for generating an interrupt are defined during the programming operation; the active level is specified as either High or Low, and the logic condition is specified as either one input active (OR) or all inputs active (AND). For example, if the port is programmed for active Low inputs and the logic function is AND, then all inputs at the specified port must go Low to generate an interrupt.

Data outputs are controlled by the CPU and can be written or changed at any time.

- Individual bits can be masked off.
- The handshake signals are not used in Mode 3; Ready is held Low, and Strobe is disabled.
- When using the Z80 PIO interrupts, the Z80 CPU interrupt mode must be set to Mode 2.

INTERNAL STRUCTURE

The internal structure of the Z80 PIO consists of a Z80 CPU bus interface, internal control logic, Port A I/O logic, Port B I/O logic, and interrupt control logic (Figure 4). The CPU bus interface logic allows the Z80 PIO to interface directly to the Z80 CPU with no other external logic. The internal control logic synchronizes the CPU data bus to the peripheral device interfaces (Port A and Port B). The two I/O ports (A and B) are virtually identical and are used to interface directly to peripheral devices.

Port Logic. Each port contains separate input and output registers, handshake control logic, and the control registers shown in Figure 5. All data transfers between the peripheral unit and the CPU use the data input and output registers. The handshake logic associated with each port controls the data transfers through the input and the output registers. The mode control register (two bits) selects one of the four programmable operating modes.

The Bit Control mode (Mode 3) uses the remaining registers. The input/output control register specifies which of the eight data bits in the port are to be outputs and enables these bits; the remaining bits are inputs. The mask register and the mask control register govern Mode 3 interrupt conditions. The mask register specifies which of the bits in the port are active and which are masked or inactive.

The mask control register specifies two conditions: first, whether the active state of the input bits is High or Low, and second, whether an interrupt is generated when any one unmasked input bit is active (OR condition) or if the interrupt is generated when *all* unmasked input bits are active (AND condition).

Interrupt Control Logic. The interrupt control logic section handles all CPU interrupt protocol for nested-priority interrupt structures. Any device's physical location in a

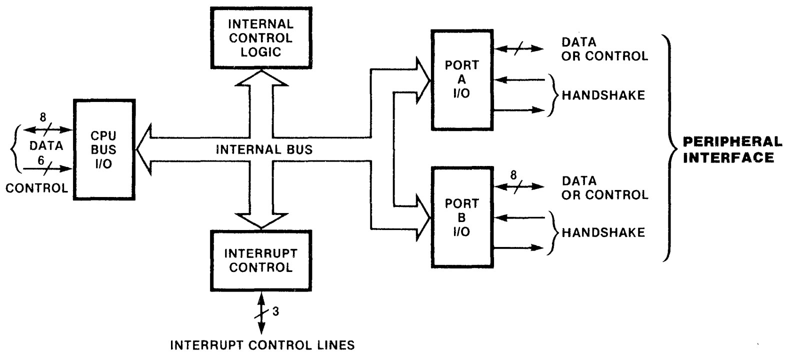


Figure 4. Block Diagram

daisy-chain configuration determines its priority. Two lines (IEI and IEO) are provided in each PIO to form this daisy chain. The device closest to the CPU has the highest priority. Within a PIO, Port A interrupts have higher priority than those of Port B. In the byte input, byte output, or bidirectional modes, an interrupt can be generated whenever the peripheral requests a new byte transfer. In the bit control mode, an interrupt can be generated when the peripheral status matches a programmed value. The PIO provides for complete control of nested interrupts. That is, lower priority devices may not interrupt higher priority devices that have not had their interrupt service routines completed by the CPU. Higher priority devices may interrupt the servicing of lower priority devices.

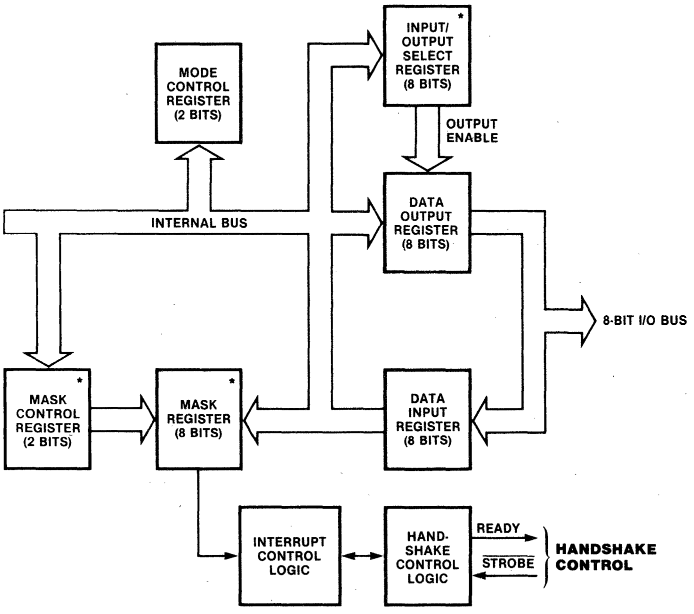
If the CPU (in interrupt Mode 2) accepts an interrupt, the interrupting device must provide an 8-bit interrupt vector for the CPU. This vector forms a pointer to a location in memory where the address of the interrupt service routine is located. The 8-bit vector from the interrupting device forms the least significant eight bits of the indirect pointer while the I Register in the CPU provides the most significant eight bits of the pointer. Each port (A and B) has an independent interrupt vector. The least significant bit of the vector is automatically set to 0 within the PIO because the pointer must point to two adjacent memory locations for a complete 16-bit address.

Unlike the other Z80 peripherals, the PIO does not enable interrupts immediately after programming. It waits until $\overline{\text{M1}}$ goes Low (e.g., during an opcode fetch). This condition is unimportant in the Z80 environment but might not be if another type of CPU is used.

The PIO decodes the RETI (Return From Interrupt) instruction directly from the CPU data bus so that each PIO in the system knows at all times whether it is being serviced by the CPU interrupt service routine. No other communication with the CPU is required.

CPU Bus I/O Logic. The CPU bus interface logic interfaces the Z80 PIO directly to the Z80 CPU, so no external logic is necessary. For large systems, however, address decoders and/or buffers may be necessary.

Internal Control Logic. This logic receives the control words for each port during programming and, in turn, controls the operating functions of the Z80 PIO. The control logic synchronizes the port operations, controls the port mode, port addressing, selects the read/write function, and issues appropriate commands to the ports and the interrupt logic. The Z80 PIO does not receive a write input from the CPU; instead, the $\overline{\text{RD}}$, $\overline{\text{CE}}$, C/$\overline{\text{D}}$ and $\overline{\text{IORQ}}$ signals internally generate the write input.


INPUT/ OUTPUT SELECT REGISTER (8 BITS) *
MODE CONTROL REGISTER (2 BITS)
OUTPUT ENABLE
INTERNAL BUS
DATA OUTPUT REGISTER (8 BITS)
8-BIT I/O BUS
MASK CONTROL REGISTER (2 BITS) *
MASK REGISTER (8 BITS) *
DATA INPUT REGISTER (8 BITS)
INTERRUPT CONTROL LOGIC
HAND-SHAKE CONTROL LOGIC
READY
STROBE
HANDSHAKE CONTROL

*Used in the bit mode only to allow generation of an interrupt if the peripheral I/O pins go to the specified state.

Figure 5. Typical Port I/O Block Diagram

PROGRAMMING

Mode 0, 1, or 2. (Input, Output, or Bidirectional). Programming a port for Mode 0, 1, or 2 requires at least one, and up to three, control words per port. These words are:

Mode Control Word (Figure 6). Selects the port operating mode. This word is required and may be written at any time.

Interrupt Vector Word (Figure 7). The Z80 PIO is designed for use with the Z80 CPU in interrupt Mode 2. This word must be programmed if interrupts are to be used.

Interrupt Control Word (Figure 9) or ***Interrupt Disable Word*** (Figure 11). Controls the enable or disable of the PIO interrupt function.

Mode 3 (Bit Control). Programming a port for Mode 3 requires at least two, and up to four, control words.

Mode Control Word (Figure 6). Selects the port operating mode. This word is required and may be written at any time.

I/O Register Control Word (Figure 8). When Mode 3 is selected, the Mode Control Word must be followed by the I/O Control Word. This word configures the I/O control register, which defines which port lines are inputs or outputs. This word is required.

Interrupt Vector Word (Figure 7). The Z80 PIO is designed for use with the Z80 CPU in interrupt Mode 2. This word must be programmed if interrupts are to be used.

Interrupt Control Word. In Mode 3, handshake is not used. Interrupts are generated as a logic function of the input signal levels. The interrupt control word sets the logic conditions and the logic levels required for generating an interrupt. Two logic conditions or functions are available: AND (if all input bits change to the active level, an interrupt is triggered), and OR (if any one of the input bits changes to the active level, an interrupt is triggered). Bit D_5 sets the logic function, as shown in Figure 9. The active level of the input bits can be set either High or Low. The active level is controlled by Bit D_5.

Mask Control Word. This word sets the mask control register, allowing any unused bits to be masked off. If any bits are to be masked, then D_4 must be set. When D_4 is set, the next word written to the port must be a mask control word (Figure 10).

Interrupt Disable Word. This control word can be used to enable or disable a port interrupt. It can be used without changing the rest of the interrupt control word (Figure 11).

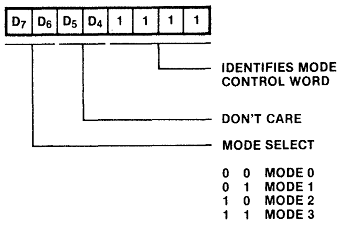
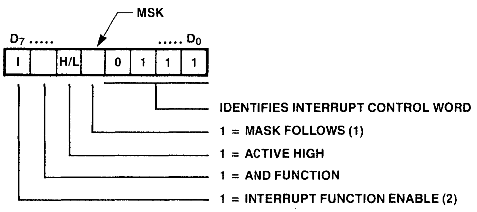


Figure 6. Mode Control Word



*NOTE:
1. Regardless of the operating mode, setting Bit D_4 = 1 causes any pending interrupts to be cleared.
2. The port interrupt is not enabled until the interrupt function enable is followed by an active $\overline{M1}$.

Figure 9. Interrupt Control Word

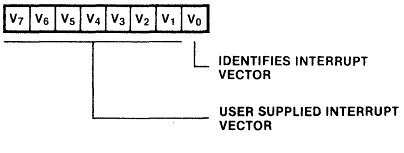


Figure 7. Interrupt Vector Word

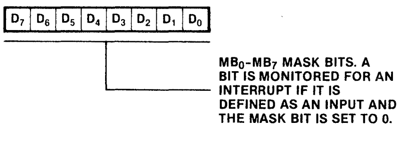

D7 D6 D5 D4 D3 D2 D1 D0
MB0-MB7 MASK BITS. A BIT IS MONITORED FOR AN INTERRUPT IF IT IS DEFINED AS AN INPUT AND THE MASK BIT IS SET TO 0.

Figure 10. Mask Control Word

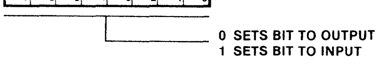


Figure 8. I/O Register Control Word

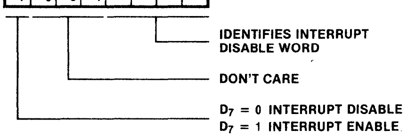


Figure 11. Interrupt Disable Word

PIN DESCRIPTION

PA_0-PA_7. *Port A Bus* (bidirectional, 3-state). This 8-bit bus transfers data, status, or control information between Port A of the PIO and a peripheral device. PA_0 is the least significant bit of the Port A data bus.

ARDY. *Register A Ready* (output, active High). The meaning of this signal depends on the mode of operation selected for Port A as follows:

Output Mode. This signal goes active to indicate that the Port A output register has been loaded and the peripheral data bus is stable and ready for transfer to the peripheral device.

Input Mode. This signal is active when the Port A input register is empty and ready to accept data from the peripheral device.

Bidirectional Mode. This signal is active when data is available in the Port A output register for transfer to the peripheral device. In this mode, data is not placed on the Port A data bus, unless $\overline{\text{ASTB}}$ is active.

Control Mode. This signal is disabled and forced to a Low state.

$\overline{\text{ASTB}}$. *Port A Strobe Pulse From Peripheral Device* (input, active Low). The meaning of this signal depends on the mode of operation selected for Port A as follows:

Output Mode. The positive edge of this strobe is issued by the peripheral to acknowledge the receipt of data made available by the PIO.

Input Mode. The strobe is issued by the peripheral to load data from the peripheral into the Port A input register. Data is loaded into the PIO when this signal is active.

Bidirectional Mode. When this signal is active, data from the Port A output register is gated onto the Port A bidirectional data bus. The positive edge of the strobe acknowledges the receipt of the data.

Control Mode. The strobe is inhibited internally.

PB_0-PB_7. *Port B Bus* (bidirectional, 3-state). This 8-bit bus transfers data, status, or control information between Port B and a peripheral device. The Port B data bus can supply 1.5 mA at 1.5V to drive Darlington transistors. PB_0 is the least significant bit of the bus.

B/$\overline{\text{A}}$. *Port B or A Select* (input, High = B). This pin defines which port is accessed during a data transfer between the CPU and the PIO. A Low on this pin selects Port A; a High selects Port B. Often address bit A_0 from the CPU is used for this selection function.

BRDY. *Register B Ready* (output, active High). This signal is similar to ARDY, except that in the Port A bidirectional mode this signal is High when the Port A input register is empty and ready to accept data from the peripheral device.

$\overline{\text{BSTB}}$. *Port B Strobe Pulse From Peripheral Device* (input, active Low). This signal is similar to $\overline{\text{ASTB}}$, except that in the Port A bidirectional mode this signal strobes data from the peripheral device into the Port A input register.

C/$\overline{\text{D}}$. *Control or Data Select* (input, High = C). This pin defines the type of data transfer to be performed between the CPU and the PIO. A High on this pin during a CPU write to the PIO causes the Z80 data bus to be interpreted as a *command* for the port selected by the B/$\overline{\text{A}}$ Select line. A Low on this pin means that the Z80 data bus is being used to transfer data between the CPU and the PIO. Often address bit A_1 from the CPU is used for this function.

$\overline{\text{CE}}$. *Chip Enable* (input, active Low). A Low on this pin enables the PIO to accept command or data inputs from the CPU during a write cycle or to transmit data to the CPU during a read cycle. This signal is generally decoded from four I/O port numbers for Ports A and B, data, and control.

CLK. *System Clock* (input). The Z80 PIO uses the standard single-phase Z80 system clock.

D_0-D_7. *Z80 CPU Data Bus* (bidirectional, 3-state). This bus is used to transfer all data and commands between the Z80 CPU and the Z80 PIO. D_0 is the least significant bit.

IEI. *Interrupt Enable In* (input, active High). This signal is used to form a priority-interrupt daisy chain when more than one interrupt driven device is being used. A High level on this pin indicates that no other devices of higher priority are being serviced by a CPU interrupt service routine.

IEO. *Interrupt Enable Out* (output, active High). The IEO signal is the other signal required to form a daisy chain priority scheme. It is High only if IEI is High and the CPU is not servicing an interrupt from this PIO. Thus this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU interrupt service routine.

$\overline{\text{INT}}$. *Interrupt Request* (output, open drain, active Low). When $\overline{\text{INT}}$ is active the Z80 PIO is requesting an interrupt from the Z80 CPU.

$\overline{\text{IORQ}}$. *Input/Output Request* (input from Z80 CPU, active Low). $\overline{\text{IORQ}}$ is used in conjunction with B/$\overline{\text{A}}$, C/$\overline{\text{D}}$, $\overline{\text{CE}}$, and $\overline{\text{RD}}$ to transfer commands and data between the Z80 CPU and the Z80 PIO. When $\overline{\text{CE}}$, $\overline{\text{RD}}$, and $\overline{\text{IORQ}}$ are active, the port addressed by B/$\overline{\text{A}}$ transfers data to the CPU (a read operation). Conversely, when $\overline{\text{CE}}$ and $\overline{\text{IORQ}}$ are active but $\overline{\text{RD}}$ is not, the port addressed by B/$\overline{\text{A}}$ is written into from the CPU with either data or control information, as specified by C/$\overline{\text{D}}$. Also, if $\overline{\text{IORQ}}$ and $\overline{\text{M1}}$ are active simultaneously, the CPU is acknowledging an interrupt; the interrupting port automatically places its interrupt vector on the CPU data bus if it is the highest priority device requesting an interrupt.

$\overline{\text{M1}}$. *Machine Cycle* (input from CPU, active Low). This signal is used as a sync pulse to control several internal PIO operations. When both the $\overline{\text{M1}}$ and $\overline{\text{RD}}$ signals are active, the Z80 CPU is fetching an instruction from memory. Conversely, when both $\overline{\text{M1}}$ and $\overline{\text{IORQ}}$ are active, the CPU is acknowledging an interrupt. In addition, $\overline{\text{M1}}$ has two other functions within the Z80 PIO: it synchronizes the PIO interrupt logic; when $\overline{\text{M1}}$ occurs without an active $\overline{\text{RD}}$ or $\overline{\text{IORQ}}$ signal, the PIO is reset.

$\overline{\text{RD}}$. *Read Cycle Status* (input from Z80 CPU, active Low). If $\overline{\text{RD}}$ is active, or an I/O operation is in progress, $\overline{\text{RD}}$ is used with B/$\overline{\text{A}}$, C/$\overline{\text{D}}$, $\overline{\text{CE}}$, and $\overline{\text{IORQ}}$ to transfer data from the Z80 PIO to the Z80 CPU.

TIMING

The following timing diagrams show typical timing in a Z80 CPU environment. For more precise specifications refer to the composite ac timing diagram.

Write Cycle. Figure 12 illustrates the timing for programming the Z80 PIO or for writing data to one of its ports. The PIO does not receive a specific write signal; it internally generates its own from the lack of an active $\overline{\text{RD}}$ signal.

Read Cycle. Figure 13 illustrates the timing for reading the data input from an external device to one of the Z80 PIO ports.

Output Mode (Mode 0). An output cycle (Figure 14) is always started by the execution of an output instruction by the CPU. The $\overline{\text{WR}}$* pulse from the CPU latches the data from the CPU data bus into the selected port's output register. The $\overline{\text{WR}}$* pulse sets the Ready flag after a Low-going edge of CLK, indicating data is available. Ready stays active until the positive edge of the strobe line is received, indicating that data was taken by the peripheral. The positive edge of the strobe pulse generates an $\overline{\text{INT}}$ if the interrupt enable flip-flop has been set and if this device has the highest priority.

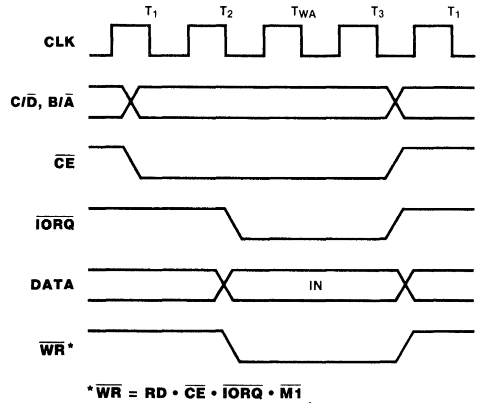


Figure 12. Write Cycle Timing

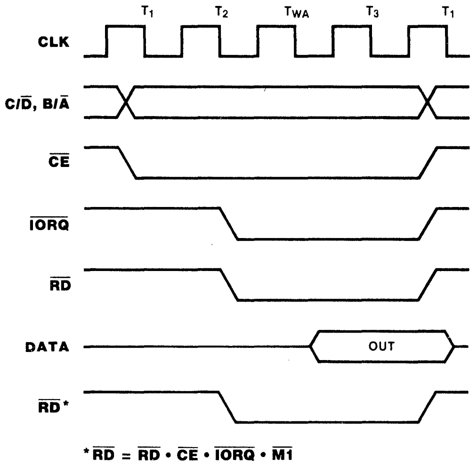


Figure 13. Read Cycle Timing

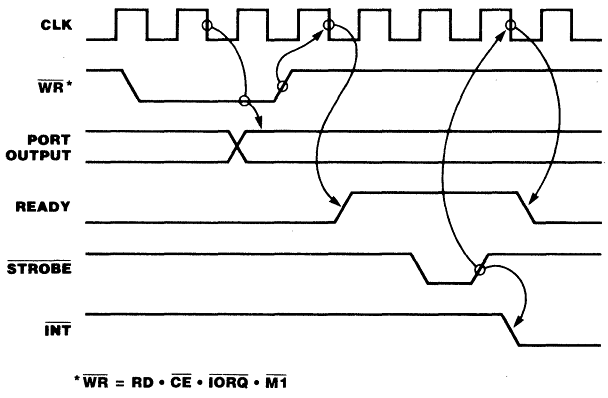


Figure 14. Mode 0 Output Timing

Input Mode (Mode 1). When $\overline{\text{STROBE}}$ goes from Low to High, data is latched into the selected port input register (Figure 15). While $\overline{\text{STROBE}}$ is Low, the input data latches are transparent. The next rising edge of $\overline{\text{STROBE}}$ activates $\overline{\text{INT}}$, if Interrupt Enable is set and this is the highest-priority requesting device. The following falling edge of CLK resets Ready to an inactive state, indicating that the input register is full and cannot accept any more data until the CPU completes a read. When a read is complete, the positive edge of $\overline{\text{RD}}$ sets Ready at the next Low-going transition of CLK. At this time new data can be loaded into the PIO.

Bidirectional Mode (Mode 2). This is a combination of Modes 0 and 1 using all four handshake lines and the eight Port A I/O lines (Figure 16). Port B must be set to the bit mode and its inputs must be masked. The Port A handshake lines are used for output control and the Port B lines are used for input control. If interrupts occur, Port A's vector will be used during port output and Port B's will be used during port input. Data is allowed out onto the Port A bus only when $\overline{\text{ASTB}}$ is Low. The rising edge of this strobe can be used to latch the data into the peripheral.

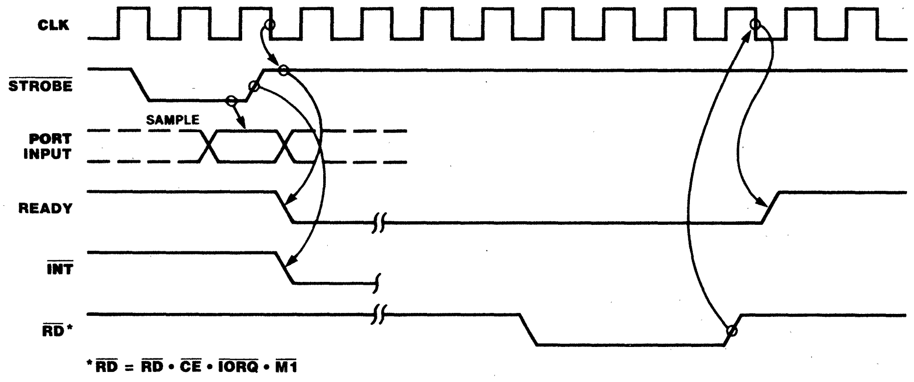


Figure 15. Mode 1 Input Timing

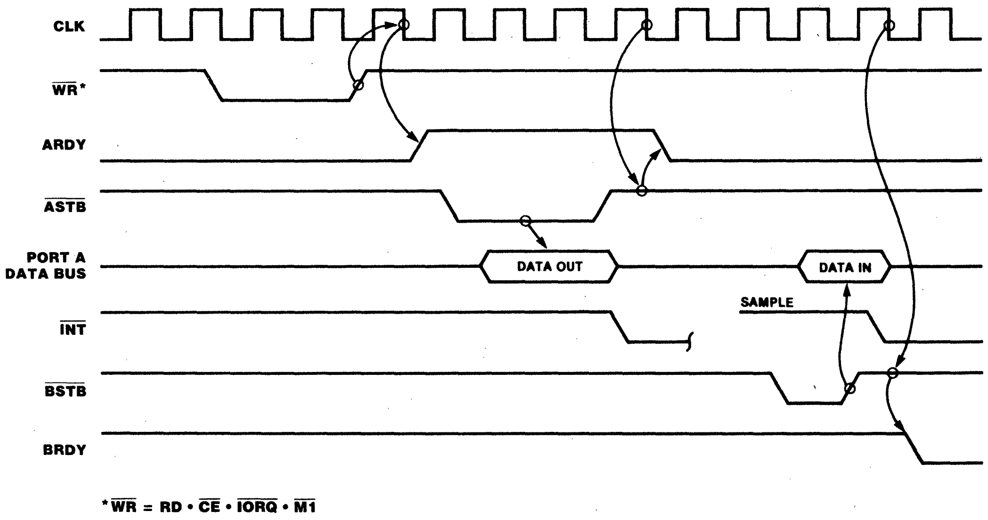


Figure 16. Mode 2 Bidirectional Timing

Bit Control Mode (Mode 3). The bit mode does not utilize the handshake signals, and a normal port write or port read can be executed at any time. When writing, the data is latched into the output registers with the same timing as the output mode.

When reading (Figure 17) the PIO, the data returned to the CPU is composed of output register data from those port data lines assigned as outputs and input register data from those port data lines assigned as inputs. The input register contains data that was present immediately prior to the falling edge of $\overline{RD}$. An interrupt is generated if interrupts from the port are enabled and the data on the port data lines satisfy the logical equation defined by the 8-bit mask and 2-bit mask control registers. However, if Port A is programmed in bidirectional mode, Port B does not issue an interrupt in bit mode and must therefore be polled.

Interrupt Acknowledge Timing. During $\overline{M1}$ time, peripheral controllers are inhibited from changing their interrupt enable status, permitting the Interrupt Enable signal to ripple through the daisy chain. The peripheral with IEI High and IEO Low during INTACK places a preprogrammed 8-bit interrupt vector on the data bus at this time (Figure 18). IEO is held Low until a Return From Interrupt (RETI) instruction is executed by the CPU while IEI is High. The 2-byte RETI instruction is decoded internally by the PIO for this purpose.

Return From Interrupt Cycle. If a Z80 peripheral has no interrupt pending and is not under service, then its IEO = IEI. If it has an interrupt under service (i.e., it has already interrupted and received an interrupt acknowledge) then its IEO is always Low, inhibiting lower priority devices from interrupting. If it has an interrupt pending which has not yet been acknowledged, IEO is Low unless an "ED" is decoded as the first byte of a 2-byte opcode (Figure 19). In this case, IEO goes High until the next opcode byte is decoded, whereupon it goes Low again. If the second byte of the opcode was a "4D," then the opcode was an RETI instruction.

After an "ED" opcode is decoded, only the peripheral device which has interrupted and is currently under service has its IEI High and its IEO Low. This device is the highest-priority device in the daisy chain that has received an interrupt acknowledge. All other peripherals have IEI = IEO. If the next opcode byte decoded is "4D," this peripheral device resets its "interrupt under service" condition.

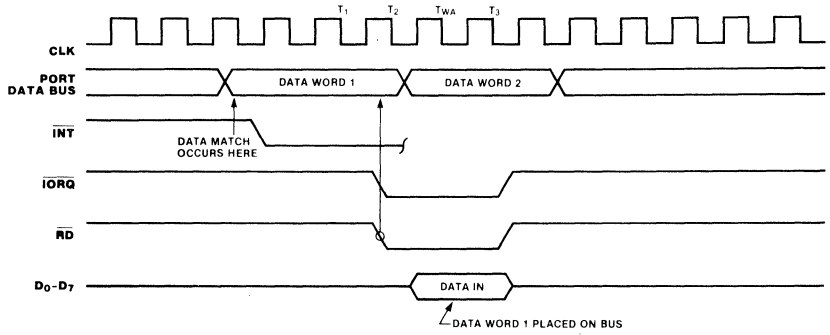


Figure 17. Mode 3 Bit Control Mode Timing, Bit Mode Read

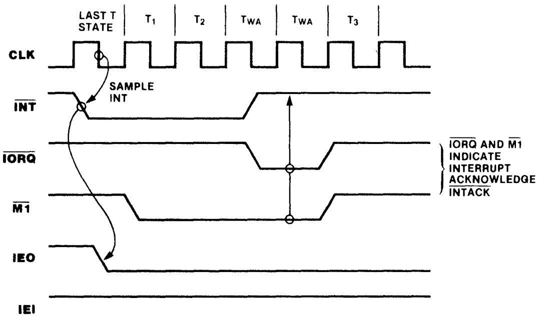


Figure 18. Interrupt Acknowledge Timing

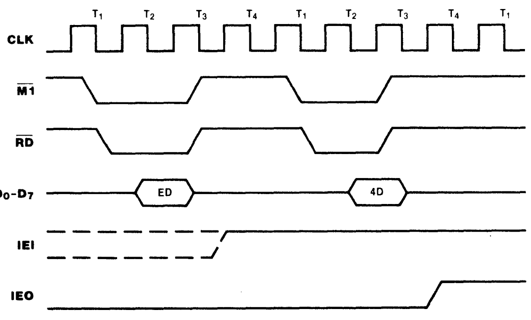


Figure 19. Return From Interrupt

ABSOLUTE MAXIMUM RATINGS

Voltages on V_{CC} with respect to V_{SS} . . . . . −0.3V to +7.0V
Voltages on all inputs with respect
to V_{SS} . . . . −0.3V to V_{CC}+0.3V
Storage Temperature . . . . −65°C to +150°C

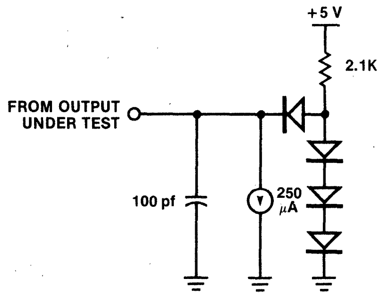
Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above these indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The characteristics below apply for the following test conditions, unless otherwise noted. All voltages are referenced to GND (0V). Positive current flows into the referenced pin. Available operating temperature range is:

- **S = 0°C to +70°C, V_{cc} Range**
  NMOS: +4.75V < V_{cc} < +5.25V
  CMOS: +4.50V < V_{cc} <+5.50V

- **E = -40°C to 100°C, +4.50V < V_{cc} < +5.50V**



The Ordering Information section lists package temperature ranges and product numbers. Refer to the Literature List for additional documentation. Package drawings are in the Package Information section.

CAPACITANCE

Symbol	Parameter	Min	Max	Unit
C	Clock Capacitance		10	pf
C_{IN}	Input Capacitance		5	pf
C_{OUT}	Output Capacitance		15	pf

Over specified temperature range; f = 1 MHz.
Unmeasured pins returned to ground.

DC CHARACTERISTICS (Z84C20/CMOS Z80 PIO)

Symbol	Parameter	Min	Max	Typ	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3	$+0.45$		V	
V_{IHC}	Clock Input High Voltage	$V_{CC}-0.6$	$V_{CC}+0.3$		V	
V_{IL}	Input Low Voltage	-0.3	$+0.8$		V	
V_{IH}	Input High Voltage	$+2.2$	V_{CC}		V	
V_{OL}	Output Low Voltage		$+0.4$		V	$I_{OL} = 2.0$ mA
V_{OH_1}	Output High Voltage	$+2.4$			V	$I_{OH} = -1.6$ mA
V_{OH_2}	Output High Voltage	$V_{CC}-0.8$			V	$I_{OH} = -250\,\mu$A
I_{LI}	Input Leakage Current		± 10		μA	$V_{IN} = 0.4$ to V_{CC}
I_{LO}	3-State Output Leakage Current in Float		± 10		μA	$V_{OUT} = 0.4$ to V_{CC}
ICC_1	Power Supply Current **4 MHz**		5	2	mA	$V_{CC} = 5V$
	6 MHz		**6**			**CLK = 4 MHz, 6 MHz, 8 MHz**
	8 MHz		**7**			$V_{IH} = V_{CC} - 0.2V$
						$V_{IL} = 0.2V$
ICC_2	Standby Supply Current		10	0.5	μA	$V_{CC} = 5V$
						CLK = (0)
						$V_{IH} = V_{CC} - 0.2V$
						$V_{IL} = 0.2V$
I_{OHD}	Darlington Drive Current, Port B only	-1.5	-5.0		mA	$V_{OH} = 1.5V$
						$R_{EXT} = 1.1K\,\Omega$

Over specified temperature and voltage range.

AC CHARACTERISTICS (Z84C20/CMOS Z80 PIO)

			Z84C2004		Z84C2006		Z84C2008		
No.	Symbol	Parameter	Min	Max	Min	Max	Min	Max	Notes
1	TcC	Clock Cycle Time	250	[1]	162	[1]	125	DC	
2	TwCh	Clock Pulse Width (High)	105	DC	65	DC	55	DC	
3	TwCl	Clock Pulse Width (Low)	105	DC	65	DC	55	DC	
4	TfC	Clock Fall Time		30		20		10	
5	TrC	Clock Rise Time		30		20		10	
6	TsCS(RI)	$\overline{CE}$,B/$\overline{A}$,C/$\overline{D}$ to $\overline{RD}$,$\overline{IORQ}$ ↓ Setup Time	50		50		40		[6]
7	Th	Any Hold Times for Specified Setup Time	40		35		15		
8	TsRI(C)	$\overline{RD}$,$\overline{IORQ}$ to Clock ↑ Setup Time	115		70		60		
9	TdRI(DO)	$\overline{RD}$,$\overline{IORQ}$ ↓ to Data Out Delay		380		300		200	[2]
10	TdRI(DOs)	$\overline{RD}$,$\overline{IORQ}$ ↑ to Data Out Float Delay		110		70		60	
11	TsDI(C)	Data In to Clock ↑ Setup Time	50		40		30		CL=50 pf
12	TdIO(DOI)	$\overline{IORQ}$ ↓ to Data Out Delay (INTACK Cycle)		160		120		80	[3]
13	TsM1(Cr)	$\overline{M1}$ ↓ to Clock ↑ Setup Time	90		70		50		
14	TsM1(Cf)	$\overline{M1}$ ↑ to Clock ↓ Setup Time (M1 Cycle)	0		0		0	[8]	
15	TdM1(IEO)	$\overline{M1}$ ↓ to IEO ↓ Delay (Interrupt Immediately Preceding $\overline{M1}$ ↓)		190		100		70	[5,7]
16	TsIEI(IO)	IEI to $\overline{IORQ}$ ↓ Setup Time (INTACK Cycle)	140		100		80		[7]
17	TdIEI(IEOf)	IEI ↓ to IEO ↓ Delay		130		120		70	[5] CL=50 pf
18	TdIEI(IEOr)	IEI ↑ to IEO ↑ Delay (after ED Decode)		160		150		70	[5]
19	TcIO(C)	$\overline{IORQ}$ ↑ to Clock ↓ Setup Time (To Activate READY on Next Clcok Cycle)	200		170		140		
20	TdC(RDYr)	Clock ↓ to READY ↑ Delay		190		170		150	[5] CL=50 pf
21	TdC(RDYf)	Clock ↓ to READY ↓ Delay		140		120		100	[5]
22	TwSTB	$\overline{STROBE}$ Pulse Width	150		120		100		[4]
23	TsSTB(C)	$\overline{STROBE}$ ↑ to Clock ↓ Setup Time (To Activate READY on Next Clock Cycle)	220		150		120		[5]
24	TdIO(PD)	$\overline{IORQ}$ ↑ to PORT DATA Stable Delay (Mode 0)		180		160		140	[5]
25	TsPD(STB)	PORT DATA to $\overline{STROBE}$ ↑ Setup Time (Mode 1)	230		190		140		
26	TdSTB(PD)	$\overline{STROBE}$ ↓ to PORT DATA Stable (Mode 2)		210		180		150	[5]
27	TdSTB(PDr)	$\overline{STROBE}$ ↑ to PORT DATA Float Delay (Mode 2)		180		160		140	CL=50 pf
28	TdPD(INT)	PORT DATA Match to $\overline{INT}$ ↓ Delay (Mode 3)		490		430		360	
29	TdSTB(INT)	$\overline{STROBE}$ ↑ to $\overline{INT}$ ↓ Delay		440		350		290	

NOTES:
[1] TcC = TwCh + TwCl + TrC + TfC.
[2] Increase TdRI(DO) by 10 ns for each 50 pf increase in load up to 200 pf max.
[3] Increase TdIO(DOI) by 10 ns for each 50 pf, increase in loading up to 200 pf max.
[4] For Mode 2: TwSTB > TsPD(STB).
[5] Increase these values by 2 ns for each 10 pf increase in loading up to 100 pf max.
[6] TsCS(RI) may be reduced. However, the time subtracted from TsCS(RI) will be added to TdRI(DO).
[7] 2.5 TcC > (N − 2)TdIEI(IEOf) + TdM1(IEO) + TsIEI(IO) + TTL Buffer Delay, if any.
[8] M1 must be active for a minimum of two clock cycles to reset the PIO.
[9] All parameters in nanoseconds unless otherwise specified.

DC CHARACTERISTICS (Z8420/NMOS Z80 PIO)

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	−0.3	+0.45	V	
V_{IHC}	Clock Input High Voltage	$V_{CC}-0.6$	$V_{CC}+0.3$	V	
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	+2.0	V_{CC}	V	
V_{OL}	Output Low Voltage		+0.4	V	$I_{OL} = 2.0$ mA
V_{OH}	Output High Voltage	+2.4		V	$I_{OH} = -250\,\mu A$
I_{LI}	Input Leakage Current		±10	μA	$V_{IN} = 0$ to V_{CC}
I_{LO}	3-State Output Leakage Current in Float		±10	μA	$V_{OUT} = 0.4V$ to V_{CC}
I_{CC}	Power Supply Current		100	mA	
I_{OHD}	Darlington Drive Current Port B Only	−1.5		mA	$V_{OH} = 1.5V$ $R_{EXT} = 390\,\Omega$

Over specified temperature and voltage range.

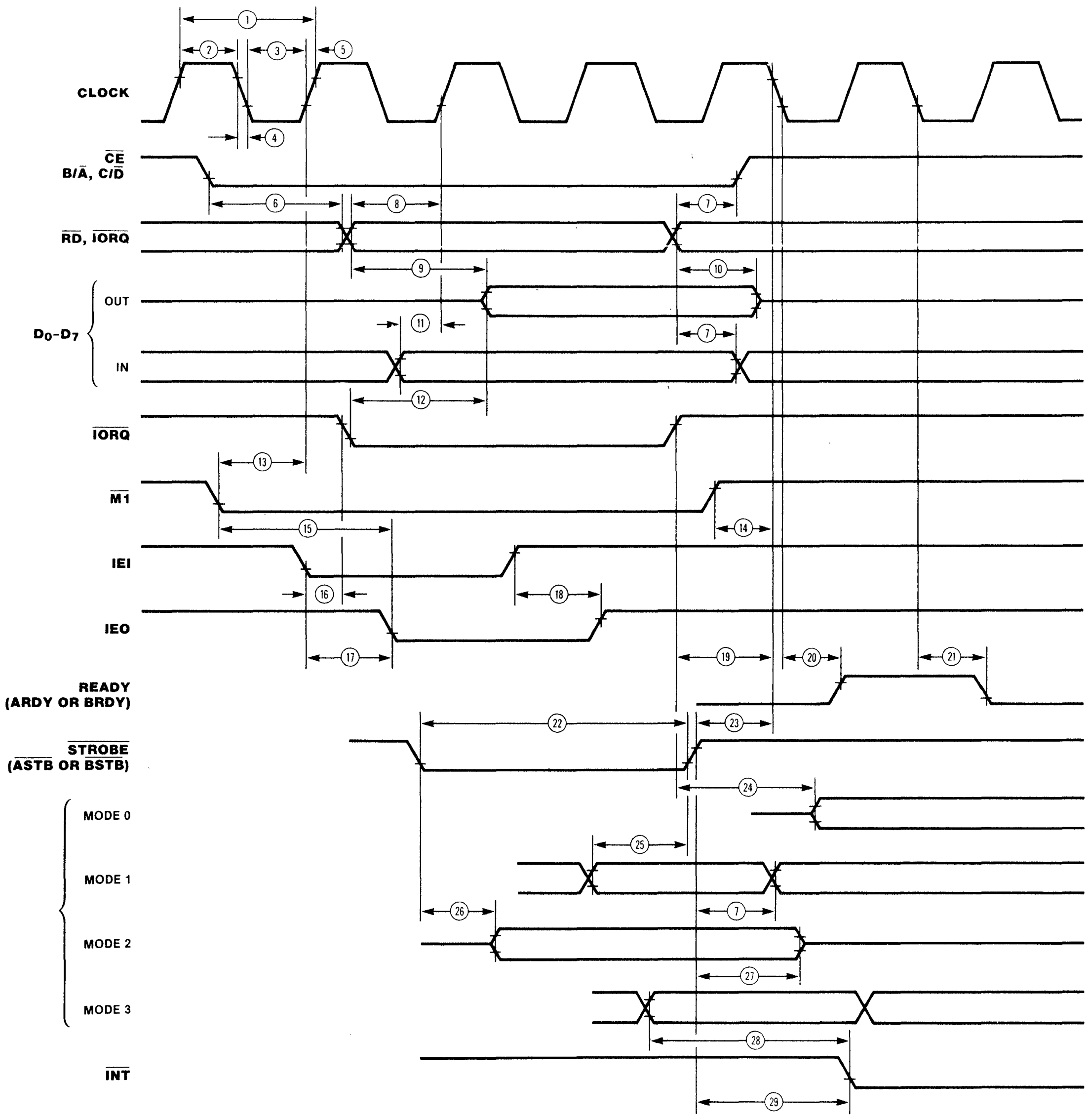
AC CHARACTERISTICS† (Z8420/NMOS Z80 PIO)

			Z0842004		Z0842006		
Number	Symbol	Parameter	Min	Max	Min	Max	Notes
1	TcC	Clock Cycle Time	250	[1]	**162**	[1]	
2	TwCh	Clock Width (High)	105	2000	65	2000	
3	TwC1	Clock Width (Low)	105	2000	65	2000	
4	TfC	Clock Fall Time		30		20	
5	TrC	Clock Rise Time		30		20	
6	TsCS(RI)	$\overline{\text{CE}}$, B/$\overline{\text{A}}$, C/$\overline{\text{D}}$ to $\overline{\text{RD}}$, $\overline{\text{IORQ}}$ ↓ Setup Time	50		50		[6]
7	Th	Any Hold Times for Specified Setup Time	0		0	0	
8	TsRI(C)	$\overline{\text{RD}}$, $\overline{\text{IORQ}}$ to Clock ↑ Setup Time	115		70		
9	TdRI(DO)	$\overline{\text{RD}}$, $\overline{\text{IORQ}}$ ↓ to Data Out Delay		380		300	[2]
10	TdRI(DOs)	$\overline{\text{RD}}$, $\overline{\text{IORQ}}$ ↑ to Data Out Float Delay		110		70	
11	TsDI(C)	Data In to Clock ↑ Setup Time	50		40		CL = 50 pf
12	TdIO(DOI)	$\overline{\text{IORQ}}$ ↓ to Data Out Delay (INTACK Cycle)		**200**		120	[3]
13	TsM1(Cr)	$\overline{\text{M1}}$ ↓ to Clock ↑ Setup Time	90		70		
14	TsM1(Cf)	$\overline{\text{M1}}$ ↑ to Clock ↓ Setup Time ($\overline{\text{M1}}$ Cycle)	0		0		[8]
15	TdM1(IEO)	$\overline{\text{M1}}$ ↓ to IEO ↓ Delay (Interrupt Immediately Preceding $\overline{\text{M1}}$ ↓)		190		100	[5,7]
16	TsIEI(IO)	IEI to $\overline{\text{IORQ}}$ ↓ Setup Time (INTACK Cycle)	140		100		[7]
17	TdIEI(IEOf)	IEI ↓ to IEO ↓ Delay		130		120	[5] CL = 50 pf
18	TdIEI(IEOr)	IEI ↑ to IEO ↑ Delay (after ED Decode)		160		150	[5]
19	TcIO(C)	$\overline{\text{IORQ}}$ ↑ to Clock ↓ Setup Time (To Activate READY on Next Clock Cycle)	200		170		
20	TdC(RDYr)	Clock ↓ to READY ↑ Delay		190		170	[5] CL = 50 pf
21	TdC(RDYf)	Clock ↓ to READY ↓ Delay		140		120	[5]
22	TwSTB	$\overline{\text{STROBE}}$ Pulse Width	150		120		[4]
23	TsSTB(C)	$\overline{\text{STROBE}}$ ↑ to Clock ↓ Setup Time (To Activate READY on Next Clock Cycle)	220		150		[5]
24	TdIO(PD)	$\overline{\text{IORQ}}$ ↑ to PORT DATA Stable Delay (Mode 0)		180		160	[5]
25	TsPD(STB)	PORT DATA to $\overline{\text{STROBE}}$ ↑ Setup Time (Mode 1)	230		190		
26	TdSTB(PD)	$\overline{\text{STROBE}}$ ↓ to PORT DATA Stable (Mode 2)		210		180	[5]
27	TdSTB(PDr)	$\overline{\text{STROBE}}$ ↑ to PORT DATA Float Delay (Mode 2)		180		160	CL = 50 pf
28	TdPD(INT)	PORT DATA Match to $\overline{\text{INT}}$ ↓ Delay (Mode 3)		490		430	
29	TdSTB(INT)	$\overline{\text{STROBE}}$ ↑ to $\overline{\text{INT}}$ ↓ Delay		440		350	

NOTES:
[1] TcC = TwCh + TwCl + TrC + TfC.
[2] Increase TdRI(DO) by 10 ns for each 50 pf increase in load up to 200 pf max.
[3] Increase TdIO(DOI) by 10 ns for each 50 pf, increase in loading up to 200 pf max.
[4] For Mode 2: TwSTB > TsPD(STB).
[5] Increase these values by 2 ns for each 10 pf increase in loading up to 100 pf max.
[6] TsCS(RI) may be reduced. However, the time subtracted from TsCS(RI) will be added to TdRI(DO).
* $\overline{\text{M1}}$ must be active for a minimum of two clock cycles to reset the PIO.
† Units in nanoseconds (ns).

*Clock-cycle time-dependent characteristics. See Footnotes to AC Characteristics.

AC TIMING DIAGRAM



Zilog **Product Specification**

January 1989

Z8430/Z84C30 NMOS/CMOS Z80 ®CTC Counter/Timer Circuit

FEATURES

- Four independently programmable counter/timer channels, each with a readable downcounter and a selectable 16 or 256 prescaler. Downcounters are reloaded automatically at zero count.
- Selectable positive or negative trigger initiates timer operation.
- Three channels have Zero Count/Timeout outputs capable of driving Darlington transistors.
- **NMOS version for high cost performance solutions.**
- **CMOS version for the designs requires low power consumption.**
- **NMOS Z0843004 - 4 MHz, Z0843006 - 6.17 MHz.**
- **CMOS Z84C3004 - DC to 4 MHz, Z84C3006 - DC to 6.17 MHz, Z84C3008 - DC to 8 MHz.**
- Interfaces directly to the Z80 CPU or—for baud rate generation—to the Z80 SIO.
- Standard Z80 Family daisy-chain interrupt structure provides fully vectored, prioritized interrupts without external logic. The CTC may also be used as an interrupt controller.
- **6 MHz version supports 6.144 MHz CPU clock operation.**

GENERAL DESCRIPTION

The Z80 CTC, hereinafter referred to as Z80 CTC or CTC, four-channel counter/timer can be programmed by system software for a broad range of counting and timing applications. The four independently programmable channels of the Z80 CTC satisfy common microcomputer system requirements for event counting, interrupt and interval timing, and general clock rate generation.

System design is simplified because the CTC connects directly to both the Z80 CPU and the Z80 SIO with no additional logic. In larger systems, address decoders and buffers may be required.

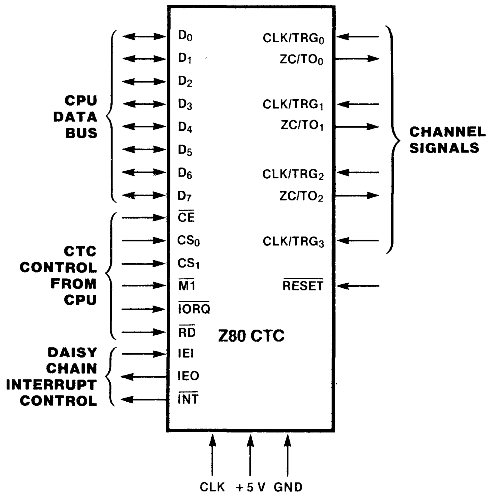


Figure 1. Pin Functions

Programming the CTC is straightforward: each channel is programmed with two bytes; a third is necessary when interrupts are enabled. Once started, the CTC counts down, automatically reloads its time constant, and resumes counting. Software timing loops are completely eliminated. Interrupt processing is simplified because only one vector need be specified; the CTC internally generates a unique vector for each channel.

The Z80 CTC requires a single +5%V power supply and the standard Z80 single-phase system clock. It is packaged in 28-pin DIPs, a 44-pin plastic chip carrier, and a 44-pin Quad Flat Pack. (Figures 2a, 2b, and 2c). Note that the QFP package is only available for CMOS versions.

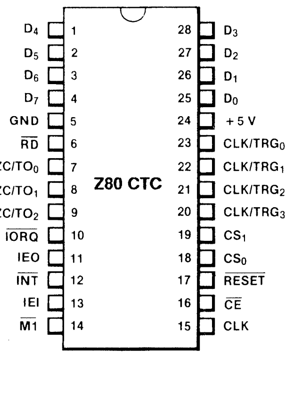


Figure 22a. Pin Assignments

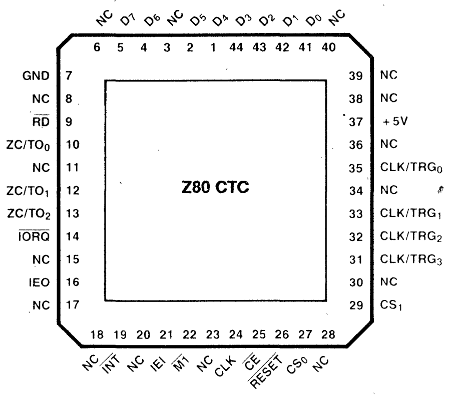


Figure 2b. 44-pin Chip Carrier, Pin Assignments

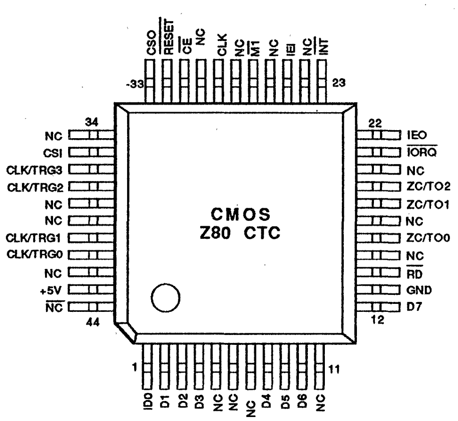


Figure 2C. 44-Pin Quad Flat Pack Pin Assignments

FUNCTIONAL DESCRIPTION

The Z80 CTC has four independent counter/timer channels. Each channel is individually programmed with two words: a control word and a time-constant word. The control word selects the operating mode (counter or timer), enables or disables the channel interrupt, and selects certain other operating parameters. If the timing mode is selected, the control word also sets a prescaler, which divides the system clock by either 16 or 256. The time-constant word is a value from 1 to 256.

During operation, the individual counter channel counts down from the preset time constant value. In counter mode operation the counter decrements on each of the CLK/TRG input pulses until zero count is reached. Each decrement is synchronized by the system clock. For counts greater than 256, more than one counter can be cascaded. At zero count, the down-counter is automatically reset with the time constant value.

The timer mode determines time intervals as small as 2 µs(8 MHz), 3 µs (6 MHz), or 4 µs (4MHz) without additional logic or software timing loops. Time intervals are generated by dividing the system clock with a prescaler that decrements a preset down-counter.

Thus, the time interval is an integral multiple of the clock period, the prescaler value (16 or 256), and the time constant that is preset in the down-counter. A timer is triggered automatically when its time constant value is programmed, or by an external CLK/TRG input.

Three channels have two outputs that occur at zero count. The first output is a zero-count/timeout pulse at the ZC/TO output. The fourth channel (Channel 3) does not have a ZC/TO output; interrupt request is the only output available from Channel 3.

The second output is Interrupt Request ($\overline{\text{INT}}$), which occurs if the channel has its interrupt enabled during programming. When the Z80 CPU acknowledges Interrupt Request, the Z80 CTC places an interrupt vector on the data bus.

The four channels of the Z80 CTC are fully prioritized and fit into four contiguous slots in a standard Z80 daisy-chain interrupt structure. Channel 0 is the highest priority and Channel 3 the lowest. Interrupts can be individually enabled (or disabled) for each of the four channels.

INTERNAL STRUCTURE

The CTC has four major elements, as shown in Figure 3.

- CPU bus I/O
- Channel control logic
- Interrupt logic
- Counter/timer circuits

CPU Bus I/O. The CPU bus I/O circuit decodes the address inputs, and interfaces the CPU data and control signals to the CTC for distribution on the internal bus.

Internal Control Logic. The CTC internal control logic controls overall chip operating functions such as the chip enable, reset, and read/write logic.

Interrupt Logic. The interrupt control logic ensures that the CTC interrupts interface properly with the Z80 CPU interrupt system. The logic controls the interrupt priority of the CTC as a function of the IEI signal. If IEI is High, the CTC has priority. During interrupt processing, the interrupt logic holds IEO Low, which inhibits the interrupt operation on lower priority devices. If the IEI input goes Low, priority is relinquished and the interrupt logic drives IEO Low.

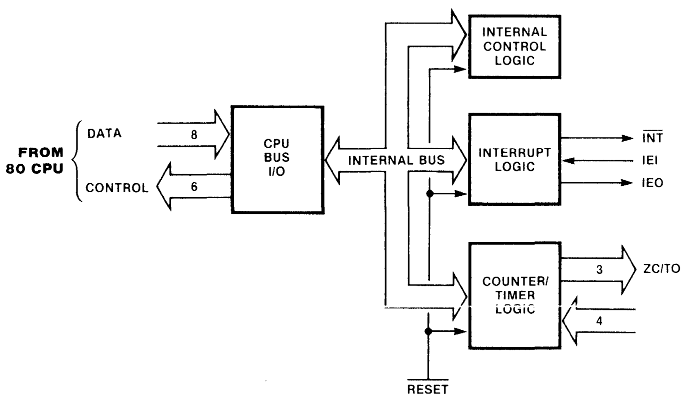


Figure 3. Functional Block Diagram

If a channel is programmed to request an interrupt, the interrupt logic drives IEO Low at the zero count, and generates an $\overline{\text{INT}}$ signal to the Z80 CPU. When the Z80 CPU responds with interrupt acknowledge ($\overline{\text{M1}}$ and $\overline{\text{IORQ}}$), then the interrupt logic arbitrates the CTC internal priorities, and the interrupt control logic places a unique interrupt vector on the data bus.

If an interrupt is pending, the interrupt logic holds IEO Low. When the Z80 CPU issues a Return From Interrupt (RETI) instruction, each peripheral device decodes the first byte (ED_{16}). If the device has a pending interrupt, it raises IEO (High) for one $\overline{\text{M1}}$ cycle. This ensures that all lower priority devices can decode the entire RETI instruction and reset properly.

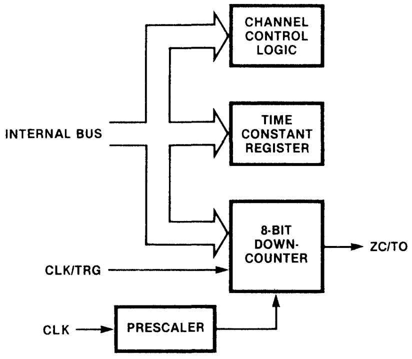


Figure 4. Counter/Timer Block Diagram

Counter/Timer Circuits. The CTC has four independent counter/timer circuits, each containing the logic shown in Figure 4.

Channel Control Logic. The channel control logic receives the 8-bit channel control word when the counter/timer channel is programmed. The channel control logic decodes the control word and sets the following operating conditions:

- Interrupt enable (or disable)
- Operating mode (timer or counter)
- Timer mode prescaler factor (16 or 256)
- Active slope for CLK/TRG input
- Timer mode trigger (automatic or CLK/TRG input)
- Time constant data word to follow
- Software reset

Time Constant Register. When the counter/timer channel is programmed, the time constant register receives and stores an 8-bit time constant value, which can be anywhere from 1 to 256 (0 = 256). This constant is automatically loaded into the down-counter when the counter/timer channel is initialized, and subsequently after each zero count.

Prescaler. The prescaler, which is used only in timer mode, divides the system clock frequency by a factor of either 16 or 256. The prescaler output clocks the down-counter during timer operation. The effect of the prescaler on the down-counter is a multiplication of the system clock period by 16 or 256. The prescaler factor is programmed by bit 5 of the channel control word.

Down-Counter. Prior to each count cycle, the down-counter is loaded with the time constant register contents. The counter is then decremented one of two ways, depending on operating mode:

- By the prescaler output (timer mode)
- By the trigger pulses into the CLK/TRG input (counter mode)

Without disturbing the down-count, the Z80 CPU can read the count remaining at any time by performing an I/O read operation at the port address assigned to the CTC channel. When the down-counter reaches the zero count, the ZC/TO output generates a positive-going pulse. When the interrupt is enabled, zero count also triggers an interrupt request signal ($\overline{\text{INT}}$) from the interrupt logic.

PROGRAMMING

Each Z80 CTC channel must be programmed prior to operation. Programming consists of writing two words to the I/O port that corresponds to the desired channel. The first word is a control word that selects the operating mode and other parameters; the second word is a time constant, which is a binary data word with a value from 1 to 256. A time constant word must be preceded by a channel control word.

After initialization, channels may be reprogrammed at any time. If updated control and time constant words are written to a channel during the count operation, the count continues to zero before the new time constant is loaded into the counter.

If the interrupt on any Z80 CTC channel is enabled, the programming procedure should also include an interrupt vector. Only one vector is required for all four channels, because the interrupt logic automatically modifies the vector for the channel requesting service.

A control word is identified by a 1 in bit 0. A 1 in bit 2 indicates a time constant word is to follow. Interrupt vectors are always addressed to Channel 0, and identified by a 0 in bit 0.

Addressing. During programming, channels are addressed with the channel select pins CS_1 and CS_2. A 2-bit binary code selects the appropriate channel as shown in the following table.

Channel	CS_1	CS_0
0	0	0
1	0	1
2	1	0
3	1	1

Reset. The CTC has both hardware and software resets. The hardware reset terminates all down-counts and disables all CTC interrupts by resetting the interrupt bits in the control registers. In addition, the ZC/TO and Interrupt outputs go inactive, IEO reflects IEI, and D_0-D_7 go to the high-impedance state. All channels must be completely reprogrammed after a hardware reset.

The software reset is controlled by bit 1 in the channel control word. When a channel receives a software reset, it stops counting. When a software reset is used, the other bits in the control word also change the contents of the channel control register. After a software reset a new time constant word must be written to the same channel.

If the channel control word has both bits D_1 and D_2 set to 1, the addressed channel stops operating, pending a new time constant word. The channel is ready to resume after the new constant is programmed. In timer mode, if D_3 = 0, operation is triggered automatically when the time constant word is loaded.

Channel Control Word Programming. The channel control word is shown in Figure 5. It sets the modes and parameters described below.

Interrupt Enable. D_7 enables the interrupt, so that an interrupt output ($\overline{\text{INT}}$) is generated at zero count. Interrupts may be programmed in either mode and may be enabled or disabled at any time.

Mode. D_6 selects either timer or counter operating mode.

Prescaler Factor. (Timer Mode Only). D_5 selects factor—either 16 or 256.

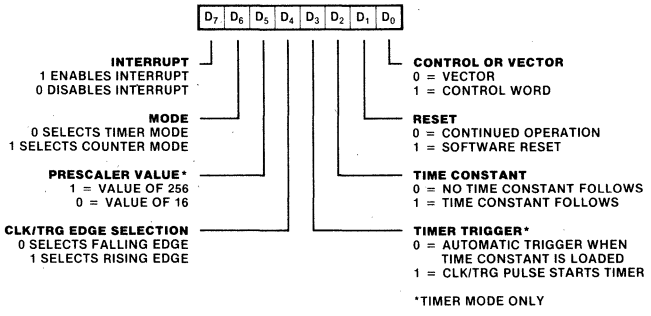


Figure 5. Channel Control Word

Clock/Trigger Edge Selector. D_4 selects the active edge or slope of the CLK/TRG input pulses. Note that reprogramming the CLK/TRG slope during operation is equivalent to issuing an active edge. If the trigger slope is changed by a control word update while a channel is pending operation in timer mode, the result is the same as a CLK/TRG pulse and the timer starts. Similarly, if the channel is in counter mode, the counter decrements.

Timer Trigger (Timer Mode Only). D_3 selects the trigger mode for timer operation. When D_3 is reset to 0, the timer is triggered automatically. The time constant word is programmed during an I/O write operation, which takes one machine cycle. At the end of the write operation there is a setup delay of one clock period. The timer starts automatically (decrements) on the rising edge of the second clock pulse (T_2) of the machine cycle following the write operation. Once started, the timer runs continuously. At zero count the timer reloads automatically and continues counting without interruption or delay, until stopped by a reset.

When D_3 is set to 1, the timer is triggered externally through the CLK/TRG input. The time constant word is programmed during an I/O write operation, which takes one machine cycle. The timer is ready for operation on the rising edge of the second clock pulse (T_2) of the following machine cycle. Note that the first timer decrement follows the active edge of the CLK/TRG pulse by a delay time of one clock cycle if a minimum setup time to the rising edge of clock is met. If this minimum is not met, the delay is extended by another clock period. Consequently, for immediate triggering, the CLK/TRG input must precede T_2 by one clock cycle plus its minimum setup time. If the minimum time is not met, the timer will start on the third clock cycle (T_3).

Once started the timer operates continuously, without interruption or delay, until stopped by a reset.

Time Constant. A 1 in D_2 indicates that the next word addressed to the selected channel is a time constant data word for the time constant register. The time constant word may be written at any time.

A 0 in D_2 indicates no time constant word is to follow. This is ordinarily used when the channel is already in operation and the new channel control word is an update. A channel will not operate without a time constant value. The only way to write a time constant value is to write a control word with D_2 set.

Software Reset. Setting D_1 to 1 causes a software reset, which is described in the Reset section.

Control Word. Setting D_0 to 0 identifies the word as a control word.

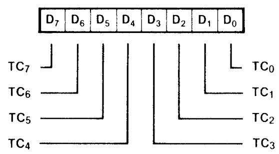


Figure 6. Time Constant Word

Time Constant Programming. Before a channel can start counting it must receive a time constant word from the CPU. During programming or reprogramming, a channel control word in which bit 2 is set must precede the time constant word to indicate that the next word is a time constant. The time constant word can be any value from 1 to 256 (Figure 6). Note that 00_{16} is interpreted as 256.

In timer mode, the time interval is controlled by three factors:

- The system clock period (CLK)
- The prescaler factor (P), which multiplies the interval by either 16 or 256
- The time constant (T), which is programmed into the time constant register

Consequently, the time interval is the product of CLK $\times$ P $\times$ T. The minimum timer resolution is $16 \times$ CLK ($4\mu s$ with a 4MHz clock). The maximum timer interval is $256 \times \text{CLK} \times 256$ (16.4 ms with a 4MHz clock). For longer intervals timers may be cascaded.

Interrupt Vector Programming. If the Z80 CTC has one or more interrupts enabled, it can supply interrupt vectors to the Z80 CPU. To do so, the Z80 CTC must be pre-programmed with the most-significant five bits of the interrupt vector. Programming consists of writing a vector word to the I/O port corresponding to the Z80 CTC Channel 0. Note that D_0 of the vector word is always zero, to distinguish the vector from a channel control word. D_1 and D_2 are not used in programming the vector word. These bits are supplied by the interrupt logic to identify the channel requesting interrupt service with a unique interrupt vector (Figure 7). Channel 0 has the highest priority.

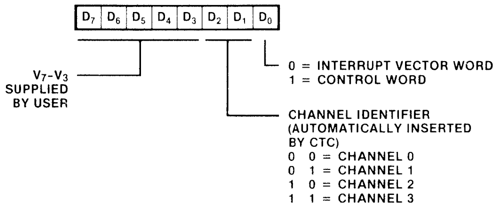


Figure 7. Interrupt Vector Word

PIN DESCRIPTION

$\overline{\text{CE}}$. *Chip Enable* (input, active Low). When enabled the CTC accepts control words, interrupt vectors, or time constant data words from the data bus during an I/O write cycle; or transmits the contents of the downcounter to the CPU during an I/O read cycle. In most applications this signal is decoded from the eight least significant bits of the address bus for any of the four I/O port addresses that are mapped to the four counter-timer channels.

CLK. *System Clock* (input). Standard single-phase Z80 system clock.

CLK/TRG$_0$-CLK/TRG$_3$. *External Clock/Timer Trigger* (input, user-selectable active High or Low). Four pins corresponding to the four Z80 CTC channels. In counter mode, every active edge on this pin decrements the downcounter. In timer mode, an active edge starts the timer.

CS$_0$-CS$_1$. *Channel Select* (inputs active High). Two-bit binary address code selects one of the four CTC channels for an I/O write or read (usually connected to A_0 and A_1).

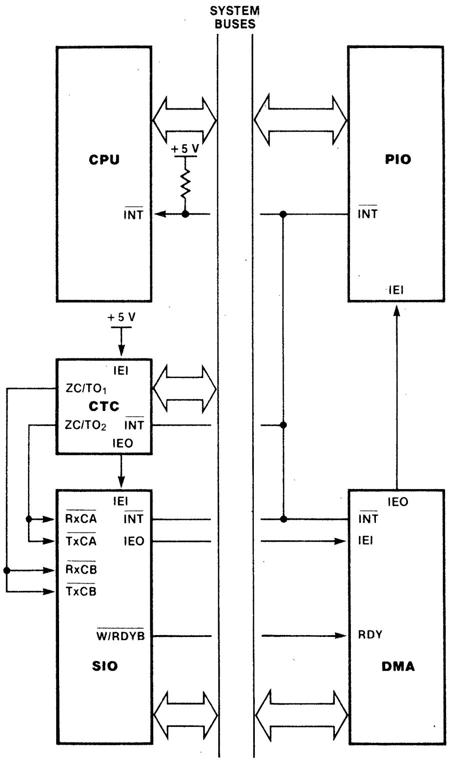


Figure 8. A Typical Z80 Environment

D$_0$-D$_7$. *System Data Bus* (bidirectional, 3-state). Transfers all data and commands between the Z80 CPU and the Z80 CTC.

IEI. *Interrupt Enable In* (input, active High). A High indicates that no other interrupting devices of higher priority in the daisy chain are being serviced by the Z80 CPU.

IEO. *Interrupt Enable Out* (output, active High). High only if IEI is High and the Z80 CPU is not servicing an interrupt from any Z80 CTC channel. IEO blocks lower priority devices from interrupting while a higher priority interrupting device is being serviced.

$\overline{\text{INT}}$. *Interrupt Request* (output, open drain, active Low). Low when any Z80 CTC channel that has been programmed to enable interrupts as a zero-count condition in its downcounter.

$\overline{\text{IORQ}}$. *Input/Output Request* (input from CPU, active Low). Used with $\overline{\text{CE}}$ and $\overline{\text{RD}}$ to transfer data and channel control words between the Z80 CPU and the Z80 CTC. During a write cycle, $\overline{\text{IORQ}}$ and $\overline{\text{CE}}$ are active and $\overline{\text{RD}}$ inactive. The Z80 CTC does not receive a specific write signal; rather, it internally generates is own from the inverse of an active $\overline{\text{RD}}$ signal. In a read cycle, $\overline{\text{IORQ}}$, $\overline{\text{CE}}$, and $\overline{\text{RD}}$ are active; the contents of the downcounter are read by the Z80 CPU. If $\overline{\text{IORQ}}$ and $\overline{\text{M1}}$ are both true, the CPU is acknowledging an interrupt request, and the highest priority interrupting channel places its interrupt vector on the Z80 data bus.

$\overline{\text{M1}}$. *Machine Cycle One* (input from CPU, active Low). When $\overline{\text{M1}}$ and $\overline{\text{IORQ}}$ are active, the Z80 CPU is acknowledging an interrupt. The Z80 CTC then places an interupt vector on the data bus if it has highest priority, and if a channel has requested an interrupt ($\overline{\text{INT}}$).

$\overline{\text{RD}}$. *Read Cycle Status* (input, active Low). Used in conjunction with $\overline{\text{IORQ}}$ and $\overline{\text{CE}}$ to transfer data and channel control words between the Z80 CPU and the Z80 CTC.

$\overline{\text{RESET}}$. *Reset* (input active Low). Terminates all down-counts and disables all interrupts by resetting the interrupt bits in all control registers; the ZC/TO and the interrupt outputs go inactive; IEO reflects IEI; D_0-D_7 go to the high-impedance state.

ZC/TO$_0$-ZC/TO$_2$. *Zero Count/Timeout* (output, active High). Three ZC/TO pins corresponding to Z80 CTC channels 2 through 0 (Channel 3 has no ZC/TO pin). In both counter and timer modes the output is an active High pulse when the downcounter decrements to zero.

TIMING

Read Cycle Timing. Figure 9 shows read cycle timing. This cycle reads the contents of a down-counter without disturbing the count. During clock cycle T_2, the Z80 CPU initiates a read cycle by driving the following inputs Low: $\overline{RD}$, $\overline{IORQ}$, and $\overline{CE}$. A 2-bit binary code at inputs CS_1 and CS_0 selects the channel to be read. $\overline{M1}$ must be High to distinguish this cycle from an interrupt acknowledge.

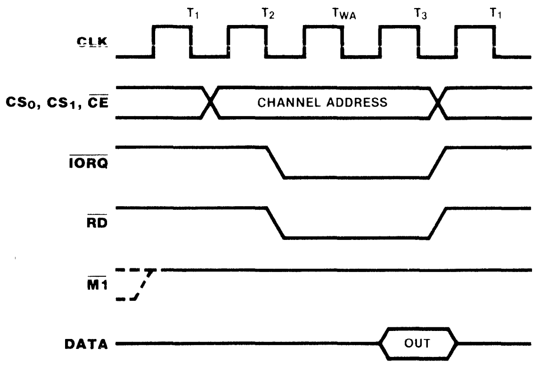


Figure 9. Read Cycle Timing

Write Cycle Timing. Figure 10 shows write cycle timing for loading control, time constant, or vector words.

The CTC does not have a write signal input, so it generates one internally when the read ($\overline{RD}$) input is High during T_1. During T_2 $\overline{IORQ}$ and $\overline{CE}$ inputs are Low. $\overline{M1}$ must be High to distinguish a write cycle from an interrupt acknowledge. A 2-bit binary code at inputs CS_1 and CS_0 selects the channel to be addressed, and the word being written is placed on the Z80 data bus. The data word is latched into the appropriate register with the rising edge of clock cycle T_3.

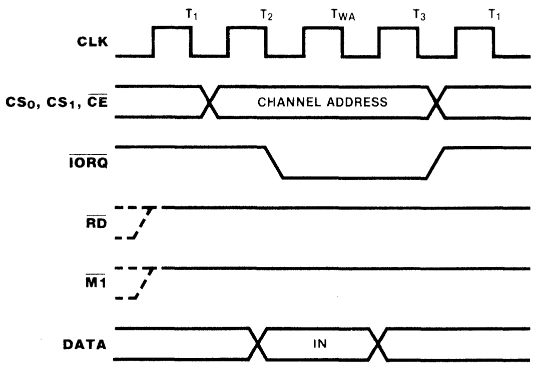


Figure 10. Write Cycle Timing

Timer Operation. In the timer mode, a CLK/TRG pulse input starts the timer (Figure 11) on the second succeeding rising edge of CLK. The trigger pulse is asynchronous, and it must have a minimum width. A minimum lead time (210 ns) is required between the active edge of the CLK/TRG and the next rising edge of CLK to enable the prescaler on the following clock edge. If the CLK/TRG edge occurs closer than this, the initiation of the timer function is delayed one clock cycle. This corresponds to the start-up timing discussed in the programming section. The timer can also be started automatically if so programmed by the channel control word.

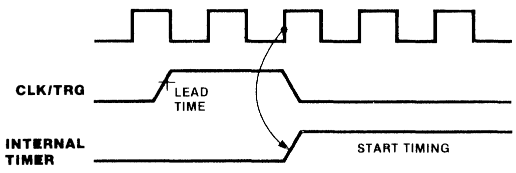


Figure 11. Timer Mode Timing

Counter Operation. In the counter mode, the CLK/TRG pulse input decrements the downcounter. The trigger is asynchronous, but the count is synchronized with CLK. For the decrement to occur on the next rising edge of CLK, the trigger edge must precede CLK by a minimum lead time as shown in Figure 12. If the lead time is less than specified, the count is delayed by one clock cycle. The trigger pulse must have a minimum width, and the trigger period must be at least twice the clock period. If the trigger repetition rate is faster than 1/3 the clock frequency, then TsCTR(Cs), AC Characteristics Specification 26, must be met.

The ZC/TO output occurs immediately after zero count, and follows the rising CLK edge.

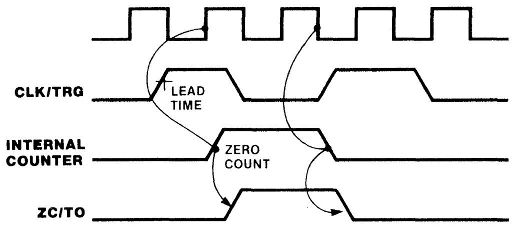


Figure 12. Counter Mode Timing

INTERRUPT OPERATION

The Z80 CTC follows the Z80 system interrupt protocol for nested priority interrupts and return from interrupt, wherein the interrupt priority of a peripheral is determined by its location in a daisy chain. Two lines—IEI and IEO—in the CTC connect it to the system daisy chain. The device closest to the +5V supply has the highest priority (Figure 13). For additional information on the Z80 interrupt structure, refer to the *Z80 CPU Product Specification* and the *Z80 CPU Technical Manual.*

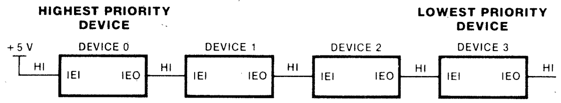


Figure 13. Daisy-Chain Interrupt Priorities

Within the Z80 CTC, interrupt priority is predetermined by channel number: Channel 0 has the highest priority, and Channel 3 the lowest. If a device or channel is being serviced with an interrupt routine, it cannot be interrupted by a device or channel with lower priority until service is complete. Higher priority devices or channels may interrupt the servicing of lower priority devices or channels.

A Z80 CTC channel may be programmed to request an interrupt every time its downcounter reaches zero. Note that the CPU must be programmed for interrupt mode 2. Some time after the interrupt request, the CPU sends an interrupt acknowledge. The CTC interrupt control logic determines the highest priority channel that is requesting an interrupt. Then, if the CTC IEI input is High (indicating that it has priority within the system daisy chain) it places an 8-bit interrupt vector on the system data bus. The high-order five bits of this vector were written to the CTC during the programming process; the next two bits are provided by the CTC interrupt control logic as a binary code that identifies the highest priority channel requesting an interrupt; the low-order bit is always zero.

Interrupt Acknowledge Timing. Figure 14 shows interrupt acknowledge timing. After an interrupt request, the Z80 CPU sends an interrupt acknowledge ($\overline{M1}$ and $\overline{IORQ}$). All channels are inhibited from changing their interrupt request status when $\overline{M1}$ is active—about two clock cycles earlier than $\overline{IORQ}$. $\overline{RD}$ is High to distinguish this cycle from an instruction fetch.

The CTC interrupt logic determines the highest priority channel requesting an interrupt. If the CTC interrupt enable input (IEI) is High, the highest priority interrupting channel within the CTC places its interrupt vector on the data bus when $\overline{IORQ}$ goes Low. Two wait states (T_{WA}) are automatically inserted at this time to allow the daisy chain to stabilize. Additional wait states may be added.

Return from Interrupt Timing. At the end of an interrupt service routine the RETI (Return From Interrupt) instruction initializes the daisy chain enable lines for proper control of nested priority interrupt handling. The CTC decodes the 2-byte RETI code internally and determines whether it is intended for a channel being serviced. Figure 15 shows RETI timing.

If several Z80 peripherals are in the daisy chain, IEI settles active (High) on the chip currently being serviced when the opcode ED_{16} is decoded. If the following opcode is $4D_{16}$, the peripheral being serviced is released and its IEO becomes active. Additional wait states are allowed.

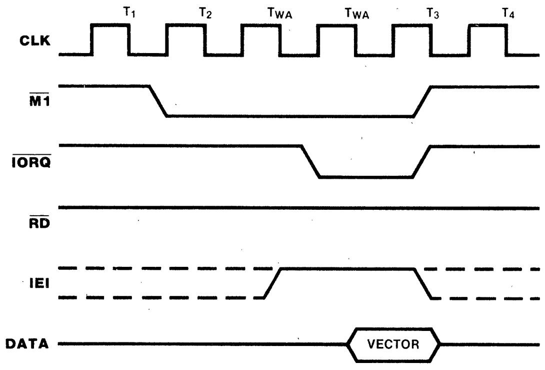


Figure 14. Interrupt Acknowledge Timing

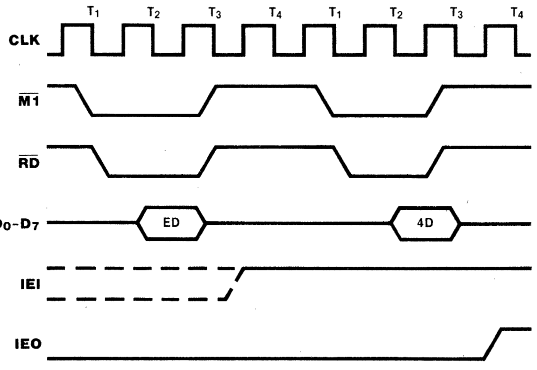


Figure 15. Return From Interrupt Timing

ABSOLUTE MAXIMUM RATINGS

Voltages on V_{CC} with respect to V_{SS}	−0.3V to +7.0V
Voltages on all inputs with respect to V_{SS}	−0.3V to V_{CC} + 0.3V
Storage Temperature	−65°C to +150°C

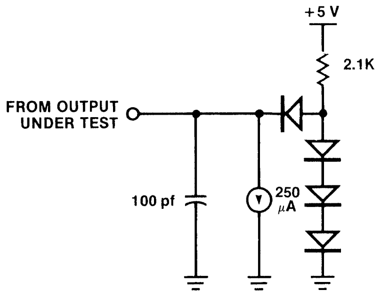
Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above these indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The characteristics below apply for the following test conditions, unless otherwise noted. All voltages are referenced to GND (0V). Positive current flows into the referenced pin. Available operating temperature range is:

- **S = 0°C to +70°C, V_{cc} Range**
  NMOS: +4.75V < V_{cc} < +5.25V
  CMOS: +4.50V < V_{cc} <+5.50V
- **E = -40°C to 100°C, +4.50V < V_{cc} < +5.50V**

The Ordering Information section lists package temperature ranges and product numbers. Refer to the Literature List for additional documentation. Package drawings are in the Package Information section.



DC CHARACTERISTICS (Z84C30/CMOS Z80 CTC)

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	−0.3	+0.45	V	
V_{IHC}	Clock Input High Voltage	V_{CC} − 0.6	V_{CC} + 0.3	V	
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	+2.2	V_{CC}	V	
V_{OL}	Output Low Voltage		+0.4	V	I_{OL} = 2.0 mA
V_{OH1}	Output High Voltage	+2.4		V	I_{OH} = −1.6 mA
V_{OH2}	Output High Voltage	V_{CC} − 0.8		V	I_{OH} = −250 μA
I_{LI}	Input Leakage Current		±10	μA	V_{IN} = 0.4 to V_{CC}
I_{LO}	3-State Output Leakage Current in Float		±10	μA	V_{OUT} = 0.4 to V_{CC}
ICC_1	Power Supply Current **4 MHz**		7	mA	V_{CC} = 5V
	6 MHz		**8**		CLK = **4 MHz, 6 MHz, 8 MHz**
	8 MHz		**10**		V_{IH} = V_{CC} − 0.2V, V_{IL} = 0.2V
ICC_2	Standby Supply Current		10	μA	V_{CC} = 5V, CLK = (0), V_{IH} = V_{CC} − 0.2V, V_{IL} = 0.2V
I_{OHD}	Darlington Drive Current	−1.5	−5.0	mA	V_{OH} = 1.5V, R_{EXT} = 1.1K Ω

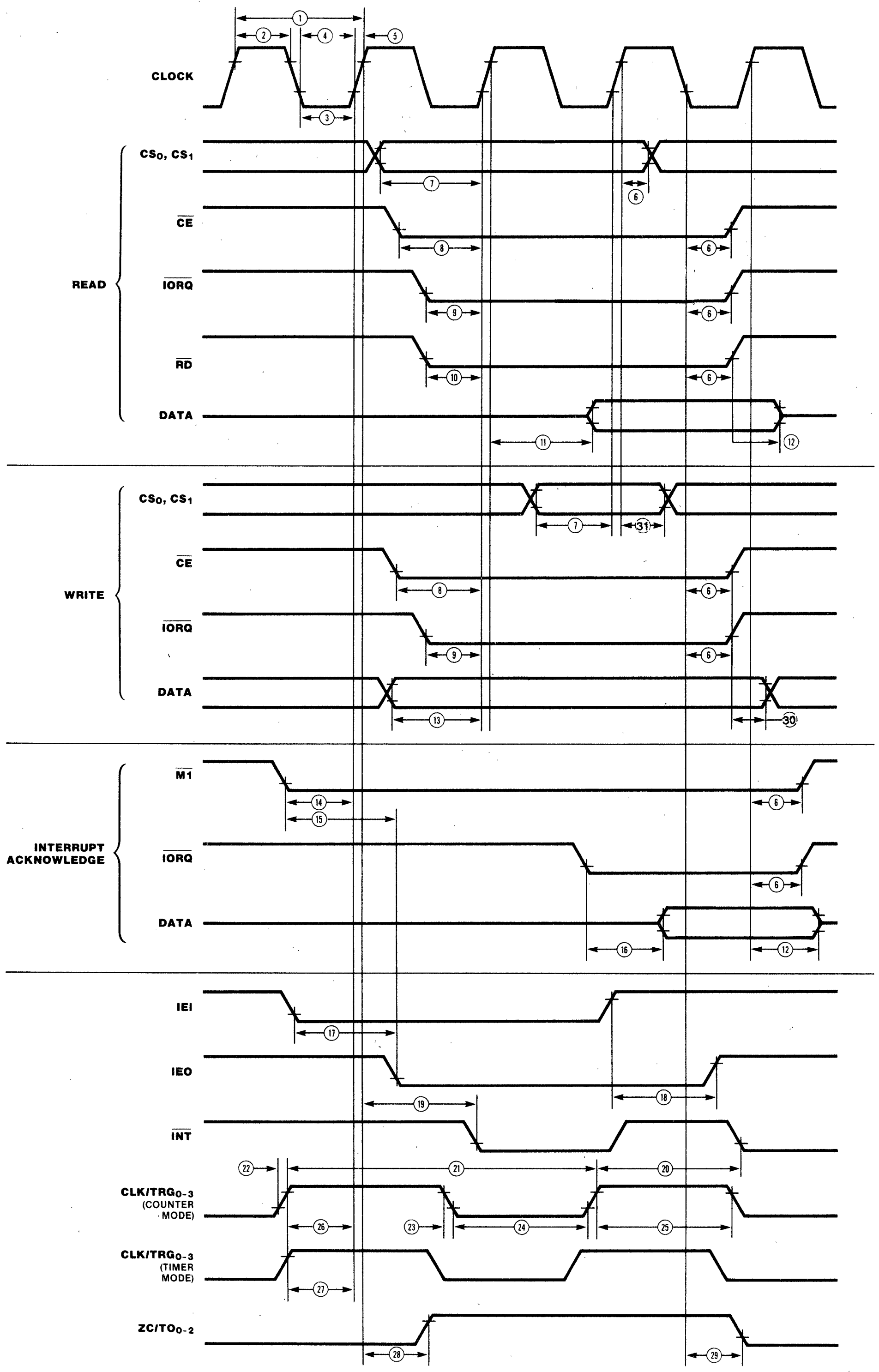
Over specified temperature and voltage range.

CAPACITANCE

Symbol	Parameter	Max	Unit
CLK	Clock Capacitance	**10**	pf
C_{IN}	Input Capacitance	**10**	pf
C_{OUT}	Output Capacitance	15	pf

T_A = 25°C, f = 1 MHz
Unmeasured pins returned to ground.

AC CHARACTERISTICS (Z84C30/CMOS Z80 CTC)



AC CHARACTERISTICS (Z84C30/CMOS Z80 CTC Continued)

			Z84C3004		Z84C3006		Z84C3008		
No.	Symbol	Parameter	Min(ns)	Max(ns)	Min(ns)	Max(ns)	Min(ns)	Max(ns)	Notes*
1	TcC	Clock Cycle Time	250	DC[1]	162	DC[1]	125	DC	
2	TwCh	Clcok Pulse Width (High)	105	DC	65	DC	55	DC	
3	TwCl	Clock Pulse Width (Low)	105	DC	65	DC	55	DC	
4	TfC	Clock Fall Time		30		20		10	
5	TrC	Clock Rise Time		30		20		10	
6	Th	All Hold Times	0		0		0		
7	TsCS(C)	$\overline{CS}$ to Clock ↑ Setup Time	160		100		50		
8	TsCE(C)	$\overline{CE}$ to Clock ↑ Setup Time	150		100		50		
9	TsIO(C)	$\overline{IORQ}$ ↓ to Clock ↑ Setup Time	115		70		40		
10	TsRD(C)	$\overline{RD}$ ↓ to Clock ↑ Setup Time	115		70		40		
11	TdC(DO)	Clock ↑ to Data Out Delay		200		130		90	[2]
12	TdRIr(DOz)	$\overline{RD}$, $\overline{IORQ}$ ↑ to Data Out Float Delay		50		40		40	
13	TsDI(C)	Data In to Clock ↑ Setup Time	50		40		30		
14	TsM1(C)	$\overline{M1}$ to Clock ↑ Setup Time	90		70		50		
15	TdM1(EO)	$\overline{M1}$ ↓ to IEO ↓ Delay (Interrupt immediately preceeding $\overline{M1}$)		190		130		90	[3]
16	TdIO(DOI)	$\overline{IORQ}$ ↓ to Data Out Delay (INTA		160		110		80	[2],[6]
17	TdIEI(IEOf)	IEI ↓ to IEO ↓ Delay		130		100		70	[3]
18	TdIEI(IEOr)	IEI ↑ to IEO ↑ Delay (after ED Decode)		160		110		70	[3]
19	TdC(INT)	Clock ↑ to $\overline{INT}$ ↓ Delay		(TcC +140)		(TcC +120)		(TcC +100)	[4]
20	TdCLK(INT)	CLK/TRG ↑ to $\overline{INT}$ ↓							
		tsCTR(C) satisfied		(19)+(26)		(19)+(26)		(19)+(26)	[5]
		tsCTR(C) not satisfied		(1)+(19)+(26)		(1)+(19)+(26)		(1)+(19)+(26)	[5]
21	TcCTR	CLK/TRG Cycle Time	(2TcC)		(2TcC)		(2TcC)	[5]	
22	TrCTR	CLK/TRG Rise Time		50		40		30	
23	TfCTR	CLK/TRG Fall Time		50		40		30	
24	TwCTRl	CLK/TRG Width (Low)	200		120		90		
25	TwCTRh	CLK/TRG Width (High)	200		120		90		
26	TsCTR(Cs)	CLK/TRG ↑ to Clock ↑ Setup Time for Immediate Count	210		150		110	[5]	
27	TsCTR(Ct)	CLK/TRG ↑ to Clock ↑ Setup Time for enabling of Prescaler on following clock ↑	210		150		110	[4]	
28	TdC(ZC/TOr)	Clock ↑ to ZC/TO ↑ Delay		190		140		100	
29	TdC(ZC/TOf)	Clock ↓ to ZC/TO ↓ Delay		190		140		100	
30	ThRIr(D)	$\overline{RD}$, $\overline{IORQ}$ ↑ to Data Hold	20		20		10		
31	ThC (CS)	Clock ↑ to CS hold	20		20		20		

*RESET must be active for a minimum of 3 clock cycles.

NOTES:

[1] TcC = TwCh + TwCl + TrC + TfC.

[2] Increase delay by 10 ns for each 50 pf increase in loading, 200 pf maximum for data lines, and 100 pf for control lines.

[3] Increase delay by 2 ns for each 10 pf increase in loading, 100 pf maximum.

[4] Timer mode.

[5] Counter mode.

[6] 2.5 TcC > (n − 2) TdIEI(IEOf) + TdM1(IEO) + TsIEI(IO) + TTL buffer delay, if any.

DC CHARACTERISTICS (Z8430/NMOS Z80 CTC)

Symbol	Parameter	Min	Max	Unit	Condition
V_{ILC}	Clock Input Low Voltage	-0.3[c]	$+0.45$[a]	V	
V_{IHC}	Clock Input High Voltage	$V_{CC}-0.6$[a]	$V_{CC}+0.3$[b]	V	
V_{IL}	Input Low Voltage	-0.3[c]	$+0.8$[a]	V	
V_{IH}	Input High Voltage	$+2.2$[a]	V_{CC}[b]	V	
V_{OL}	Output Low Voltage		$+0.4$[a]	V	$I_{OL} = 2.0$ mA
V_{OH}	Output High Voltage	$+2.4$[a]		V	$I_{OH} = -250\ \mu A$
I_{CC}	Power Supply Current:		$+120$[a]	mA	
I_{LI}	Input Leakage Current		± 10[a]	μA	$V_{IN} = 0.4$ to V_{CC}
I_{LO}	3-State Output Leakage Current in Float		± 10[a]	μA	$V_{OUT} = 0.4$ to V_{CC}
I_{OHD}	Darlington Drive Current	-1.5[a]		mA	$V_{OH} = 1.5V$ $R_{EXT} = 390\Omega$

CAPACITANCE

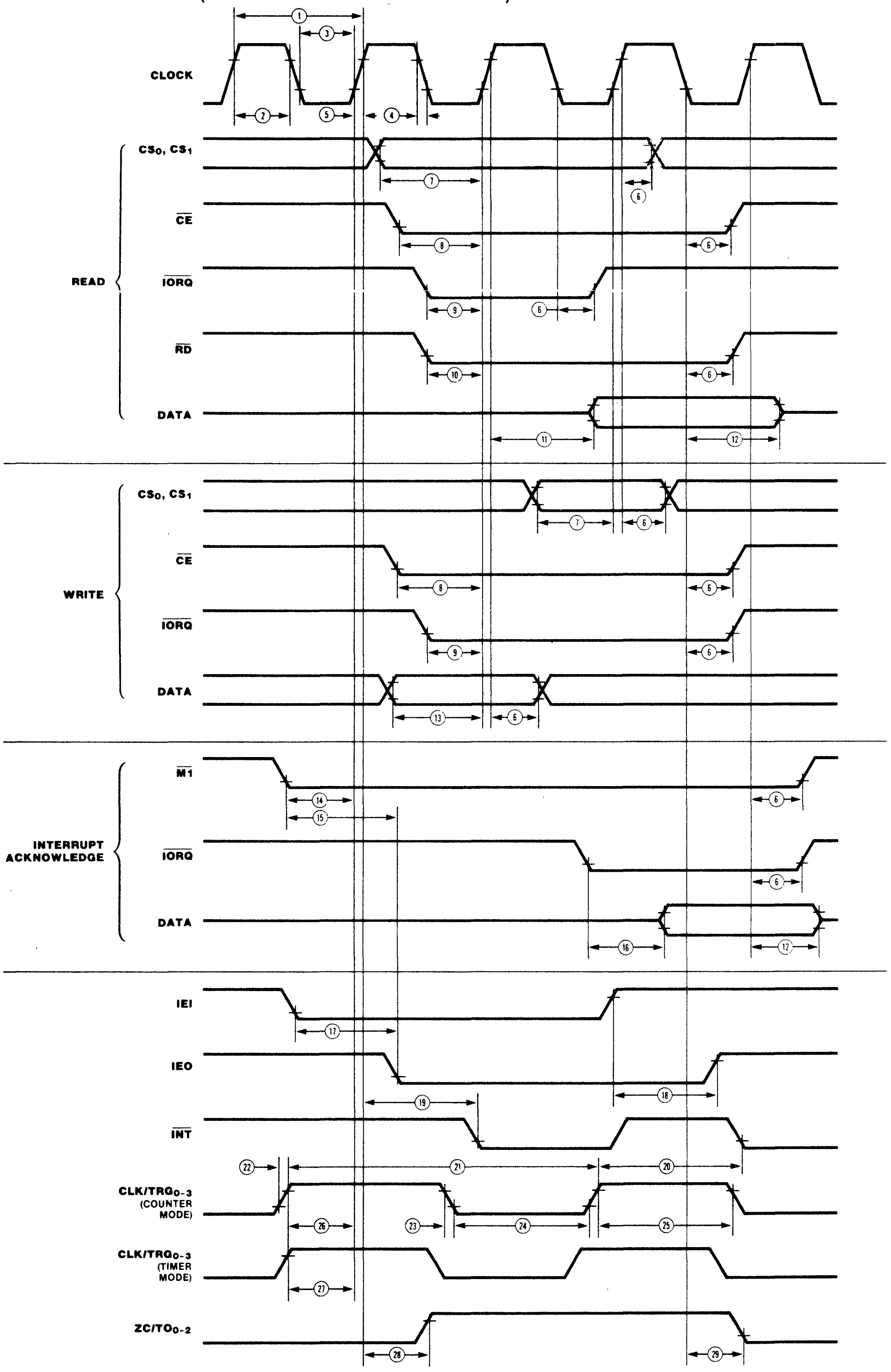
Symbol	Parameter	Max	Unit
CLK	Clock Capacitance	20[c]	pf
C_{IN}	Input Capacitance	5[c]	pf
C_{OUT}	Output Capacitance	15[c]	pf

$T_A = 25°C$, f = 1 MHz
Unmeasured pins returned to ground.

Parameter Test Status:

[a] Tested
[b] Guaranteed
[c] Guaranteed by characterization/design

AC CHARACTERISTICS (Z8430/NMOS Z80 CTC Continued)



AC CHARACTERISTICS (Z8430/NMOS Z80 CTC)

			Z0843004		Z0843006		
Number	Symbol	Parameter	Min	Max	Min	Max	Notes†
1	TcC	Clock Cycle Time	250	[1]	162	[1]	
2	TwCh	Clock Width (High)	105	2000	65	2000	
3	TwCl	Clock Width (Low)	105	2000	65	2000	
4	TfC	Clock Fall Time		30		20	
5	TrC	Clock Rise Time		30		20	
6	Th	All Hold Times	0		0		
7	TsCS(C)	CS to Clock ↑ Setup Time	160		100		
8	TsCE(C)	$\overline{\text{CE}}$ to Clock ↑ Setup Time	150		100		
9	TsIO(C)	$\overline{\text{IORQ}}$ ↓ to Clock ↑ Setup Time	115		70		
10	TsRD(C)	$\overline{\text{RD}}$ ↓ to Clock ↑ Setup Time	115		70		
11	TdC(DO)	Clock ↑ to Data Out Delay		200		130	[2]
12	TdC(DOz)	Clock ↓ to Data Out Float Delay		110		90	
13	TsDI(C)	Data In to Clock ↑ Setup Time	50		40		
14	TsM1(C)	$\overline{\text{M1}}$ to Clock ↑ Setup Time	90		70		
15	TdM1(IEO)	$\overline{\text{M1}}$ ↓ to IEO ↓ Delay (Interrupt immediately preceding M1)		190		130	[3]
16	TdIO(DOI)	$\overline{\text{IORQ}}$ ↓ to Data Out Delay (INTA Cycle)		160		110	[2]
17	TdIEI(IEOf)	IEI ↓ to IEO ↓ Delay		130		100	[3]
18	TdIEI(IEOr)	IEI ↑ to IEO ↑ Delay (After ED Decode)		160		110	[3]
19	TdC(INT)	Clock ↑ to $\overline{\text{INT}}$ ↓ Delay		(1) + 140		(1) + 120	[4,6]
20	TdCLK(INT)	CLK/TRG ↑ to $\overline{\text{INT}}$ ↓					
		tsCTR(C) satisfied		(19) + (26)		(19) + (26)	[5,6]
		tsCTR(C) not satisfied		(1) + (19) + (26)		(1) + (19) + (26)	[5,6]
21	TcCTR	CLK/TRG Cycle Time	2TcC		2TcC		[5]
22	TrCTR	CLK/TRG Rise Time		50		40	
23	TfCTR	CLK/TRG Fall Time		50		40	
24	TwCTRl	CLK/TRG Width (Low)	200		120		
25	TwCTRh	CLK/TRG Width (High)	200		120		

NOTES:
[1] TcC = TwCh + TwCl + TrC + TfC.
[2] Increase delay by 10 ns for each 50 pf increase in loading, 200 pf maximum for data lines, and 100 pf for control lines.
[3] Increase delay by 2 ns for each 10 pf increase in loading, 100 pf maximum.
[4] Timer mode
[5] Counter mode.
[6] Parenthetical numbers reference the table number of a parameter. e.g., (1) refers to TcC.

† 2.5 TcC > (n – 2) TDIEI(IEOf) + TDM1(IEO) + TsIEI(IO) + TTL buffer delay, if any. $\overline{\text{RESET}}$ must be active for a minimum of 3 clock cycles. Units are nanoseconds unless otherwise specified.

AC CHARACTERISTICS (Z8430/NMOS Z80 CTC Continued)

			Z0843004		Z0843006		
Number	Symbol	Parameter	Min	Max	Min	Max	Notes†
26	TsCTR(Cs)	CLK/TRG ↑ to Clock ↑ Setup Time for Immediate Count	210		150		[5]
27	TsCTR(Ct)	CLK/TRG ↑ to Clock ↑ Setup Time for enabling of Prescaler on following clock ↑	210		150		[4]
28	TdC(ZC/TOr)	Clock ↑ to ZC/TO ↑ Delay		190		140	
29	TdC(ZC/TOf)	Clock ↓ to ZC/TO ↓ Delay		190		140	

NOTES:
[1] TcC = TwCh + TwCl + TrC + TfC.
[2] Increase delay by 10 ns for each 50 pf increase in loading, 200 pf maximum for data lines, and 100 pf for control lines.
[3] Increase delay by 2 ns for each 10 pf increase in loading, 100 pf maximum.
[4] Timer mode
[5] Counter mode.
[6] Parenthetical numbers reference the table number of a parameter, e.g., (1) refers to TcC.
† 2.5 TcC > (n – 2) TDIEI(IEOf) + TDM1(IEO) + TsIEI(IO) + TTL buffer delay, if any. $\overline{\text{RESET}}$ must be active for a minimum of 3 clock cycles. Units are nanoseconds unless otherwise specified.

Zilog **Product Specification**

January 1989

Z8440/1/2/4, Z84C40/1/2/3/4 NMOS/ CMOS Z80® SIO Serial Input/Output Controller

FEATURES

- Two independent full-duplex channels, with separate control and status lines for modems or other devices.
- **Data rate in the x1 clock mode of 0 to 1.6M bits/ second with a 8.0 MHz clock.**
- **NMOS version for high cost performance solutions, CMOS version for the designs requires low power consumption.**
- **NMOS Z0844x04 - 4 MHz Z0844x06 - 6.17 MHz (Where x is the designator for the bonding option; 0, 1, 2 or 4)**
- **CMOS Z84C4x04 - DC 4 MHz Z84C4x06 - DC to 6.7 MHz Z84C4x08 - DC to 8 MHz (Where x is the designator for the bonding option; 0, 1, 2 or 3, 4)**
- **6 MHz version supports 6.144 MHz CPU clock operation.**
- Asynchronous protocols: everything necessary for complete messages in 5, 6, 7, or 8 bits/character. Includes variable stop bits and several clock-rate multipliers; break generation and detection; parity; overrun and framing error detection.
- Synchronous protocols: everything necessary for complete bit- or byte-oriented messages in 5, 6, 7, or 8 bits/character, including IBM Bisync, SDLC, HDLC, CCITT-X.25 and others. Automatic CRC generation/ checking, sync character and zero insertion/deletion, abort generation/detection, and flag insertion.
- Receiver data registers quadruply buffered, transmitter registers doubly buffered.
- Highly sophisticated and flexible daisy-chain interrupt vectoring for interrupts without external logic.

GENERAL DESCRIPTION

The Z80 SIO (here in after referred to as the Z80 SIO or, SIO). Serial Input/Output Controller is a dual-channel data communication interface with extraordinary versatility and capability. Its basic functions as a serial-to-parallel, parallel-to-serial converter/controller can be programmed by a CPU for a broad range of serial communication applications.

The device supports all common asynchronous and synchronous protocols, byte- or bit-oriented, and performs all of the functions traditionally done by UARTs, USARTs, and synchronous communication controllers combined, plus additional functions traditionally performed by the CPU. Moreover, it does this on two fully-independent channels, with an exceptionally sophisticated interrupt structure that allows very fast transfers.

Full interfacing is provided for CPU or DMA control. In addition to data communication, the circuit can handle virtually all types of serial I/O with fast, or slow, peripheral devices. While designed primarily as a member of the Z80 family, its versatility makes it well suited to many other CPUs.

The Z80 SIO uses a single +5V power supply and the standard Z80 family single-phase clock. The SIO/0, SIO/1, and SIO/2 are packaged in a 40-pin DIP, the SIO/4 is packaged in a 44-pin PCC and the SIO/3 is packaged in a 44-pin QFP. Note that SIO/3 is only available in CMOS and in QFP package.

PIN DESCRIPTION

Figures 1 through 6 illustrate the three 40-pin configurations (bonding options) available in the Z80C SIO (hereafter referred to as SIO or Z80 SIO). The constraints of a 40-pin package make it impossible to bring out the Receive Clock ($\overline{\text{RxC}}$), Transmit Clock ($\overline{\text{TxC}}$), Data Terminal Ready ($\overline{\text{DTR}}$) and Sync ($\overline{\text{SYNC}}$) signals for both channels. Therefore, either Channel B lacks a signal or two signals are bonded together:

- Z80 SIO/2 lacks $\overline{\text{SYNCB}}$
- Z80 SIO/1 lacks $\overline{\text{DTRB}}$
- Z80 SIO/0 has all four signals, but $\overline{\text{TxCB}}$ and $\overline{\text{RxCB}}$ are bonded together

The 44-pin package, the Z80 SIO/4 for PLCC package, and Z80 SIO/3 for QFP, has all options (Figure 7a and 7b).

The first bonding option above (SIO/2) is the preferred version for most applications. The pin descriptions are as follows:

B/$\overline{\textbf{A}}$. *Channel A or B Select* (input, High selects Channel B). This input defines which channel is accessed during a data

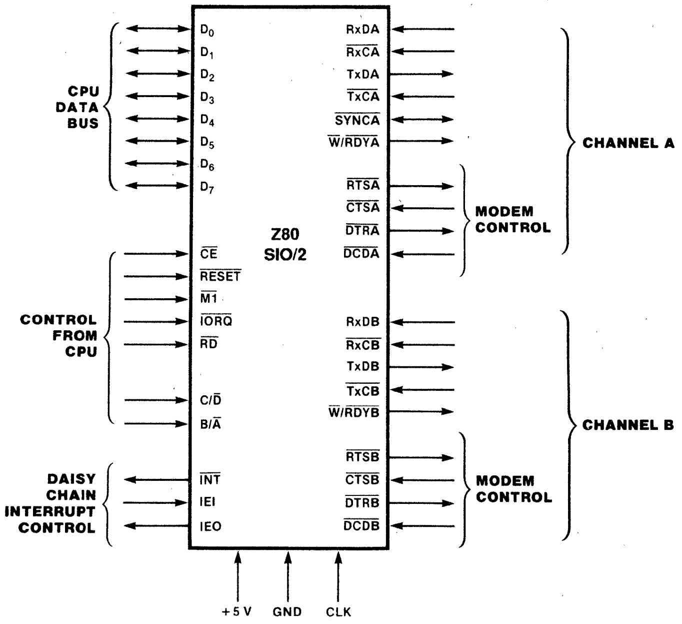


Figure 1. Pin Functions

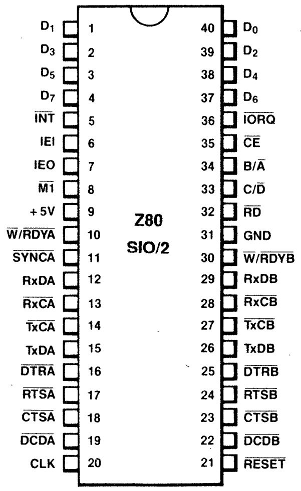


Figure 2. 40-pin Dual-In-Line Package (DIP), Pin Assignments

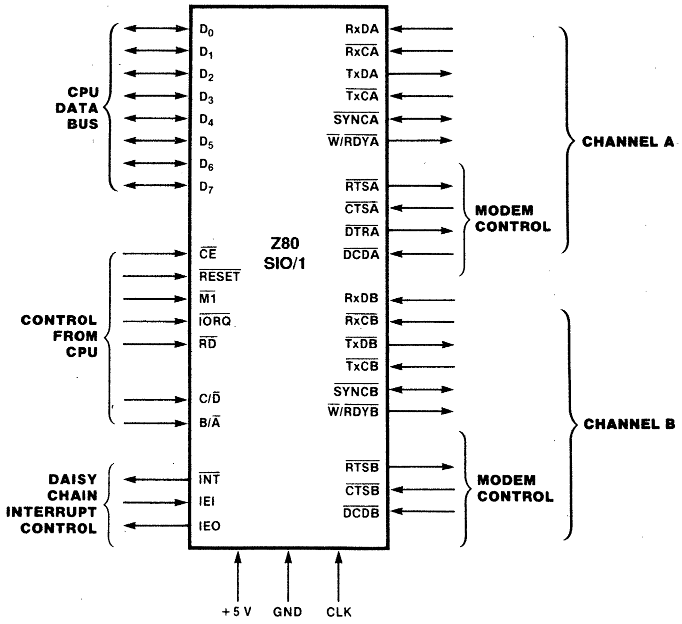


Figure 3. Pin Functions

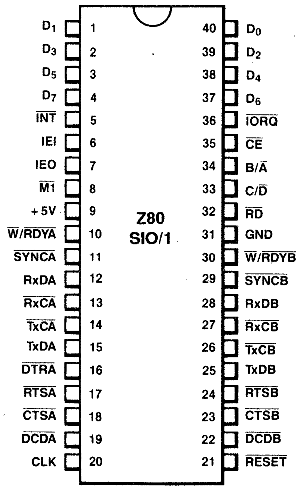


Figure 4. 40-pin Dual-In-Line Package (DIP), Pin Assignments

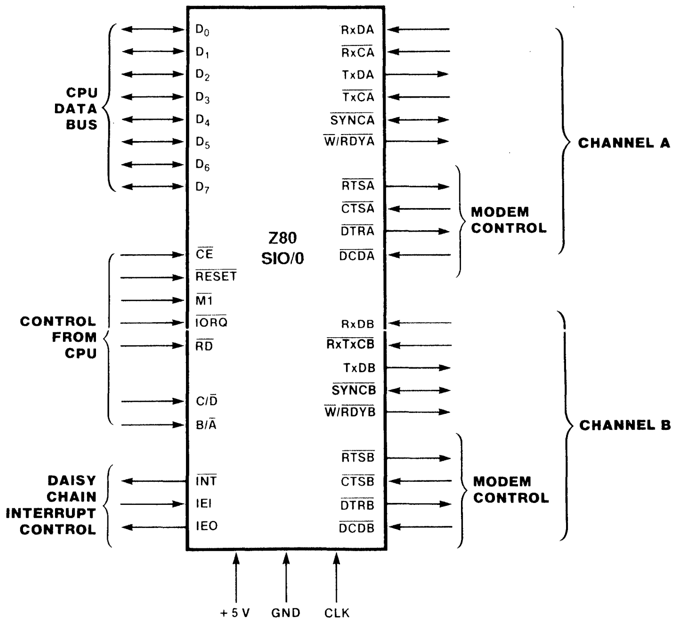


Figure 5. Pin Functions

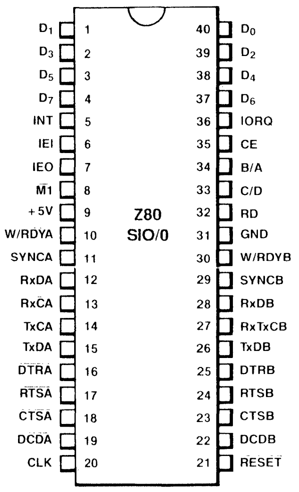


Figure 6. 40-pin Dual-In-Line Package (DIP), Pin Assignments

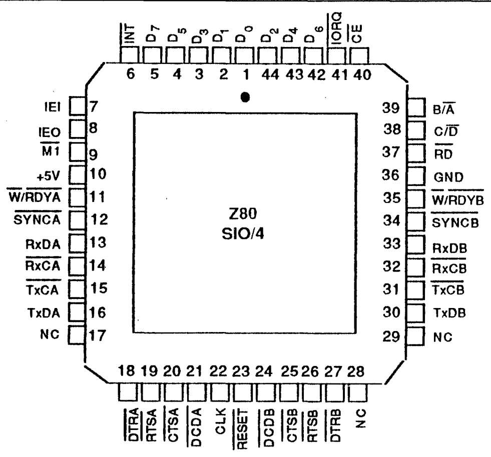


Figure 7a. 44-pin Chip Carrier, Pin Assignments

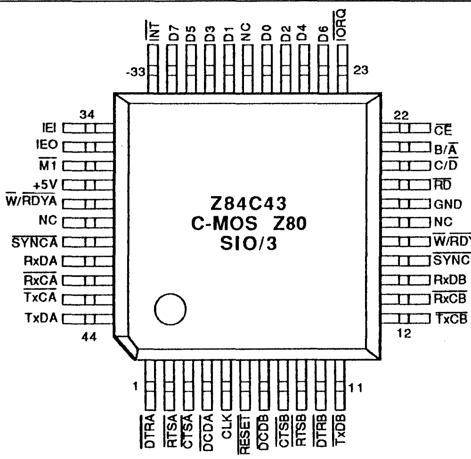


Figure 7b. 44-pin Quad Flat Pack Pin Assignments

transfer between the CPU and the SIO. Address bit A_0 from the CPU is often used for the selection function.

C/D̄. *Control or Data Select* (input, High selects Control). This input defines the type of information transfer performed between the CPU and the SIO. A High at this input during a CPU write to the SIO causes the information on the data bus to be interpreted as a command for the channel selected by B/Ā. A Low at C/D̄ means that the information on the data bus is data. Address bit A_1 is often used for this function.

C̄Ē. *Chip Enable* (Input, active Low). A Low level at this input enables the SIO to accept command or data input from the CPU during a write cycle, or to transmit data to the CPU during a read cycle.

CLK. *System Clock* (input). The SIO uses the standard Z8C System Clock to synchronize internal signals. This is single-phase clock.

$\overline{\text{CTSA}}$, $\overline{\text{CTSB}}$. *Clear To Send* (inputs, active Low). When programmed as Auto Enables, a Low on these inputs enables the respective transmitter. If not programmed as Auto Enables, these inputs may be programmed as general-purpose inputs. Both inputs are Schmitt-trigger buffered to accommodate slow-risetime signals. The SIO detects pulses on these inputs and interrupts the CPU on both logic level transitions. The Schmitt-trigger buffering does not guarantee a specified noise-level margin.

D_0-D_7. *System Data Bus* (bidirectional, 3-state). The system data bus transfers data and commands between the CPU and the Z80 SIO. D_0 is the least significant bit.

$\overline{\text{DCDA}}$, $\overline{\text{DCDB}}$. *Data Carrier Detect* (inputs, active Low). These pins function as receiver enables if the SIO is programmed for Auto Enables; otherwise they may be used as general-purpose input pins. Both pins are Schmitt-trigger buffered to accommodate slow-risetime signals. The SIO detects pulses on these pins and interrupts the CPU on both logic level transitions. Schmitt-trigger buffering does not guarantee a specific noise-level margin.

$\overline{\text{DTRA}}$, $\overline{\text{DTRB}}$. *Data Terminal Ready* (outputs, active Low). These outputs follow the state programmed into the Z80 SIO. They can also be programmed as general-purpose outputs.

In the Z80 SIO/1 bonding option, $\overline{\text{DTRB}}$ is omitted.

IEI. *Interrupt Enable In* (input, active High). This signal is used with IEO to form a priority daisy chain when there is more than one interrupt-driven device. A High on this line indicates that no other device of higher priority is being serviced by a CPU interrupt service routine.

IEO. *Interrupt Enable Out* (output, active High). IEO is High only if IEI is High and the CPU is not servicing an interrupt from this SIO. Thus, this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU interrupt service routine.

$\overline{\text{INT}}$. *Interrupt Request* (output, open drain, active Low). When the SIO is requesting an interrupt, it pulls $\overline{\text{INT}}$ Low.

$\overline{\text{IORQ}}$. *Input/Output Request* (input from CPU, active Low). $\overline{\text{IORQ}}$ is used in conjunction with B/$\overline{\text{A}}$, C/$\overline{\text{D}}$, $\overline{\text{CE}}$, and $\overline{\text{RD}}$ to transfer commands and data between the CPU and the SIO. When $\overline{\text{CE}}$, $\overline{\text{RD}}$, and $\overline{\text{IORQ}}$ are all active, the channel selected by B/$\overline{\text{A}}$ transfers data to the CPU (a read operation). When $\overline{\text{CE}}$ and $\overline{\text{IORQ}}$ are active, but $\overline{\text{RD}}$ is inactive, the channel selected by B/$\overline{\text{A}}$ is written to by the CPU with either data or control information as specified by C/$\overline{\text{D}}$. As mentioned previously, if $\overline{\text{IORQ}}$ and $\overline{\text{M1}}$ are active simultaneously, the CPU is acknowledging an interrupt and the SIO automatically places its interrupt vector on the CPU data bus if it is the highest priority device requesting an interrupt.

$\overline{\text{M1}}$. *Machine Cycle One*(input from Z80 CPU, active Low). When $\overline{\text{M1}}$ is active and $\overline{\text{RD}}$ is also active, the Z80 CPU is fetching an instruction from memory; when $\overline{\text{M1}}$ is active while $\overline{\text{IORQ}}$ is active, the SIO accepts $\overline{\text{M1}}$ and $\overline{\text{IORQ}}$ as an interrupt acknowledge if the SIO is the highest priority device that has interrupted the Z80 CPU.

$\overline{\text{RxCA}}$, $\overline{\text{RxCB}}$. *Receiver Clocks* (inputs). Receive data is sampled on the rising edge of $\overline{\text{RxC}}$. The Receive Clocks may be 1, 16, 32, or 64 times the data rate in asynchronous modes. These clocks may be driven by the Z80 CTC Counter Timer Circuit for programmable baud rate generation. Both inputs are Schmitt-trigger buffered; no noise level margin is specified.

In the Z80 SIO/0 bonding option, $\overline{\text{RxCB}}$ is bonded together with $\overline{\text{TxCB}}$.

$\overline{\text{RD}}$. *Read Cycle Status* (input from CPU, active Low). If $\overline{\text{RD}}$ is active, a memory or I/O read operation is in progress. $\overline{\text{RD}}$ is used with B/$\overline{\text{A}}$, $\overline{\text{CE}}$, and $\overline{\text{IORQ}}$ to transfer data from the SIO to the CPU.

RxDA, RxDB. *Receive Data* (inputs, active High). Serial data at TTL levels.

$\overline{\text{RESET}}$. *Reset* (input, active Low). A Low $\overline{\text{RESET}}$ disables both receivers and transmitters, forces TxDA and TxDB marking, forces the modem controls High, and disables all interrupts. The control registers must be rewritten after the SIO is reset and before data is transmitted or received.

$\overline{\text{RTSA}}$, $\overline{\text{RTSB}}$. *Request To Send* (outputs, active Low). When the RTS bit in Write Register 5 (Figure 14) is set, the $\overline{\text{RTS}}$ output goes Low. When the RTS bit is reset in the Asynchronous mode, the output goes High after the transmitter is empty. In Synchronous modes, the $\overline{\text{RTS}}$ pin strictly follows the state of the RTS bit. Both pins can be used as general-purpose outputs.

$\overline{\text{SYNCA}}$, $\overline{\text{SYNCB}}$. *Synchronization* (bidirectional, active Low). These pins can act either as inputs or outputs. In the asynchronous receive mode, they are inputs similar to $\overline{\text{CTS}}$ and $\overline{\text{DCD}}$. In this mode, the transitions on these lines affect the state of the Sync/Hunt status bits in Read Register 0 (Figure 13), but have no other function. In the External Sync mode, these lines also act as inputs. When external synchronization is achieved, $\overline{\text{SYNC}}$ must be driven Low on the second rising edge of $\overline{\text{RxC}}$ after that rising edge of $\overline{\text{RxC}}$ on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activate the $\overline{\text{SYNC}}$ input. Once $\overline{\text{SYNC}}$ is forced Low, it should be kept Low until the CPU informs the external synchronization detect logic that synchronization has been lost or a new message is about to start. Character assembly begins on the rising edge of $\overline{\text{RxC}}$ that immediately precedes the falling edge of $\overline{\text{SYNC}}$ in the External Sync mode.

In the internal synchronization mode (Monosync and Bisync), these pins act as outputs that are active during the part of the receive clock ($\overline{\text{RxC}}$) cycle in which sync characters are recognized. The sync condition is not latched, so these outputs are active each time a sync pattern is recognized, regardless of character boundaries.

In the Z80 SIO/2 bonding option, $\overline{\text{SYNCB}}$ is omitted.

$\overline{\text{TxCA}}$, $\overline{\text{TxCB}}$. *Transmitter Clocks* (inputs). In asynchronous modes, the Transmitter Clocks may be 1, 16, 32, or 64 times the data rate; however, the clock multiplier must be the same for the transmitter and the receiver. The Transmit Clock inputs are Schmitt-trigger buffered for relaxed rise- and fall-time requirements; no noise level margin is specified. Transmitter Clocks may be driven by the Z80 CTC Counter Timer Circuit for programmable baud rate generation.

In the Z80 SIO/0 bonding option, $\overline{\text{TxCB}}$ is bonded together with $\overline{\text{RxCB}}$.

TxDA, TxDB. *Transmit Data* (outputs, active High). Serial data at TTL levels. TxD changes from the falling edge of $\overline{\text{TxC}}$.

$\overline{\text{W/RDYA}}$, $\overline{\text{W/RDYB}}$. *Wait/Ready* (outputs, open drain when programmed for Wait function; driven High and Low when programmed for Ready function). These dual-purpose outputs may be programmed as Ready lines for a DMA controller or as Wait lines that synchronize the CPU to the SIO data rate. The reset state is open drain.

FUNCTIONAL DESCRIPTION

The functional capabilities of the Z80 SIO can be described from two different points of view: as a data communications device, it transmits and receives serial data in a wide variety of data-communication protocols; as a Z80 family peripheral, it interacts with the Z80 CPU and other peripheral circuits, sharing the data, address and control buses, as well as being a part of the Z80 interrupt structure. As a peripheral to other microprocessors, the SIO offers valuable features such as non-vectored interrupts, polling, and simple handshake capability. Figure 8 is a block diagram.

Figure 9 illustrates the conventional devices that the SIO replaces.

The first part of the following discussion covers SIO data-communication capabilities; the second part describes interactions between the CPU and the SIO.

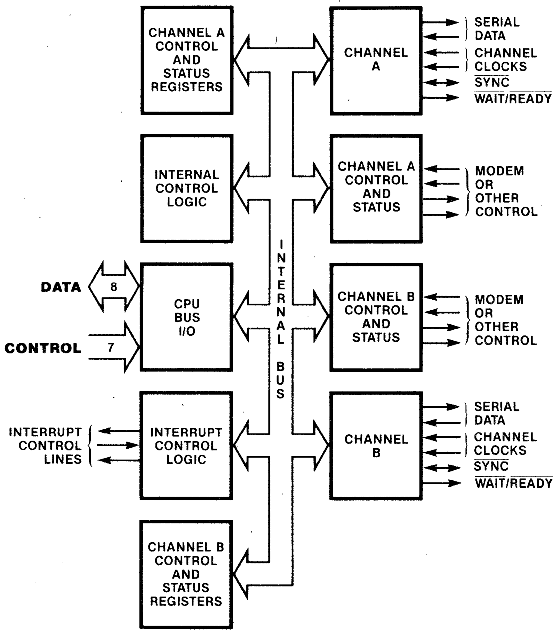


Figure 8. Block Diagram

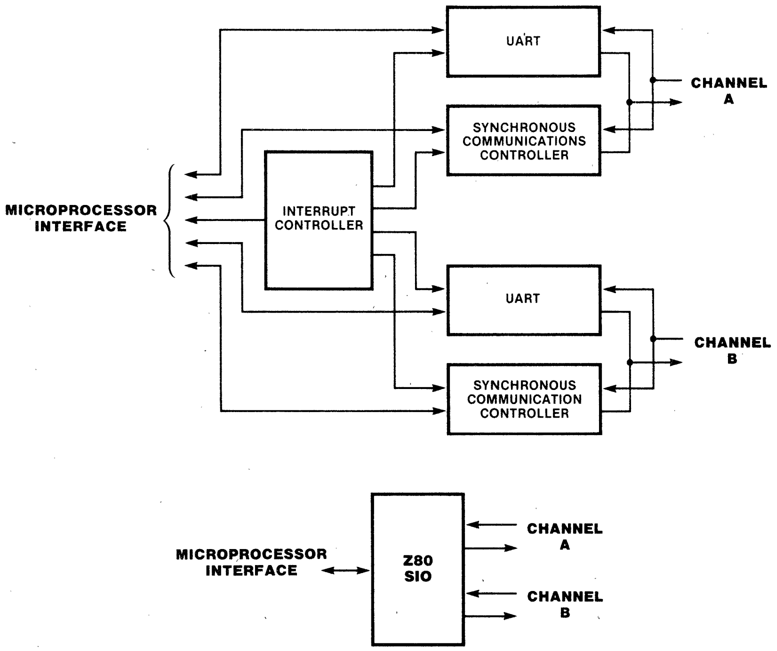


Figure 9. Conventional Devices Replaced by the Z80 SIO

DATA COMMUNICATION CAPABILITIES

The SIO provides two independent full-duplex channels that can be programmed for use in any common asynchronous, or synchronous data-communication protocol. Figure 10a illustrates some of these protocols. The following is a short description of them. A more detailed explanation of these modes can be found in the *Z80 SIO Technical Manual* (03-3033-01).

Asynchronous Modes. Transmission and reception can be done independently on each channel with five to eight bits per character, plus optional even or odd parity. The transmitters can supply one, one-and-a-half, or two stop bits per character and can provide a break output at any time. The receiver break-detection logic interrupts the CPU both at the start and end of a received break. Reception is protected from spikes by a transient spike-rejection mechanism that checks the signal one-half a bit time after a Low level is detected on the receive data input (RxDA or RxDB in Figure 5). If the Low does not persist, as in the case of a transient, the character assembly process is not started.

Framing errors and overrun errors are detected and buffered together with the partial character on which they occurred. Vectored interrupts allow fast servicing of error conditions using dedicated routines. Furthermore, a built-in checking process avoids interpreting a framing error as a new start bit: a framing error results in the addition of one-half a bit time to the point at which the search for the next start bit is begun.

The SIO does not require symmetric transmit and receive clock signals, a feature that allows it to be used with a Z80 CTC or many other clock sources. The transmitter and receiver can handle data at a rate of 1, 1/16, 1/32, or 1/64 of the clock rate supplied to the receive and transmit clock inputs.

In asynchronous modes, the $\overline{\text{SYNC}}$ pin may be programmed as an input that can be used for functions such as monitoring a ring indicator.

Synchronous Modes. The SIO supports both byte-oriented and bit-oriented synchronous communication.

Synchronous byte-oriented protocols can be handled in several modes that allow character synchronization with an 8-bit sync character (Monosync), any 16-bit sync pattern (Bisync), or with an external sync signal. Leading sync characters can be removed without interrupting the CPU.

Five-, six-, or seven-bit sync characters are detected with 8- or 16-bit patterns in the SIO by overlapping the larger pattern across multiple incoming sync characters, as shown in Figure 10b.

CRC checking for synchronous byte-oriented modes is delayed by one character time so the CPU may disable CRC checking on specific characters. This permits implementation of protocols such as IBM Bisync.

Figure 10a. Some Z80 SIO Protocols

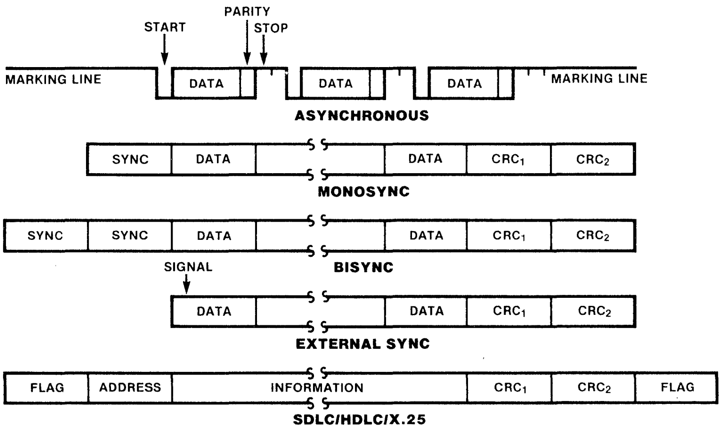


Figure 10b. Six-Bit Sync Character Recognition

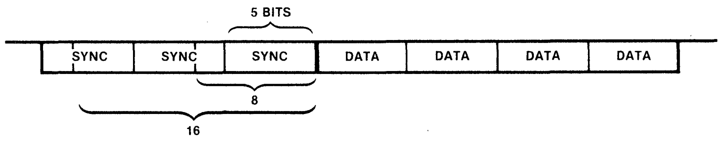


Figure 10. Data Communication

Both CRC-16 ($X^{16} + X^{15} + X^2 + 1$) and CCITT ($X^{16} + X^{12} + X^5 + 1$) error checking polynomials are supported. In all non-SDLC modes, the CRC generator is initialized to 0s; in SDLC modes, it is initialized to 1s. The SIO can be used for interfacing to peripherals such as hard-sectored floppy disks, but it cannot generate or check CRC for IBM-compatible soft-sectored disks. The SIO also provides a feature that automatically transmits CRC data when no other data is available for transmission. This allows very high-speed transmissions under DMA control with no need for CPU intervention at the end of a message. When there is no data or CRC to send in synchronous modes, the transmitter inserts 8- or 16-bit sync characters regardless of the programmed character length.

The SIO supports synchronous bit-oriented protocols such as SDLC and HDLC by performing automatic flag sending, zero insertion, and CRC generation. A special command can be used to abort a frame in transmission. At the end of a message the SIO automatically transmits the CRC and trailing flag when the transmit buffer becomes empty. If a transmit underrun occurs in the middle of a message, an external/status interrupt warns the CPU of this status change so that an abort may be issued. One to eight bits per character can be sent, which allows reception of a message with no prior information about the character structure in the information field of a frame.

The receiver automatically synchronizes on the leading flag of a frame in SDLC or HDLC, and provides a synchronization signal on the $\overline{\text{SYNC}}$ pin; an interrupt can also be programmed. The receiver can be programmed to search for frames addressed by a single byte to only a specified user-selected address or to a global broadcast address. In this mode, frames that do not match either the user-selected or broadcast address are ignored. The number of address bytes can be extended under software control. For transmitting data, an interrupt on the first received character or on every character can be selected. The receiver automatically deletes all zeroes inserted by the transmitter during character assembly. It also calculates and automatically checks the CRC to validate frame transmission. At the end of transmission, the status of a received frame is available in the status registers.

The SIO can be conveniently used under DMA control to provide high-speed reception or transmission. In reception, for example, the SIO can interrupt the CPU when the first character of a message is received. The CPU then enables the DMA to transfer the message to memory. The SIO then issues an end-of-frame interrupt and the CPU can check the status of the received message. Thus, the CPU is freed for other service while the message is being received.

I/O INTERFACE CAPABILITIES

The SIO offers the choice of polling, vectored or non-vectored interrupts and block-transfer modes to transfer data, status, and control information to, and from, the CPU. The block-transfer mode can also be implemented under DMA control.

Polling. Two status registers are updated at appropriate times for each function being performed (for example, CRC error-status valid at the end of a message). When the CPU is operated in a polling fashion, one of the SIO's two status registers is used to indicate whether the SIO has some data or needs some data. Depending on the contents of this register, the CPU will either write data, read data, or just go on. Two bits in the register indicate that a data transfer is needed. In addition, error and other conditions are indicated. The second status register (special receive conditions) does not have to be read in a polling sequence, until a character has been received. All interrupt modes are disabled when operating the device in a polled environment.

Interrupts. The SIO has an elaborate interrupt scheme to provide fast interrupt service in real-time applications. A control register and a status register in Channel B contain the interrupt vector. When programmed to do so, the SIO can modify three bits of the interrupt vector in the status register so that it points directly to one of eight interrupt service routines in memory, thereby servicing conditions in both channels and eliminating most of the needs for a status-analysis routine.

Transmit interrupts, receive interrupts, and external/status interrupts are the main sources of interrupts. Each interrupt source is enabled under program control, with Channel A having a higher priority than Channel B, and with receive, transmit, and external/status interrupts prioritized in that order within each channel. When the transmit interrupt is enabled, the CPU is interrupted by the transmit buffer becoming empty. (This implies that the transmitter must have had a data character written into it so it can become empty.) The receiver can interrupt the CPU in one of two ways:

- Interrupt on first received character
- Interrupt on all received characters

Interrupt-on-first-received-character is typically used with the block-transfer mode. Interrupt-on-all-received-characters has the option of modifying the interrupt vector in the event of a parity error. Both of these interrupt modes will also interrupt under special receive conditions on a character or message basis (end-of-frame interrupt in SDLC, for example). This means that the special-receive condition can cause an interrupt only if the interrupt-on-first-received-character or interrupt-on-all-received-characters mode is selected. In interrupt-on-first-received-character, an interrupt can occur from special-receive conditions (except parity error) after the first-received-character interrupt (example: receive-overrun interrupt).

The main function of the external/status interrupt is to monitor the signal transitions of the Clear To Send ($\overline{\text{CTS}}$), Data Carrier Detect ($\overline{\text{DCD}}$), and Synchronization ($\overline{\text{SYNC}}$) pins (Figures 1 through 7). In addition, an external/status interrupt is also caused by a CRC-sending condition, or by the detection of a break sequence (asynchronous mode) or abort sequence (SDLC mode) in the data stream. The interrupt caused by the break/abort sequence allows the SIO to interrupt when the break/abort sequence is detected or terminated. This feature facilitates the proper termination of the current message, correct initialization of the next message, and the accurate timing of the break/abort condition in external logic.

In a Z80 CPU environment (Figure 11), SIO interrupt vectoring is "automatic": the SIO passes its internally-modifiable 8-bit interrupt vector to the CPU, which adds an additional 8 bits from its interrupt-vector (I) register to form the memory address of the interrupt-routine table. This table contains the address of the beginning of the interrupt routine itself. The process entails an indirect transfer of CPU control to the interrupt routine, so that the next instruction executed after an interrupt acknowledge by the CPU is the first instruction of the interrupt routine itself.

CPU/DMA Block Transfer. The SIO's block-transfer mode accommodates both CPU block transfers and DMA controllers (Z80 DMA or other designs). The block-transfer mode uses the Wait/Ready output signal, which is selected with three bits in an internal control register. The Wait/Ready output signal can be programmed as a $\overline{\text{WAIT}}$ line in the CPU block-transfer mode or as a $\overline{\text{READY}}$ line in the DMA block-transfer mode.

To a DMA controller, the SIO $\overline{\text{READY}}$ output indicates that the SIO is ready to transfer data to, or from, memory. To the CPU, the $\overline{\text{WAIT}}$ output indicates that the SIO is not ready to transfer data, thereby requesting the CPU to extend the I/O cycle.

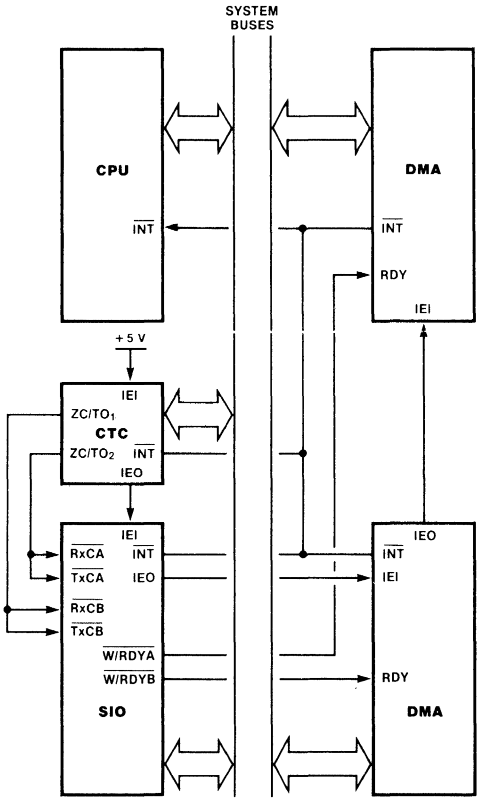


Figure 11. Typical Z80 Environment

INTERNAL STRUCTURE

The internal structure of the device includes a Z80 CPU interface, internal control and interrupt logic, and two full-duplex channels. Each channel contains its own set of control and status (write and read) registers, and control and status logic that provides the interface to modems or other external devices.

The registers for each channel are designated as follows:

WR0-WR7 — Write Registers 0 through 7
RR0-RR2 — Read Registers 0 through 2

The register group includes five 8-bit control registers, two sync-character registers and two status registers. The interrupt vector is written into an additional 8-bit register (Write Register 2) in Channel B that may be read through another 8-bit register (Read Register 2) in Channel B. The bit assignment and functional grouping of each register is configured to simplify and organize the programming process. Table 1 lists the functions assigned to each read or write register.

The logic for both channels provides formats, synchronization, and validation for data transferred to and from the channel interface. The modem control inputs, Clear To Send ($\overline{\text{CTS}}$) and Data Carrier Detect ($\overline{\text{DCD}}$), are

Table 1. Register Functions

Read Register Functions	
RR0	Transmit/Receive buffer status, interrupt status and external status
RR1	Special Receive Condition status
RR2	Modified interrupt vector (Channel B only)

Write Register Functions	
WR0	Register pointers, CRC initialize, and initialization commands for the various modes.
WR1	Transmit/Receive interrupt and data transfer mode definition.
WR2	Interrupt vector (Channel B only)
WR3	Receive parameters and control
WR4	Transmit/Receive miscellaneous parameters and modes
WR5	Transmit parameters and controls
WR6	Sync character or SDLC address field
WR7	Sync character or SDLC flag

monitored by the external control and status logic under program control. All external control-and-status-logic signals are general-purpose in nature and can be used for functions other than modem control.

Data Path. The transmit and receive data path illustrated for Channel A in Figure 12 is identical for both channels. The receiver has three 8-bit buffer registers in a FIFO arrangement, in addition to the 8-bit receive shift register. This scheme creates additional time for the CPU to service an interrupt at the beginning of a block of high-speed data. Incoming data is routed through one of several paths (data or CRC) depending on the selected mode and—in asynchronous modes—the character length.

The transmitter has an 8-bit transmit data buffer register that is loaded from the internal data bus, and a 20-bit transmit shift register that can be loaded from the sync-character buffers or from the transmit data register. Depending on the operational mode, outgoing data is routed through one of four main paths before it is transmitted from the Transmit Data output (TxD).

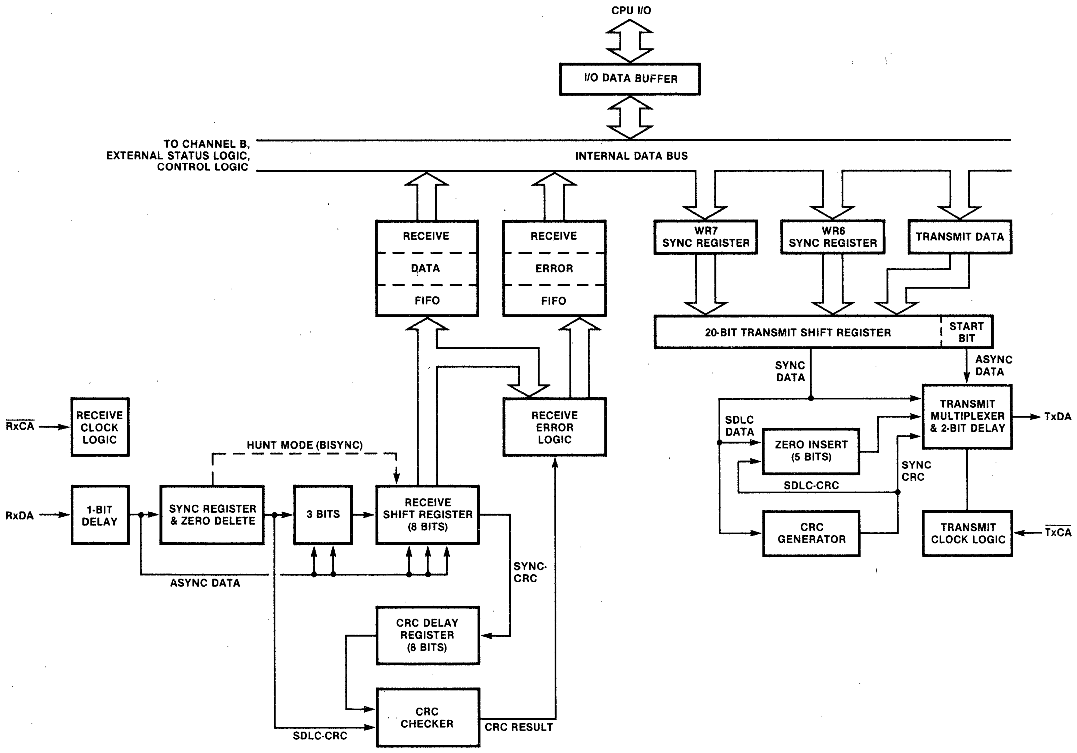


Figure 12. Transmit and Receive Data Path (Channel A)

PROGRAMMING

The system program first issues a series of commands that initialize the basic mode of operation and then issues other commands that qualify conditions within the selected mode. For example, the asynchronous mode, character length, clock rate, number of stop bits, even or odd parity might be set first; then the interrupt mode; and finally, receiver or transmitter enable.

Both channels contain registers that must be programmed via the system program prior to operation. The channel-select input (B/$\overline{A}$) and the control/data (C/$\overline{D}$) are the command-structure addressing controls, and are normally controlled by the CPU address bus. Figures 15 and 16 illustrate the timing relationships for programming the write registers and transferring data and status.

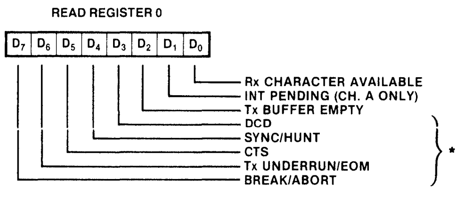
Read Registers. The SIO contains three read registers for Channel B and two read registers for Channel A (RR0-RR2 in Figure 13) that can be read to obtain the status information; RR2 contains the internally-modifiable interrupt vector and is only in the Channel B register set. The status information includes error conditions, interrupt vector, and standard communications-interface signals.

To read the contents of a selected read register other than RR0, the system program must first write the pointer byte to WR0 in exactly the same way as a write register operation. Then, by executing a read instruction, the contents of the addressed read register can be read by the CPU.

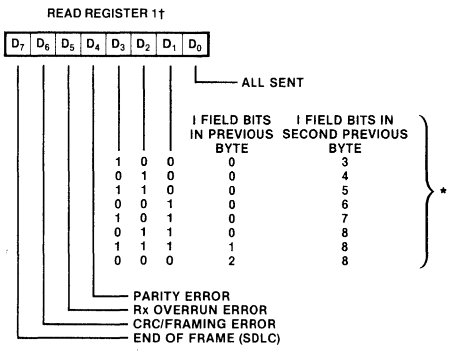
The status bits of RR0 and RR1 are carefully grouped to simplify status monitoring. For example, when the interrupt vector indicates that a Special Receive Condition interrupt has occurred, all the appropriate error bits can be read from a single register (RR1).

Write Registers. The SIO contains eight write registers for Channel B and seven write registers for Channel A (WR0-WR7 in Figure 14) that are programmed separately to configure the functional personality of the channels; WR2 contains the interrupt vector for both channels and is only in the Channel B register set. With the exception of WR0, programming the write registers requires two bytes. The first byte is to WR0 and contains three bits (D_0-D_2) that point to the selected register; the second byte is the actual control word that is written into the register to configure the SIO.

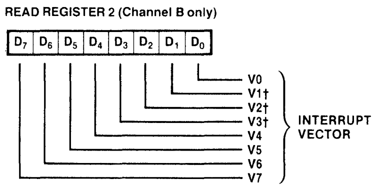
WR0 is a special case in that all of the basic commands can be written to it with a single byte. Reset (internal or external) initializes the pointer bits D_0-D_2 to point to WR0. This implies that a channel reset must not be combined with the pointing to any register.



*Used With "External/Status Interrupt" Modes



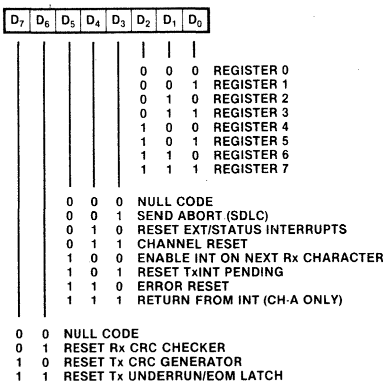
*Residue data for eight Rx bits/character programmed
†Used with special receive condition mode



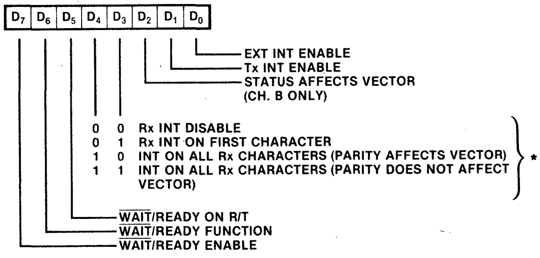
†Variable if "Status Affects Vector" is programmed

Figure 13. Read Register Bit Functions

WRITE REGISTER 0

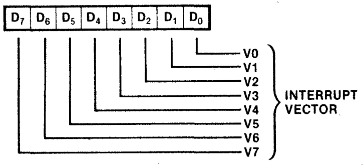


WRITE REGISTER 1

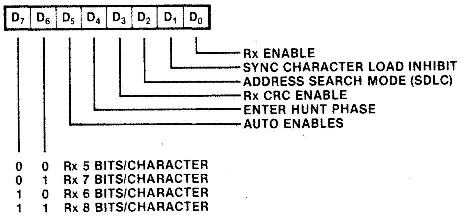

D7 D6 D5 D4 D3 D2 D1 D0
EXT INT ENABLE
Tx INT ENABLE
STATUS AFFECTS VECTOR (CH. B ONLY)
0 0 Rx INT DISABLE
0 1 Rx INT ON FIRST CHARACTER
1 0 INT ON ALL Rx CHARACTERS (PARITY AFFECTS VECTOR)
1 1 INT ON ALL Rx CHARACTERS (PARITY DOES NOT AFFECT VECTOR)
*
WAIT/READY ON R/T
WAIT/READY FUNCTION
WAIT/READY ENABLE

* Or on special condition

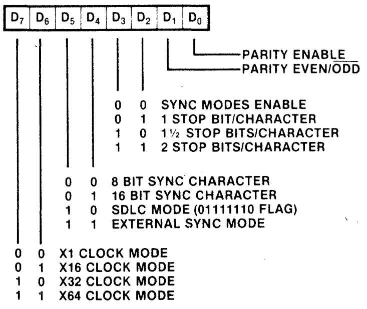
WRITE REGISTER 2 (Channel B only)



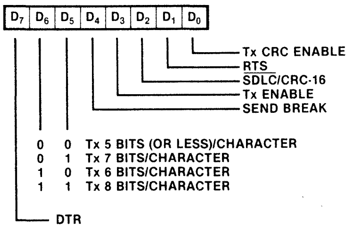
WRITE REGISTER 3



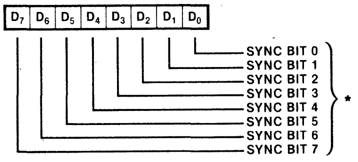
WRITE REGISTER 4



WRITE REGISTER 5

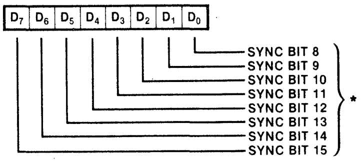


WRITE REGISTER 6



*Also SDLC address field

WRITE REGISTER 7



*For SDLC it must be programmed to "01111110" for flag recognition

Figure 14. Write Register Bit Functions

TIMING

The SIO must have the same clock as the CPU (same phase and frequency relationship, not necessarily the same driver).

Read Cycle. The timing signals generated by a Z80 CPU input instruction to read a data or status byte from the SIO are illustrated in Figure 15.

Write Cycle. Figure 16 illustrates the timing and data signals generated by a Z80 CPU output instruction to write a data or control byte into the SIO.

Interrupt-Acknowledge Cycle. After receiving an interrupt-request signal from an SIO ($\overline{\text{INT}}$ pulled Low), the Z80 CPU sends an interrupt-acknowledge sequence, $\overline{\text{M1}}$ Low and $\overline{\text{IORQ}}$ Low, a few cycles later (Figure 17).

The SIO contains an internal daisy-chained interrupt structure for prioritizing nested interrupts for the various functions of its two channels, and this structure can be used within an external user-defined daisy chain that prioritizes several peripheral circuits.

The IEI of the highest-priority device is terminated High. A device that has an interrupt pending or under service forces its IEO Low. For devices with no interrupt pending or under service, IEO = IEI.

To insure stable conditions in the daisy chain, all interrupt status signals are prevented from changing while $\overline{\text{M1}}$ is Low. When $\overline{\text{IORQ}}$ is Low, the highest priority interrupt requestor (the one with IEI High) places its interrupt vector on the data bus and sets its internal interrupt-under-service latch.

Return From Interrupt Cycle. Figure 18 illustrates the return from interrupt cycle. Normally, the Z80 CPU issues a Return From Interrupt (RETI) instruction at the end of an interrupt service routine. RETI is a 2-byte opcode (ED-4D) that resets the interrupt-under-service latch in the SIO to terminate the interrupt that has just been processed. This is accomplished by manipulating the daisy chain in the following way.

The normal daisy-chain operation can be used to detect a pending interrupt; however, it cannot distinguish between an interrupt under service and a pending unacknowledged interrupt of a higher priority. Whenever ED is decoded, the daisy chain is modified by forcing High the IEO of any interrupt that has not yet been acknowledged. Thus the daisy chain identifies the device presently under service as the only one with an IEI High and an IEO Low. If the next opcode byte is 4D, the interrupt-under-service latch is reset.

The ripple time of the interrupt daisy chain (both the High-to-Low and the Low-to-High transitions) limits the number of devices that can be placed in the daisy chain. Ripple time can be improved with carry-look-ahead, or by extending the interrupt-acknowledge cycle. For further information about techniques for increasing the number of daisy-chained devices, refer to the *Z8400 Z80 CPU Product Specification* (00-2001-04).

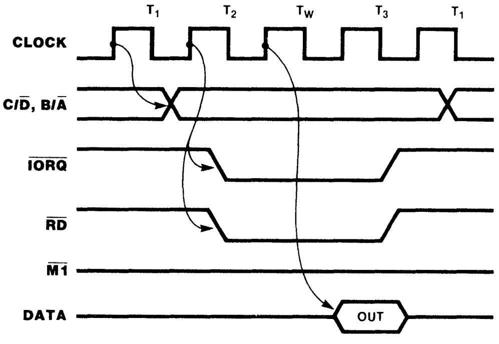


Figure 15. Read Cycle

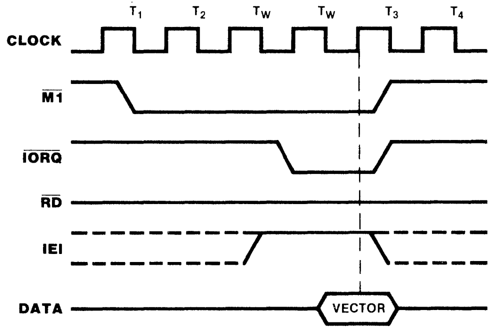


Figure 17. Interrupt Acknowledge Cycle

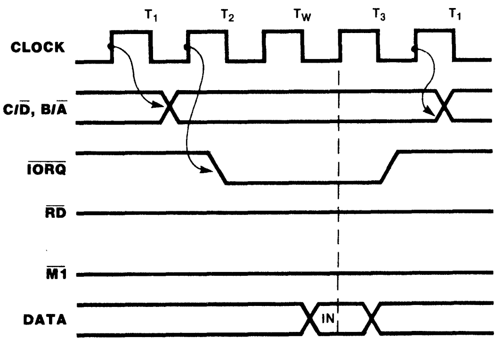


Figure 16. Write Cycle

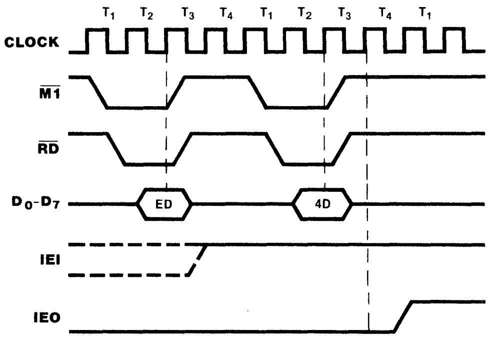


Figure 18. Return from Interrupt Cycle

ABSOLUTE MAXIMUM RATINGS

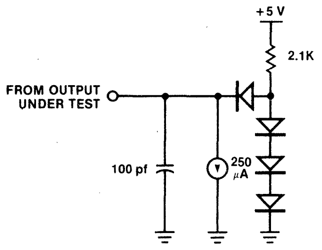
Voltages in V_{CC} with respect to V_{SS}	$-0.3V$ to $+0.7V$
Voltages on all inputs with respect to V_{SS}	$-0.3V$ to $V_{CC}+0.3V$
Storage Temperature	$-65°C$ to $+150°C$

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above these indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The characteristics below apply for the following test conditions, unless otherwise noted. All voltages are referenced to GND (0V). Positive current flows into the referenced pin. Available operating temperature range is:

- **S = 0°C to +70°C, V_{cc} Range**
  NMOS: +4.75V < V_{cc} < +5.25V
  CMOS: +4.50V < V_{cc} < +5.50V
- **E = -40°C to 100°C, =4.50V < V_{cc} < +5.50V**



DC CHARACTERISTICS (Z84C4X CMOS Z80 SIO)

Symbol	Parameter	Min	Max	Typ	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	−0.3	+0.45		V	
V_{IHC}	Clock Input High Voltage	$V_{CC}-0.6$	$V_{CC}+0.3$		V	
V_{IL}	Input Low Voltage	−0.3	+0.8		V	
V_{IH}	Input High Voltage	+2.2	V_{CC}		V	
V_{OL}	Output Low Voltage		+0.4		V	I_{OL} = 2.0 mA
V_{OH1}	Output High Voltage	+2.4			V	I_{OH} = −1.6 mA
V_{OH2}	Output High Voltage	$V_{CC}-0.8$			V	I_{OH} = −250 μA
I_{LI}	Input Leakage Current		±10		μA	V_{IN} = 0.4 to V_{CC}
I_{LO}	3-State Output Leakage Current in Float		±10		μA	V_{OUT} = 0.4 to V_{CC}
$I_{L(SY)}$	SYNC Pin Leakage Current		+10/−40		μA	V_{IN} = 0.4 to V_{CC}
ICC_1	Power Supply Current					V_{CC} = 5V
			10	7	mA	CLK = 4MHz
			10	**7**	**mA**	**CLK = 6MHz**
			12	**8**	**mA**	**CLK = 8MHz**
						$V_{IH} = V_{CC} - 0.2V$
						V_{IL} = 0.2V
ICC_2	Standby Supply Current		10	0.5	μA	V_{CC} = 5V
						CLK = (0)
						$V_{IH} = V_{CC} - 0.2V$
						V_{IL} = 0.2V

Over specified temperature and voltage range.

CAPACITANCE

Symbol	Parameter	Min	Max	Unit
C	Clock Capacitance		7	pf
C_{IN}	Input Capacitance		5	pf
C_{OUT}	Output Capacitance		10	pf

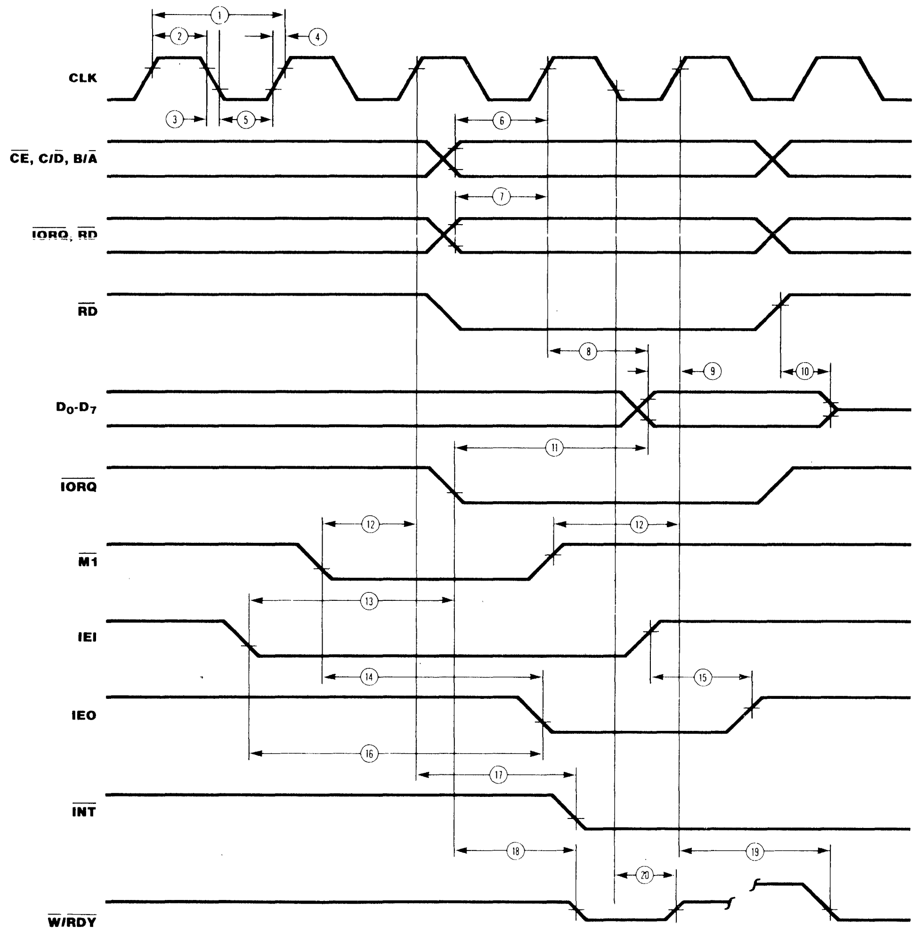
Over specified temperature range; f = 1 MHz.
Unmeasured pins returned to ground.

AC CHARACTERISTICS* (Z84C4X CMOS Z80 SIO)

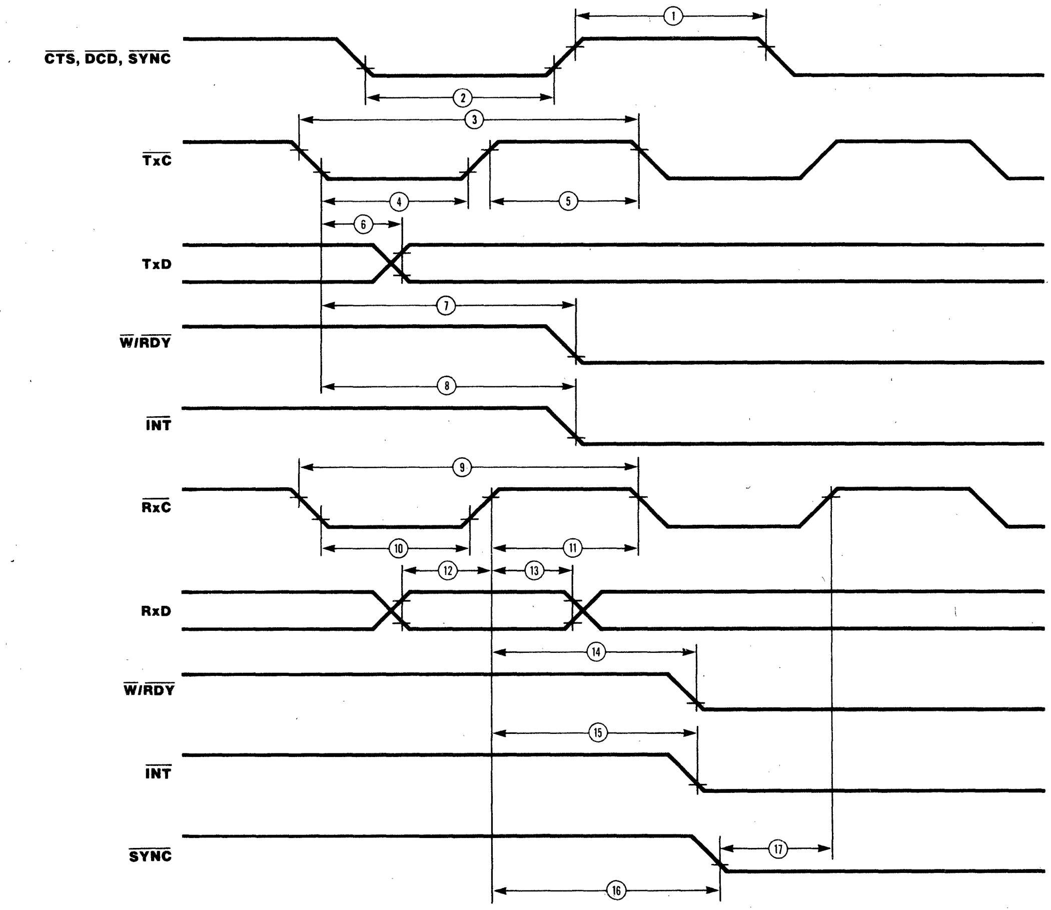
			Z84C40/1/2/4						
			Z84C4X04		Z84C4X06		Z84C4X08		
No.	Symbol	Parameter	Min	Max	Min	Max	Min	Max	Notes
1	TcC	Clock Cycle Time	250	DC	162	DC	125	DC	
2	TwCh	Clock Width (High)	105	DC	65	DC	55	DC	
3	TfC	Clock Fall Time		30		20		10	
4	TrC	Clock Rise Time		30		20		10	
5	TwCl	Clock Width (Low)	105	DC	65	DC	55	DC	
6	TsAD(C)	$\overline{CE}$, C/$\overline{D}$, B/$\overline{A}$ to Clock ↑ Setup Time	145		60		40		
7	TsCS(C)	$\overline{IORQ}$, $\overline{RD}$ to Clock ↑ Setup Time	115		60		40		
8	TdC(DO)	Clock ↑ to Data Out Delay		220		150		100	
9	TsDI(C)	Data In to Clock ↑ Setup (write or $\overline{M1}$ Cycle)	50		30		20		
10	TdRD(DOz)	$\overline{RD}$ ↑ to Data Out Float Delay		110		90		75	
11	TdIO(DOI)	$\overline{IORQ}$ ↓ to Data Out Delay (INTACK Cycle)		160		120		90	
12	TsM1(C)	$\overline{M1}$ to Clock ↑ Setup Time	90		75		55		
13	TsIEI(IO)	IEI to $\overline{IORQ}$ ↓ Setup Time (INTACK Cycle)	140		120		80		
14	TdM1(IEO)	$\overline{M1}$ ↓ to IEO ↓ Delay (interrupt before $\overline{M1}$)		190		160		130	
15	TdIEI(IEOr)	IEI ↑ to IEO ↑ Delay (after ED decode)		100		70		60	
16	TdIEI(IEOf)	IEI ↓ to IEO ↓ Delay		100		70		60	
17	TdC(INT)	Clock ↑ to $\overline{INT}$ ↓ Delay		200		150		120	
18	TdIO(W/RWf)	$\overline{IORQ}$ ↓ or $\overline{CE}$ ↓ to $\overline{W}$/$\overline{RDY}$ Delay (Wait Mode)		210		175		130	
19	TdC(W/RR)	Clock ↑ to $\overline{W}$/$\overline{RDY}$ Delay (Ready Mode)		120		100		90	
20	TdC(W/RWz)	Clock ↓ to $\overline{W}$/$\overline{RDY}$ Float Delay (Wait Mode)		130		110		90	
21	Th	Any unspecified Hold when Setup is specified	0		0		0		

* Units in nanoseconds (ns).

AC CHARACTERISTICS TIMING (Z84C4X CMOS Z80 SIO)



AC CHARACTERISTICS TIMING (Z84C4X CMOS Z80 SIO; Continued)



AC CHARACTERISTICS (Z84C4X CMOS Z80 SIO; Continued)

			Z84C40/1/2/4						
			Z84C4X04		Z84C4X06		Z84C4X08		
No.	Symbol	Parameter	Min	Max	Min	Max	Min	Max	Notes
1	TwPh	Pulse Width (High)	200		200		150		2
2	TwPl	Pulse Width (Low)	200		200		150		2
3	TcTxC	$\overline{\text{TxC}}$ Cycle Time	400		330		250		2
4	TwTxCl	$\overline{\text{TxC}}$ Width (Low)	180		100		85		2
5	TwTxCh	$\overline{\text{TxC}}$ Width (High)	180		100		85		2
6	TdTxC(TxD)	$\overline{\text{TxC}}$ ↓ to TxD Delay		300		220		160	2
7	TdTxC(W/RRf)	$\overline{\text{TxC}}$ ↓ to $\overline{\text{W/RDY}}$ ↓ Delay (Ready Mode)		5-9		5-9		5-9	1
8	TdTxC(INT)	$\overline{\text{TxC}}$ ↓ to $\overline{\text{INT}}$ ↓ Delay		5-9		5-9		5-9	1
9	TcRxC	$\overline{\text{RxC}}$ Cycle Time	400		330		250		2
10	TwRxCl	$\overline{\text{RxC}}$ Width (Low)	180		100		85		2
11	TwRxCh	$\overline{\text{RxC}}$ Width (High)	180		100		85		2
12	TsRxD(RxC)	RxD to $\overline{\text{RxC}}$ ↑ Setup Time (x1 Mode)	0		0		0		2
13	ThRxD(RxC)	$\overline{\text{RxC}}$ ↑ RxD Hold Time (x1 Mode)	140		100		80		2
14	TdRxC(W/RRf)	$\overline{\text{RxC}}$ ↑ to $\overline{\text{W/RDY}}$ ↓ Delay (Ready Mode)		10-13		10-13		10-13	1
15	TdRxC(INT)	$\overline{\text{RxC}}$ ↑ to $\overline{\text{INT}}$ ↓ Delay		10-13		10-13		10-13	1
16	TdRxC(SYNC)	$\overline{\text{RxC}}$ ↑ to $\overline{\text{SYNC}}$ ↓ Delay (Output Modes)		4-7		4-7		4-7	1
17	TsSYNC(RxC)	$\overline{\text{SYNC}}$ ↓ to $\overline{\text{RxC}}$ ↑ Setup (External Sync Modes)	-100		-100		-100		2

*In all modes, the System Clock rate must be at least five times the maximum data rate. RESET must be active a minimum of one complete clock cycle.

1. Units equal to System Clock Periods.

2. Units in nanoseconds (ns).

DC CHARACTERISTICS (Z844X / NMOS Z80 SIO)

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3	$+0.45$	V	
V_{IHC}	Clock Input High Voltage	$V_{CC}-0.6$	$V_{CC}+0.3$	V	
V_{IL}	Input Low Voltage	-0.3	$+0.8$	V	
V_{IH}	Input High Voltage	$+2.0$	V_{CC}	V	
V_{OL}	Output Low Voltage		$+0.4$	V	
V_{OH1}	Output High Voltage	$+2.4$		V	$I_{OL} = 2.0$ mA
V_{OH2}	Output High Voltage			V	$I_{OH} = -250\ \mu A$
I_{LI}	Input Leakage Current		± 10	μA	$V_{IN} = 0.4$ to V_{CC}
I_{LO}	3-State Output Leakage Current in Float		± 10	μA	$V_{OUT} = 0.4$ to V_{CC}
$I_{L(SY)}$	$\overline{\text{SYNC}}$ Pin Leakage Current		$+10/-40$	μA	$0 < V_{IN} < V_{CC}$
ICC_1	Power Supply Current		100	mA	

Over specified temperature and voltage range.

CAPACITANCE

Symbol	Parameter	Min	Max	Unit
C	Clock Capacitance		40	pf
C_{IN}	Input Capacitance		5	pf
C_{OUT}	Output Capacitance		15	pf

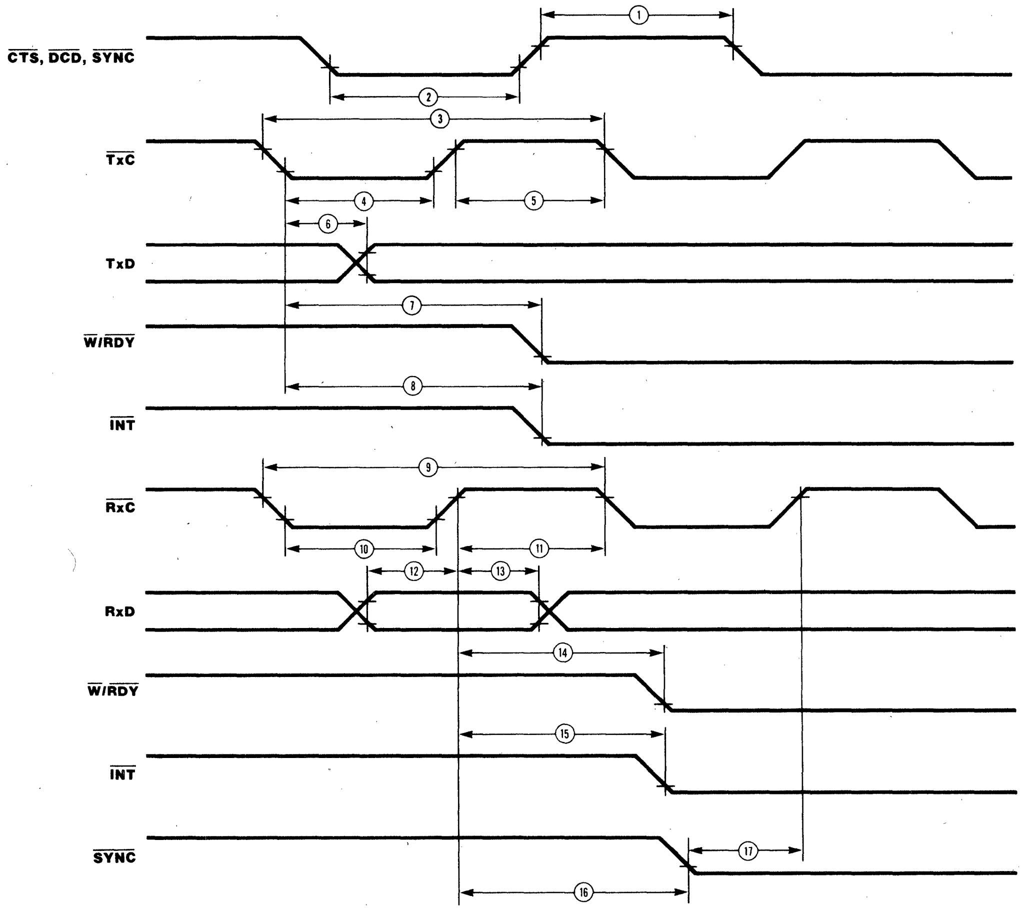
Over specified temperature range; f = 1 MHz.
Unmeasured pins returned to ground.

AC CHARACTERISTICS* (Z844X / NMOS Z80 SIO)

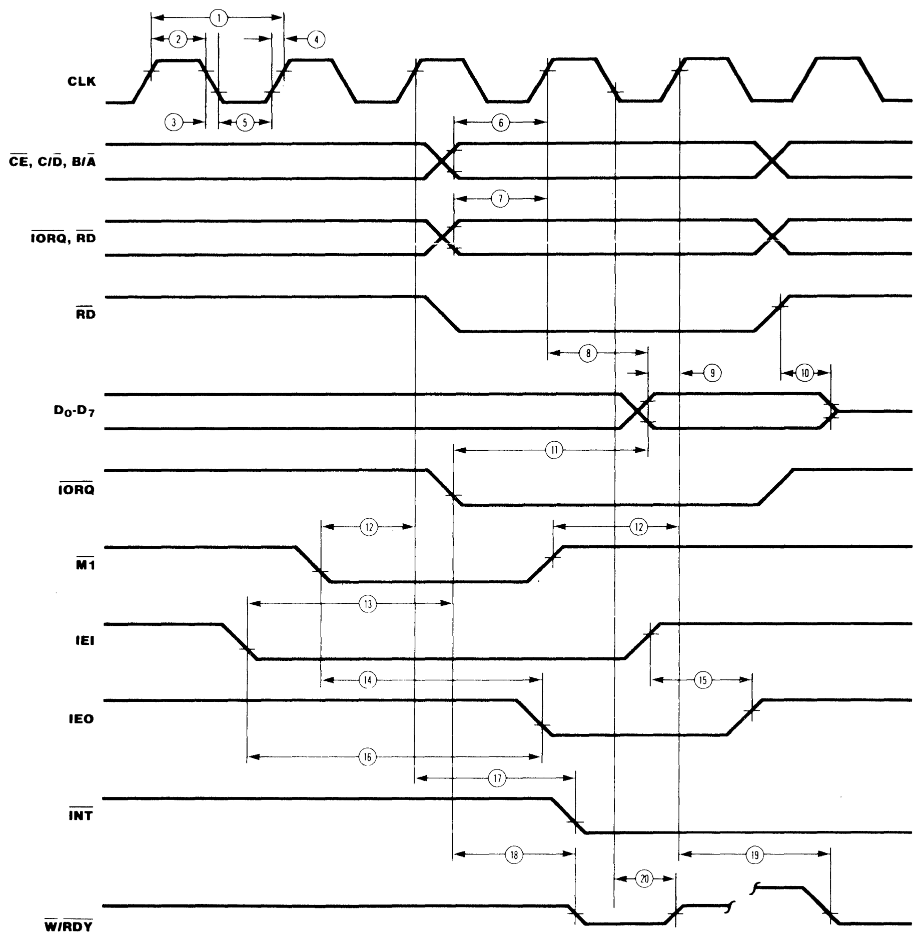
			Z0844X04		Z0844X06	
Number	**Symbol**	**Parameter**	**Min**	**Max**	**Min**	**Max**
1	TcC	Clock Cycle Time	250	4000	162	4000
2	TwCh	Clock Width (High)	105	2000	70	2000
3	TfC	Clock Fall Time		30		15
4	TrC	Clock Rise Time		30		15
5	TwCl	Clock Width (Low)	105	2000	70	2000
6	TsAD(C)	$\overline{\text{CE}}$, C/$\overline{\text{D}}$, B/$\overline{\text{A}}$ to Clock ↑ Setup Time	145		60	
7	TsCS(C)	$\overline{\text{IORQ}}$, $\overline{\text{RD}}$ to Clock ↑ Setup Time	115		60	
8	TdC(DO)	Clock ↑ to Data Out Delay		220		150
9	TsDI(C)	Data In to Clock ↑ Setup (Write or $\overline{\text{M1}}$ Cycle)	50		30	
10	TdRD(DOz)	$\overline{\text{RD}}$ ↑ to Data Out Float Delay		110		90
11	TdIO(DOI)	$\overline{\text{IORQ}}$ ↓ to Data Out Delay (INTACK Cycle)		160		120
12	TsM1(C)	$\overline{\text{M1}}$ to Clock ↑ Setup Time	90		75	
13	TsIEI(IO)	IEI to $\overline{\text{IORQ}}$ ↓ Setup Time (INTACK Cycle)	140		120	
14	TdM1(IEO)	$\overline{\text{M1}}$ ↓ to IEO ↓ Delay (interrupt before $\overline{\text{M1}}$)		190		160
15	TdIEI(IEOr)	IEI ↑ to IEO ↑ Delay (after ED decode)		100		70
16	TdIEI(IEOf)	IEI ↓ to IEO ↓ Delay		100		70
17	TdC(INT)	Clock ↑ to $\overline{\text{INT}}$ ↓ Delay		200		150
18	TdIO(W/RWf)	$\overline{\text{IORQ}}$ ↓ or $\overline{\text{CE}}$ ↓ to $\overline{\text{W/RDY}}$ ↓ Delay (Wait Mode)		210		175
19	TdC(W/RRf)	Clock ↑ to $\overline{\text{W/RDY}}$ ↓ Delay (Ready Mode)		120		100
20	TdC(W/RWz)	Clock ↓ to $\overline{\text{W/RDY}}$ Float Delay (Wait Mode)		130		110
21	Th	Any unspecified Hold when Setup is specified	0		0	

*Units in nanoseconds (ns).

AC CHARACTERISTICS TIMING (Z844X / NMOS Z80 SIO; Continued)



AC CHARACTERISTICS TIMING (Z844X / NMOS Z80 SIO)



AC CHARACTERISTICS (Z844X / NMOS Z80 SIO; Continued)

No.	Symbol	Parameter	Z0844X04 Min	Z0844X04 Max	Z0844X06 Min	Z0844X06 Max	Notes*
1	TwPh	Pulse Width (High)	200		200		2
2	TwPl	Pulse Width (Low)	200		200		2
3	TcTxC	$\overline{\text{TxC}}$ Cycle Time	400	∞	330	∞	2
4	TwTxCl	$\overline{\text{TxC}}$ Width (Low)	180	∞	100	∞	2
5	TwTxCh	$\overline{\text{TxC}}$ Width (High)	180	∞	100	∞	2
6	TdTxC(TxD)	$\overline{\text{TxC}}$ ↓ to TxD Delay		300		220	2
7	TdTxC(W/RRf)	$\overline{\text{TxC}}$ ↓ to $\text{W}/\overline{\text{RDY}}$ ↓ Delay (Ready Mode)	5	9	5	9	1
8	TdTxC(INT)	$\overline{\text{TxC}}$ ↓ to $\overline{\text{INT}}$ ↓ Delay	5	9	5	9	1
9	TcRxC	$\overline{\text{RxC}}$ Cycle Time	400	∞	330	∞	2
10	TwRxCl	$\overline{\text{RxC}}$ Width (Low)	180	∞	100	∞	2
11	TwRxCh	$\overline{\text{RxC}}$ Width (High)	180	∞	100	∞	2
12	TsRxD(RxC)	RxD to $\overline{\text{RxC}}$ ↑ Setup Time (x1 Mode)	0		0		2
13	ThRxD(RxC)	$\overline{\text{RxC}}$ ↑ RxD Hold Time (x1 Mode)	140		100		2
14	TdRxC(W/RRf)	$\overline{\text{RxC}}$ ↑ to $\text{W}/\overline{\text{RDY}}$ ↓ Delay (Ready Mode)	10	13	10	13	1
15	TdRxC(INT)	$\overline{\text{RxC}}$ ↑ to $\overline{\text{INT}}$ ↓ Delay	10	13	10	13	1
16	TdRxC(SYNC)	$\overline{\text{RxC}}$ ↑ to $\overline{\text{SYNC}}$ ↓ Delay (Output Modes)	4	7	4	7	1
17	TsSYNC(RxC)	$\overline{\text{SYNC}}$ ↓ to $\overline{\text{RxC}}$ ↑ Setup (External Sync Modes)	-100		-100		2

*In all modes, the System Clock rate must be at least five times the maximum data rate. $\overline{\text{RESET}}$ must be active a minimum of one complete clock cycle.

1. Units equal to System Clock Periods.

2. Units in nanoseconds (ns).

Z8470 Z80® DART Dual Asynchronous Receiver/Transmitter

Zilog

Product Specification

January 1989

FEATURES

- Two independent full-duplex channels with separate modem controls. Modem status can be monitored.
- **In x1 clock mode, data rates are 0 to 800K bits/second with a 4.0 MHz clock, or 0 to 1.2 M bits/second with a 6.0 MHz clock.**
- Receiver data registers are quadruply buffered; the transmitter is doubly buffered.
- Programmable options include 1, 1½, or 2 stop bits; even, odd, or no parity; and x1, x16, x32, and x64 clock modes.
- Break generation and detection as well as parity-, overrun-, and framing-error detection are available.
- Interrupt features include a programmable interrupt vector, a "status affects vector" mode for fast interrupt processing, and the standard Z80 peripheral daisy-chain interrupt structure that provides automatic interrupt vectoring with no external logic.
- On-chip logic for ring indication and carrier-detect status.

GENERAL DESCRIPTION

The Z80 DART (Dual-Channel Asynchronous Receiver/Transmitter) is a dual-channel multifunction peripheral component that satisfies a wide variety of asynchronous serial data communications requirements in microcomputer systems. The Z80 DART is used as a serial-to-parallel, parallel-to-serial converter/controller in asynchronous applications. In addition, the device also provides modem controls for both channels. In applications where modem controls are not needed, these lines can be used for general-purpose I/O.

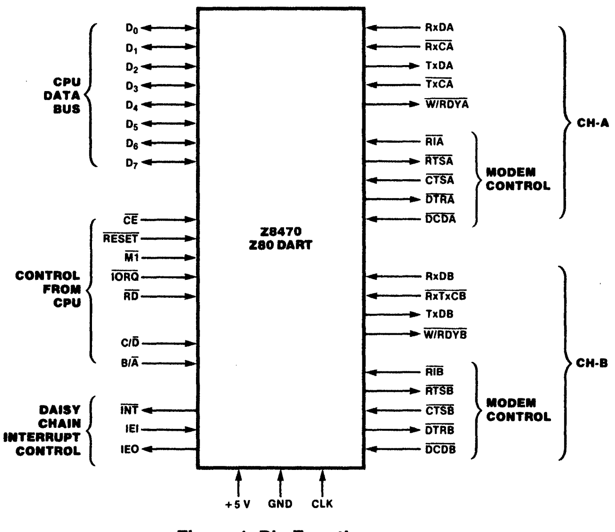

CPU DATA BUS
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
CONTROL FROM CPU
$\overline{CE}$ $\overline{RESET}$ $\overline{M1}$ $\overline{IORQ}$ $\overline{RD}$ $C/\overline{D}$ $B/\overline{A}$
DAISY CHAIN INTERRUPT CONTROL
$\overline{INT}$ IEI IEO
Z8470 Z80 DART
RxDA $\overline{RxCA}$ TxDA $\overline{TxCA}$ $\overline{W/RDYA}$
$\overline{RIA}$ $\overline{RTSA}$ $\overline{CTSA}$ $\overline{DTRA}$ $\overline{DCDA}$
MODEM CONTROL
CH-A
RxDB $\overline{RxTxCB}$ TxDB $\overline{W/RDYB}$
$\overline{RIB}$ $\overline{RTSB}$ $\overline{CTSB}$ $\overline{DTRB}$ $\overline{DCDB}$
MODEM CONTROL
CH-B
+5 V GND CLK

Figure 1. Pin Functions

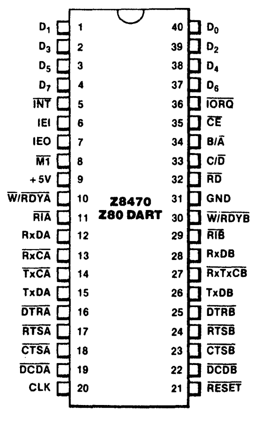


Figure 2. 40-Pin Dual-In-Line Package (DIP), Pin Assignments

Zilog also offers the Z80 SIO, a more versatile device that provides synchronous (Bisync, HDLC, and SDLC) as well as asynchronous operation.

The Z80 DART is fabricated with n-channel silicon-gate depletion-load technology, and is packaged in a 40-pin plastic or ceramic DIP (Figures 1 and 2).

PIN DESCRIPTION

B/$\overline{\text{A}}$. *Channel A or B Select* (input, High selects Channel B). This input defines which channel is accessed during a data transfer between the CPU and the Z80 DART.

C/$\overline{\text{D}}$. *Control or Data Select* (input, High selects Control). This input specifies the type of information (control or data) transferred on the data bus between the CPU and the Z80 DART.

$\overline{\text{CE}}$. *Chip Enable* (input, active Low). A Low at this input enables the Z80 DART to accept command or data input from the CPU during a write cycle, or to transmit data to the CPU during a read cycle.

CLK. *System Clock* (input). The Z80 DART uses the standard Z80 single-phase system clock to synchronize internal signals.

$\overline{\text{CTSA}}$, $\overline{\text{CTSB}}$. *Clear To Send* (inputs, active Low). When programmed as Auto Enables, a Low on these inputs enables the respective transmitter. If not programmed as Auto Enables, these inputs may be programmed as general-purpose inputs. Both inputs are Schmitt-trigger buffered to accommodate slow-risetime signals.

D_0-D_7. *System Data Bus* (bidirectional, 3-state). This bus transfers data and commands between the CPU and the Z80 DART.

$\overline{\text{DCDA}}$, $\overline{\text{DCDB}}$. *Data Carrier Detect* (inputs, active Low). These pins function as receiver enables if the Z80 DART is programmed for Auto Enables; otherwise they may be used as general-purpose input pins. Both pins are Schmitt-trigger buffered.

$\overline{\text{DTRA}}$, $\overline{\text{DTRB}}$. *Data Terminal Ready* (outputs, active Low). These outputs follow the state programmed into the DTR bit. They can also be programmed as general-purpose outputs.

IEI. *Interrupt Enable In* (input, active High). This signal is used with IEO to form a priority daisy chain when there is more than one interrupt-driven device. A High on this line indicates that no other device of higher priority is being serviced by a CPU interrupt service routine.

IEO. *Interrupt Enable Out* (output, active High). IEO is High only if IEI is High and the CPU is not servicing an interrupt from this Z80 DART. Thus, this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU interrupt service routine.

$\overline{\text{INT}}$. *Interrupt Request* (output, open drain, active Low). When the Z80 DART is requesting an interrupt, it pulls $\overline{\text{INT}}$ Low.

$\overline{\text{M1}}$. *Machine Cycle One* (input from Z80 CPU, active Low). When $\overline{\text{M1}}$ and $\overline{\text{RD}}$ are both active, the Z80 CPU is fetching an instruction from memory; when $\overline{\text{M1}}$ is active while $\overline{\text{IORQ}}$ is active, the Z80 DART accepts $\overline{\text{M1}}$ and $\overline{\text{IORQ}}$ as an interrupt acknowledge if the Z80 DART is the highest priority device that has interrupted the Z80 CPU.

$\overline{\text{IORQ}}$. *Input/Output Request* (input from CPU, active Low). $\overline{\text{IORQ}}$ is used in conjunction with B/$\overline{\text{A}}$, C/$\overline{\text{D}}$, $\overline{\text{CE}}$, and $\overline{\text{RD}}$ to transfer commands and data between the CPU and the Z80 DART. When $\overline{\text{CE}}$, $\overline{\text{RD}}$, and $\overline{\text{IORQ}}$ are all active, the channel selected by B/$\overline{\text{A}}$ transfers data to the CPU (a read operation). When $\overline{\text{CE}}$ and $\overline{\text{IORQ}}$ are active, but $\overline{\text{RD}}$ is inactive, the channel selected by B/$\overline{\text{A}}$ is written to by the CPU with either data or control information as specified by C/$\overline{\text{D}}$.

$\overline{\text{RxCA}}$, $\overline{\text{RxCB}}$. *Receiver Clocks* (inputs). Receive data is sampled on the rising edge of RxC. The Receive Clocks may be 1, 16, 32, or 64 times the data rate.

$\overline{\text{RD}}$. *Read Cycle Status* (input from CPU, active Low). If $\overline{\text{RD}}$ is active, a memory or I/O read operation is in progress.

RxDA, RxDB. *Receive Data* (inputs, active High).

$\overline{\text{RESET}}$. *Reset* (input, active Low). Disables both receivers and transmitters, forces TxDA and TxDB marking, forces the modem controls High, and disables all interrupts.

$\overline{\text{RIA}}$, $\overline{\text{RIB}}$. *Ring Indicator* (inputs, active Low). These inputs are similar to CTS and DCD. The Z80 DART detects both logic level transitions and interrupts the CPU. When not used in switched-line applications, these inputs can be used as general-purpose inputs.

$\overline{\text{RTSA}}$, $\overline{\text{RTSB}}$. *Request to Send* (outputs, active Low). When the RTS bit is set, the RTS output goes Low. When the $\overline{\text{RTS}}$ bit is reset, the output goes High after the transmitter empties.

$\overline{\text{TxCA}}$, $\overline{\text{TxCB}}$. *Transmitter Clocks* (inputs). TxD changes on the falling edge of $\overline{\text{TxC}}$. The Transmitter Clocks may be 1, 16, 32, or 64 times the data rate; however, the clock multiplier for the transmitter and the receiver must be the same. The Transmit Clock inputs are Schmitt-trigger buffered. Both the Receiver and Transmitter Clocks may be driven by the Z80 CTC Counter Time Circuit for programmable baud rate generation.

TxDA, TxDB. *Transmit Data* (outputs, active High).

$\overline{\text{W}}$/$\overline{\text{RDYA}}$, $\overline{\text{W}}$/$\overline{\text{RDYB}}$. *Wait/Ready* (outputs, open drain when programmed for Wait function, driven High and Low when programmed for Ready function). These dual-purpose outputs may be programmed as Ready lines for a DMA controller or as Wait lines that synchronize the CPU to the Z80 DART data rate. The reset state is open drain.

FUNCTIONAL DESCRIPTION

The functional capabilities of the Z80 DART can be described from two different points of view: as a data communications device, it transmits and receives serial data, and meets the requirements of asynchronous data communications protocols; as a Z80 family peripheral, it interacts with the Z80 CPU and other Z80 peripheral circuits, and shares the data, address, and control buses, as well as being a part of the Z80 interrupt structure. As a peripheral to other microprocessors, the Z80 DART offers valuable features such as nonvectored interrupts, polling, and simple handshake capability.

The first part of the following functional description introduces Z80 DART data communications capabilities; the second part describes the interaction between the CPU and the Z80 DART.

The Z80 DART offers RS-232 serial communications support by providing device signals for external modem control. In addition to dual-channel Request To Send, Clear To Send, and Data Carrier Detect ports, the Z80 DART also features a dual channel Ring Indicator (RIA, RIB) input to facilitate local/remote or station-to-station communication capability. Figure 3 is a block diagram.

Communications Capabilities. The Z80 DART provides two independent full-duplex channels for use as an asynchronous receiver/transmitter. The following is a short description of receiver/transmitter capabilities. For more details, refer to the Asynchronous Mode section of the *Z80 SIO Technical Manual* (03-3033-01).

The Z80 DART offers transmission and reception of five to eight bits per character, plus optional even or odd parity. The transmitter can supply one, one and a half, or two stop bits per character and can provide a break output at any time. The receiver break detection logic interrupts the CPU both at the start and end of a received break. Reception is protected from spikes by a transient spike rejection mechanism that checks the signal one-half a bit time after a Low level is detected on the Receive Data input. If the Low does not persist—as in the case of a transient—the character assembly process is not started.

Framing errors and overrun errors are detected and buffered together with the character on which they occurred. Vectored interrupts allow fast servicing of interrupting conditions using dedicated routines. Furthermore, a built-in checking process avoids interpreting a framing error as a new start bit: a framing error results in the addition of one-half a bit time to the point at which the search for the next start bit is begun.

The Z80 DART does not require symmetric Transmit and Receive Clock signals, a feature that allows it to be used with a Z80 CTC or any other clock source. The transmitter and receiver can handle data at a rate of 1, 1/16, 1/32, or 1/64 of the clock rate supplied to the Receive and Transmit Clock inputs. When using Channel B, the bit rates for transmit and receive operations must be the same because $\overline{\text{RxC}}$ and $\overline{\text{TxC}}$ are bonded together ($\overline{\text{RxTxCB}}$).

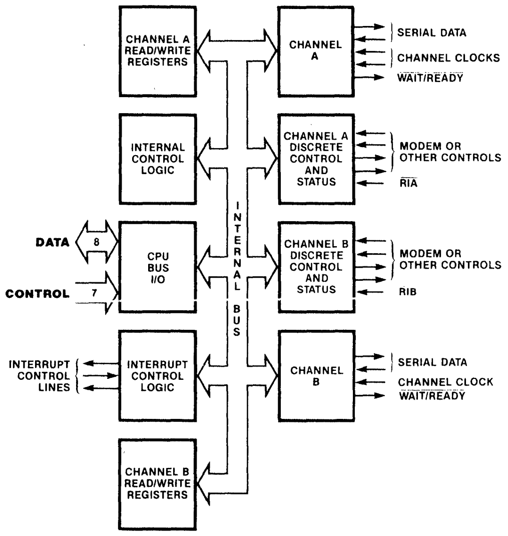


Figure 3. Block Diagram

I/O Interface Capabilities. The Z80 DART offers the choice of Polling, Interrupt (vectored or non-vectored) and Block Transfer modes to transfer data, status, and control information to and from the CPU. The Block Transfer mode can be implemented under CPU or DMA control.

Polling. There are no interrupts in the Polled mode. Status registers RR0 and RR1 are updated at appropriate times for each function being performed. All the interrupt modes of the Z80 DART must be disabled to operate the device in a Polled environment.

While in its Polling sequence, the CPU examines the status contained in RR0 for each channel; the RR0 status bits serve as an acknowledge to the Poll inquiry. The two RR0 status bits D_0 and D_2 indicate that a data transfer is needed. The status also indicates Error or other special status conditions. The Z80 DART Programming section contains more information. The Special Receive Condition status contained in RR1 does not have to be read in a Polling sequence because the status bits in RR1 are accompanied by a Receive Character Available status in RR0.

Interrupts. The Z80 DART offers an elaborate interrupt scheme that provides fast interrupt response in real-time applications. As a member of the Z80 family, the Z80 DART can be daisy-chained along with other Z80 peripherals for peripheral interrupt-priority resolution. In addition, the internal interrupts of the Z80 DART are nested to prioritize the various interrupts generated by Channels A and B. Channel B registers WR2 and RR2 contain the interrupt vector that points to an interrupt service routine in the memory. To eliminate the necessity of writing a status analysis routine, the Z80 DART can modify the interrupt vector in RR2 so it points directly to one of eight interrupt service routines. This is done under program control by setting a program bit (WR1, D_2) in Channel B called "Status

Affects Vector." When this bit is set, the interrupt vector in RR2 is modified according to the assigned priority of the various interrupting conditions.

Transmit interrupts, Receive interrupts, and External/Status interrupts are the main sources of interrupts. Each interrupt source is enabled under program control with Channel A having a higher priority than Channel B, and with Receiver, Transmit, and External/Status interrupts prioritized in that order within each channel. When the Transmit interrupt is enabled, the CPU is interrupted by the transmit buffer becoming empty. (This implies that the transmitter must have had a data character written into it so it can become empty.) When enabled, the receiver can interrupt the CPU in one of three ways:

- Interrupt on the first received character
- Interrupt on all received characters
- Interrupt on a Special receive condition

Interrupt On First Character is typically used with the Block Transfer mode. Interrupt On All Received Characters can optionally modify the interrupt vector in the event of a parity error. The Special Receive Condition interrupt can occur on a character basis. The Special Receive Condition can cause an interrupt can occur on a character basis. The Special Receive condition can cause an interrupt only if the Interrupt On First Received Character or Interrupt On All Received Characters mode is selected. In Interrupt On First Receive Character, an interrupt can occur from Special Receive conditions (except Parity Error) after the first Received character interrupt (example: Receive Overrun interrupt).

The main function of the External/Status interrupt is to monitor the signal transitions of the $\overline{\text{CTS}}$, $\overline{\text{DCD}}$, and $\overline{\text{RI}}$ pins; however, an External/Status interrupt is also caused by the detection of a Break sequence in the data stream. The interrupt caused by the Break sequence has a special feature that allows the Z80 DART to interrupt when the Break sequence is detected or terminated. This feature facilitates the proper termination of the current message, correct initialization of the next message, and the accurate timing of the Break condition.

CPU/DMA Block Transfer. The Z80 DART provides a Block Transfer mode to accommodate CPU block transfer functions and DMA block transfers (Z80 DMA or other designs). The Block Transfer mode uses the $\text{W}/\overline{\text{RDY}}$ output in conjunction with the Wait/Ready bits of Write Register 1. The $\text{W}/\overline{\text{RDY}}$ output can be defined under software control as a Wait line in the CPU Block Transfer mode or as a Ready line in the DMA Block Transfer mode.

To a DMA controller, the Z80 DART Ready output indicates that the Z80 DART is ready to transfer data to or from memory. To the CPU, the Wait output indicates that the Z80 DART is not ready to transfer data, thereby requesting the CPU to extend the I/O cycle.

INTERNAL ARCHITECTURE

The device internal structure includes a Z80 CPU interface, internal control and interrupt logic, and two full-duplex channels. Each channel contains read and write registers, and discrete control and status logic that provides the interface to modems or other external devices.

The read and write register group includes five 8-bit control registers and two status registers. The interrupt vector is written into an additional 8-bit register (Write Register 2) in Channel B, that may be read through Read Register 2 in Channel B. The registers for both channels are designated as follows:

WR0-WR5	Write Registers 0 through 5
RR0-RR2	Read Registers 0 through 2

The bit assignment and functional grouping of each register is configured to simplify and organize the programming process.

The logic for both channels provides formats, bit synchronization, and validation for data transferred to and from the channel interface. The modem control inputs Clear to Send ($\overline{\text{CTS}}$), Data Carrier Detect ($\overline{\text{DCD}}$), and Ring Indicator ($\overline{\text{RI}}$) are monitored by the control logic under program control. All the modem control signals are general purpose in nature and can be used for functions other than modem control.

For automatic interrupt vectoring, the interrupt control logic determines which channel and which device within the channel has the highest priority. Priority is fixed with Channel A assigned a higher priority than Channel B; Receive, Transmit, and External/Status interrupts are prioritized in that order within each channel.

Data Path. The transmit and receive data path illustrated for Channel A in Figure 4 is identical for both channels. The receiver has three 8-bit buffer registers in a FIFO arrangement in addition to the 8-bit receive shift register. This scheme creates additional time for the CPU to service a Receive Character Available interrupt in a high-speed data transfer.

The transmitter has an 8-bit transmit data register that is loaded from the internal data bus, and a 9-bit transmit shift register that is loaded from the transmit data register.

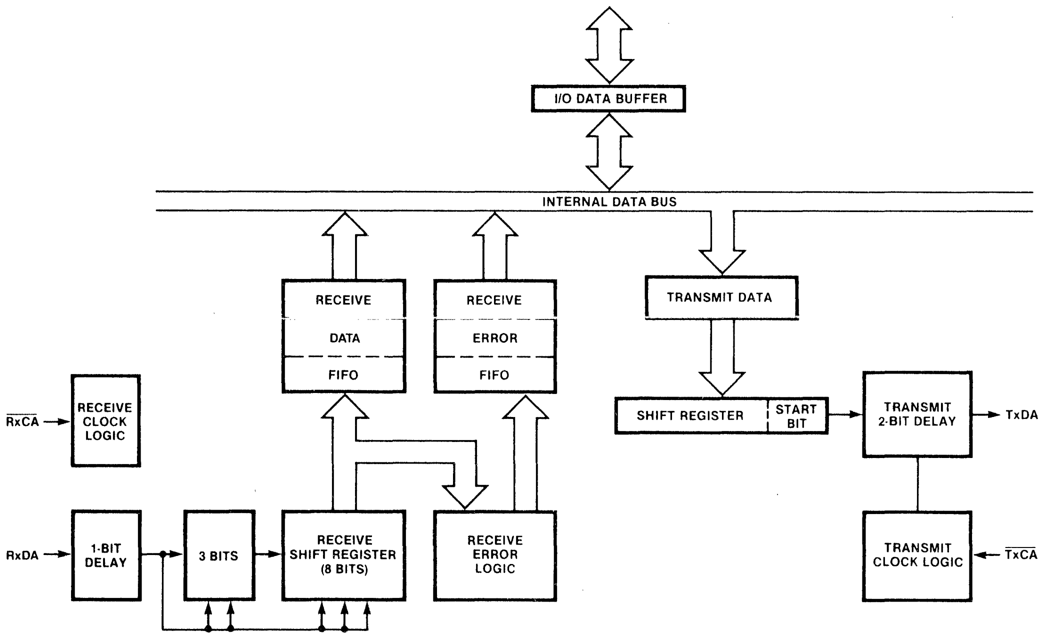


Figure 4. Data Path

READ, WRITE AND INTERRUPT TIMING

Read Cycle. The timing signals generated by a Z80 CPU input instruction to read a Data or Status byte from the Z80 DART are illustrated in Figure 5.

Write Cycle. Figure 6 illustrates the timing and data signals generated by a Z80 CPU output instruction to write a Data or Control byte into the Z80 DART.

Interrupt Acknowledge Cycle. (Figure 7) After receiving an Interrupt Request signal ($\overline{INT}$ pulled Low), the Z80 CPU sends an Interrupt Acknowledge signal ($\overline{M1}$ and $\overline{IORQ}$ both Low). The daisy-chained interrupt circuits determine the highest priority interrupt requestor. The IEI of the highest priority peripheral is terminated High. For any peripheral that has no interrupt pending or under service, IEO = IEI. Any peripheral that does have an interrupt pending or under service forces its IEO Low.

To insure stable conditions in the daisy chain, all interrupt status signals are prevented from changing while $\overline{M1}$ is Low. When $\overline{IORQ}$ is Low, the highest priority interrupt requestor (the one with IEI High) places its interrupt vector on the data bus and sets its internal interrupt-under-service latch.

Refer to the *Technical Manual* (03-3033-01) for additional details on the interrupt daisy chain and interrupt nesting.

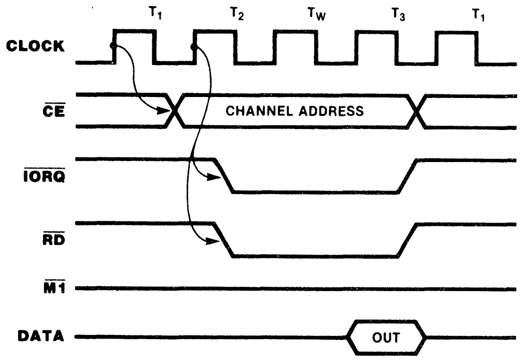


Figure 5. Read Cycle

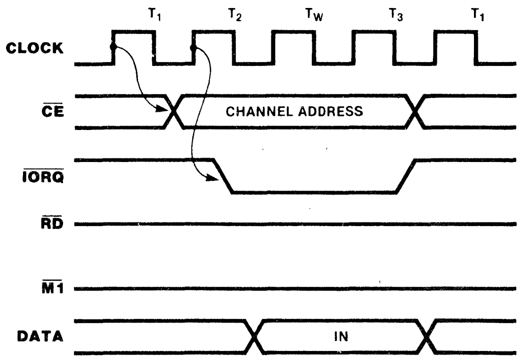


Figure 6. Write Cycle

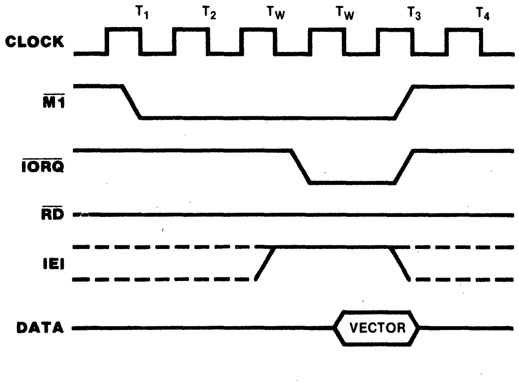


Figure 7. Interrupt Acknowledge Cycle

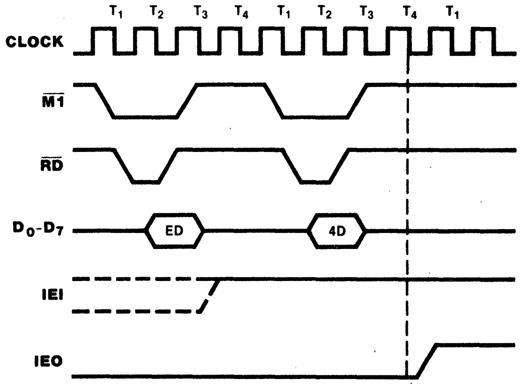


Figure 8. Return from Interrupt Cycle

Return From Interrupt Cycle. (Figure 8) Normally, the Z80 CPU issues an RETI (Return From Interrupt) instruction at the end of an interrupt service routine. RETI is a 2-byte opcode (ED-4D) that resets the interrupt-under-service latch to terminate the interrupt that has just been processed.

When used with other CPUs, the Z80 DART allows the user to return from the interrupt cycle with a special command called "Return From Interrupt" in Write Register 0 of Channel A. This command is interpreted by the Z80 DART in exactly the same way it would interpret an RETI command on the data bus.

Z80 DART PROGRAMMING

To program the Z80 DART, the system program first issues a series of commands that initialize the basic mode and then other commands that qualify conditions within the selected mode. For example, the character length, clock rate, number of stop bits, even or odd parity are first set, then the Interrupt mode and, finally, receiver or transmitter enable.

Both channels contain command registers that must be programmed via the system program prior to operation. The Channel Select input (B/$\overline{A}$) and the Control/Data input (C/$\overline{D}$) are the command structure addressing controls, and are normally controlled by the CPU address bus.

Write Registers. The Z80 DART contains six registers (WR0-WR5) in each channel that are programmed separately by the system program to configure the functional personality of the channels (Figure 4). With the exception of WR0, programming the write registers requires two bytes. The first byte contains three bits (D_0-D_2) that point to the selected register; the second byte is the actual control word that is written into the register to configure the Z80 DART.

WR0 is a special case in that all the basic commands (CMD_0-CMD_2) can be accessed with a single byte. Reset (internal or external) initializes the pointer bits D_0-D_2 to point to WR0. This means that a register cannot be pointed to in the same operation as a channel reset.

Write Register Functions

WR0	Register pointers, initialization commands for the various modes
WR1	Transmit/Receive interrupt and data transfer mode definition
WR2	Interrupt vector (Channel B only)
WR3	Receive parameters and control
WR4	Transmit/Receive miscellaneous parameters and modes
WR5	Transmit parameters and controls

Read Registers. The Z80 DART contains three registers (RR0-RR2) that can be read to obtain the status information for each channel (except for RR2, which applies to Channel B only). The status information includes error conditions, interrupt vector, and standard communications-interface signals.

To read the contents of a selected read register other than RR0, the system program must first write the pointer byte to WR0 in exactly the same way as a write register operation. Then, by executing an input instruction, the contents of the addressed read register can be read by the CPU.

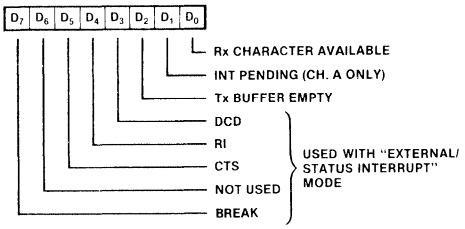
The status bits of RR0 and RR1 are carefully grouped to simplify status monitoring. For example, when the interrupt vector indicates that a Special Receive Condition interrupt has occurred, all the appropriate error bits can be read from a single register (RR1).

Read Register Functions

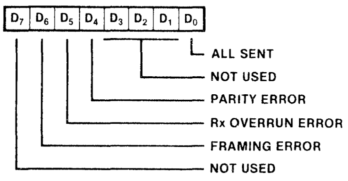
RR0	Transmit/Receive buffer status, interrupt status and external status
RR1	Special Receive Condition status
RR2	Modified interrupt vector (Channel B only)

Z80 DART READ AND WRITE REGISTERS

READ REGISTER 0

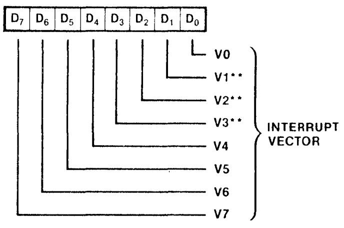


READ REGISTER 1*



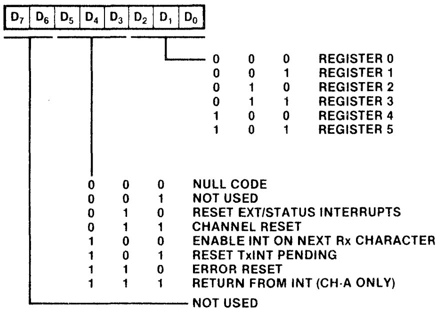
*Used With Special Receive Condition Mode.

READ REGISTER 2

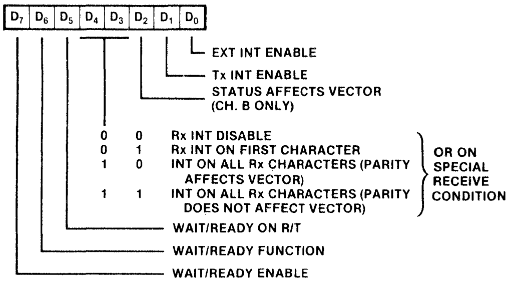


**Variable if "Status Affects Vector" is Programmed.

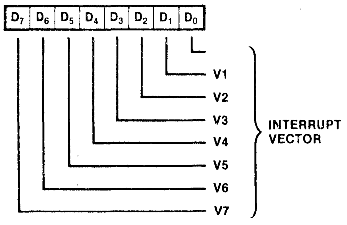
WRITE REGISTER 0



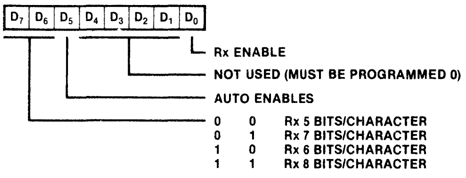
WRITE REGISTER 1


D7 D6 D5 D4 D3 D2 D1 D0
EXT INT ENABLE
Tx INT ENABLE
STATUS AFFECTS VECTOR (CH. B ONLY)
0 0 Rx INT DISABLE
0 1 Rx INT ON FIRST CHARACTER
1 0 INT ON ALL Rx CHARACTERS (PARITY AFFECTS VECTOR)
1 1 INT ON ALL Rx CHARACTERS (PARITY DOES NOT AFFECT VECTOR)
OR ON SPECIAL RECEIVE CONDITION
WAIT/READY ON R/T
WAIT/READY FUNCTION
WAIT/READY ENABLE

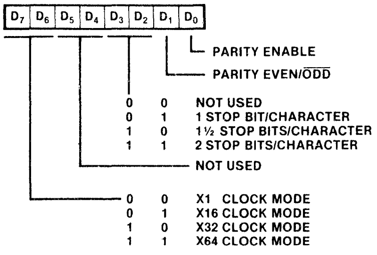
WRITE REGISTER 2 (Channel B only)



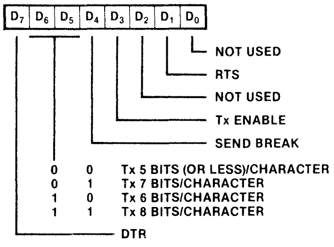
WRITE REGISTER 3



WRITE REGISTER 4



WRITE REGISTER 5



ABSOLUTE MAXIMUM RATINGS

Voltages on all pins with respect to GND	−0.3V to +7V
Operating Ambient Temperature	See Ordering Information
Storage Temperature	−65°C to +150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

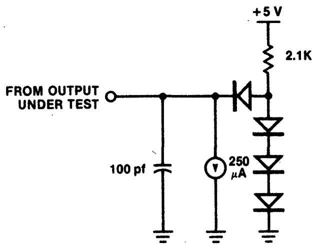
STANDARD TEST CONDITIONS

The DC characteristics and capacitance sections listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND (0V). Positive current flows into the referenced pin.

Available operating temperature ranges are:

- S = 0°C to +70°C, $+4.75V \leqslant V_{CC} \leqslant +5.25V$

The Ordering Information section lists package temperature ranges and product numbers. Package drawings are in the Package Information section. Refer to the Literature List for additional documentation.



DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	−0.3	+0.45	V	
V_{IHC}	Clock Input High Voltage	V_{CC} − 0.6		V	V_{cc} + 0.3V
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	+2.0	+5.5	V	
V_{OL}	Output Low Voltage		+0.4	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	+2.4		V	I_{OH} = −250 µA
I_L	Input/3-State Output Leakage Current	−10	+10	µA	$0.4 < V_{IN} < 2.4V$
$I_{L(RI)}$	RI Pin Leakage Current	−40	+10	µA	$0.4 < V_{IN} < 2.4V$
I_{CC}	Power Supply Current		100	mA	

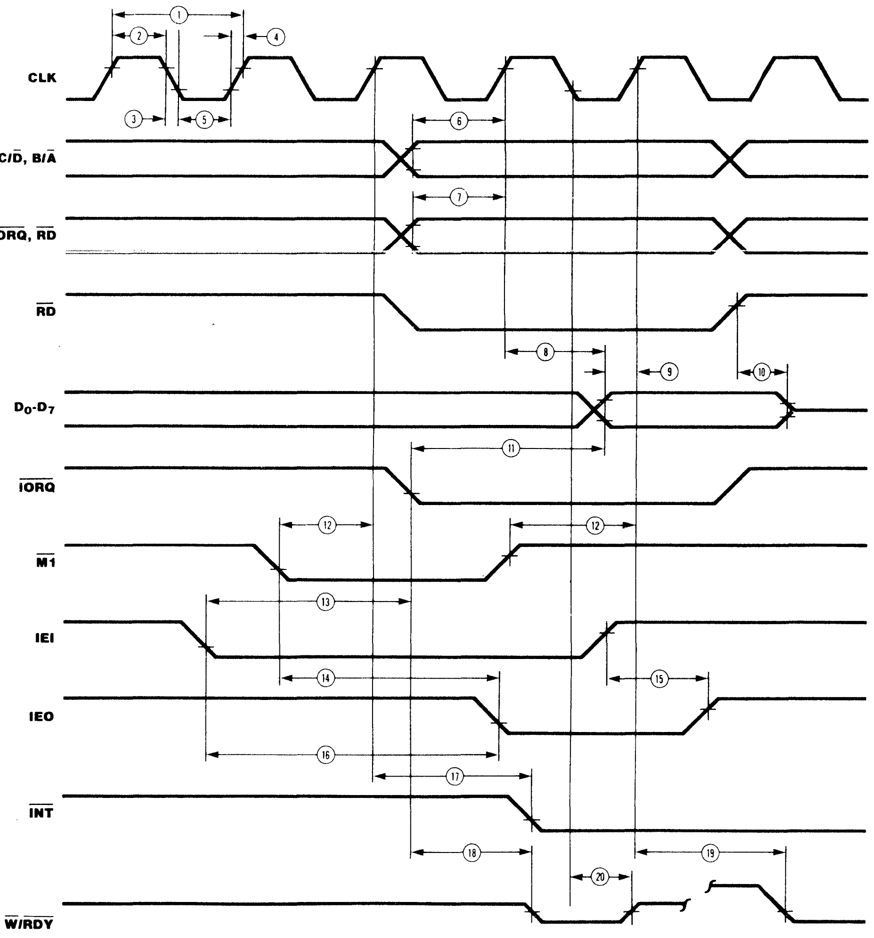
T_A = 0°C to 70°C, V_{CC} = +5V, ±5%.

CAPACITANCE

Symbol	Parameter	Min	Max	Unit
C	Clock Capacitance		40	pf
C_{IN}	Input Capacitance		5	pf
C_{OUT}	Output Capacitance		15	pf

Over specified temperature range; f = 1 MHz.
Unmeasured pins returned to ground.

AC CHARACTERISTICS TIMING

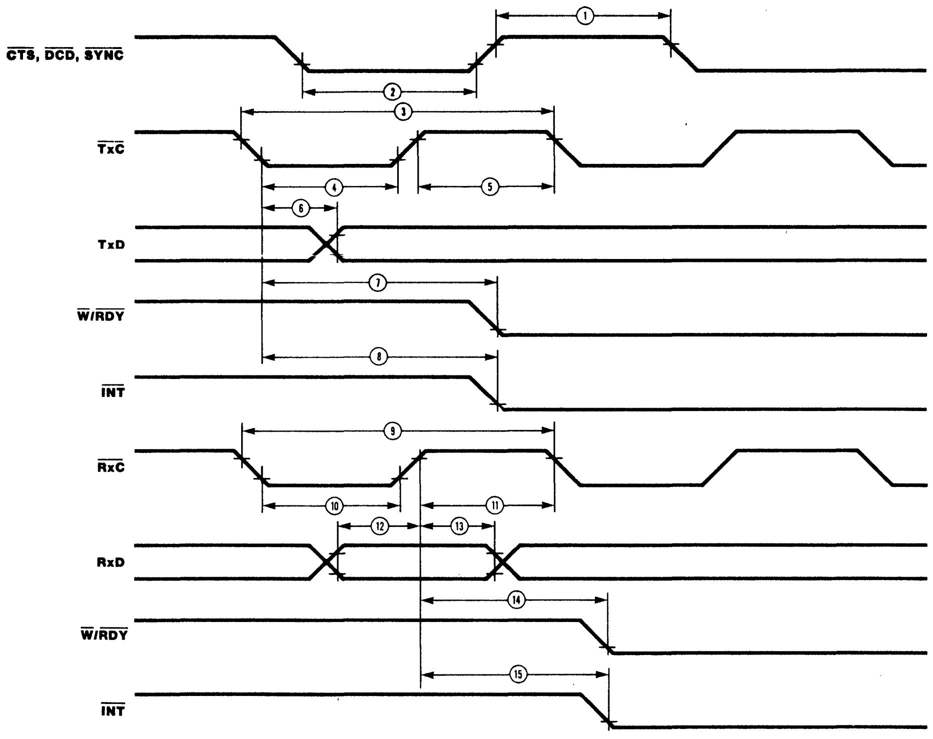


AC CHARACTERISTICS

			Z0847004		Z0847006	
Number	**Symbol**	**Parameter**	**Min**	**Max**	**Min**	**Max**
1	TcC	Clock Cycle Time	250	4000	165	4000
2	TwCh	Clock Width (High)	105	2000	70	2000
3	TfC	Clock Fall Time		30		15
4	TrC	Clock Rise Time		30		15
5	TwCl	Clock Width (Low)	105	2000	70	2000
6	TsAD(C)	$\overline{CE}$, C/$\overline{D}$, B/$\overline{A}$ to Clock ↑ Setup Time	145		60	
7	TsCS(C)	$\overline{IORQ}$, $\overline{RD}$ to Clock ↑ Setup Time	115		60	
8	TdC(DO)	Clock ↑ to Data Out Delay		220		150
9	TsDI(C)	Data In to Clock ↑ Setup (Write or M1 Cycle)	50		30	
10	TdRD(DOz)	$\overline{RD}$ ↑ to Data Out Float Delay		110		90
11	TdIO(DOI)	$\overline{IORQ}$ ↓ to Data Out Delay (INTACK Cycle)		160		100
12	TsM1(C)	$\overline{M1}$ to Clock ↑ Setup Time	90		75	
13	TsIEI(IO)	IEI to $\overline{IORQ}$ ↓ Setup Time (INTACK Cycle)	140		120	
14	TdM1(IEO)	$\overline{M1}$ ↓ to IEO ↓ Delay (interrupt before M1)		190		160
15	TdIEI(IEOr)	IEI ↑ to IEO ↑ Delay (after ED decode)		100		70
16	TdIEI(IEOf)	IEI ↓ to IEO ↓ Delay		100		70
17	TdC(INT)	Clock ↑ to INT ↓ Delay		200		150
18	TdIO(W/RWf)	$\overline{IORQ}$ ↓ or $\overline{CE}$ ↓ to W/$\overline{RDY}$ ↓Delay (Wait Mode)		210		175
19	TdC(W/RR)	Clock ↑ to W/$\overline{RDY}$ ↓ Delay (Ready Mode)		120		100
20	TdC(W/RWz)	Clock ↓ to W/$\overline{RDY}$ Float Delay (Wait Mode)		130		110

*Units in nanoseconds (ns).

AC CHARACTERISTICS (Continued)



			Z0847004		Z0847006		
Number	**Symbol**	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Notes***
1	TwPh	Pulse Width (High)	200		200		2
2	TwPl	Pulse Width (Low)	200		200		2
3	TcTxC	TxC Cycle Time	400	∞	330	∞	2
4	TwTxCl	TxC Width (Low)	180	∞	100	∞	2
5	TwTxCh	TxC Width (High)	180	∞	100	∞	2
6	TdTxC(TxD)	TxC ↓ to TxD Delay		300		220	2
7	TdTxC(W/RRf)	TxC ↓ to W/RDY ↓ Delay (Ready Mode)	5	9	5	9	1
8	TdTxC(INT)	TxC ↓ to INT ↓ Delay	5	9	5	9	1
9	TcRxC	RxC Cycle Time	400	∞	330	∞	2
10	TwRxCl	RxC Width (Low)	180	∞	100	∞	2
11	TwRxCh	RxC Width (High)	180	∞	100	∞	2
12	TsRxD(RxC)	RxD to RxC ↑ Setup Time (x1 Mode)	0		0		2
13	ThRxD(RxC)	RxD Hold Time (x1 Mode)	140		100		2
14	TdRxC(W/RRf)	RxC ↑ to W/RDY ↓ Delay (Ready Mode)	10	13	10	13	1
15	TdRxC(INT)	RxC ↑ to INT ↓ Delay	10	13	10	13	1

* In all modes, the System Clock rate must be at least five times the maximum data rate. RESET must be active a minimum of one complete clock cycle.
1. Units equal to System Clock Periods.
2. Units in nanoseconds (ns).

January 1989

Z84C80 CMOS Z80® GLU General Logic Unit

FEATURES:

- On-Chip Clock Oscillator with Power-Save Monitor Circuitry
- Dynamic Memory Interface Controller
- Static Memory Interface
- Memory and I/O Chip Selects
- Reset Synchronization and Power-On Reset
- Watchdog Timer
- Z80 CPU to Z8500 Peripheral Interface
- General Purpose Outputs
- 5 Wait State Generators (WSG)
  - Static Memory WSG
  - I/O WSG
  - Interrupt Acknowledge Cycle Timing Stretch
  - RETI Cycle Timing Stretch
  - Opcode Fetch WSG
- 68-pin PLCC Package
- Single +5 Volt Power Supply

GENERAL DESCRIPTION:

Zilog's new Z84C80 General Logic Unit (hereafter referred to as the Z80 GLU) is a programmable, multi-purpose interface controller designed to perform many of the functions required to "glue" a Z80-based microprocessor system together. The CPU programs the Z80 GLU to interface with a wide range of peripheral devices, both memories and I/O.

In offering the features found in most system designs, the Z84C80 can replace approximately 100 SSI and MSI packages with a single 68-pin PLCC. By combining the most used logical functions onto a single piece of silicon, the Z80 GLU chip offers a cost effective and powerful solution to the size and complexity constraints in a system design. This solution will allow the systems designer to simplify the design, decrease the time to market, and reduce costs.

Designed and manufactured in N-Well CMOS, the device offers high-speed performance and low power consumption. Zilog's CMOS process provides these features plus a high degree of reliability.

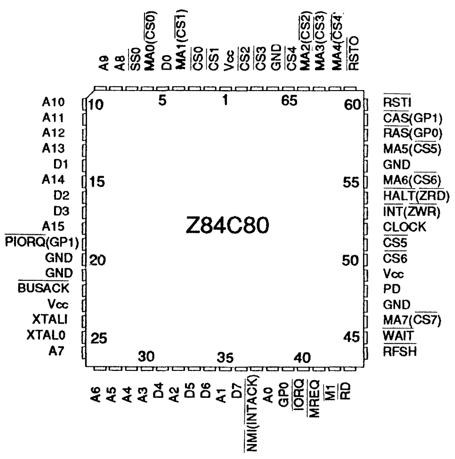


Figure 1: Z84C80 Pin Functions

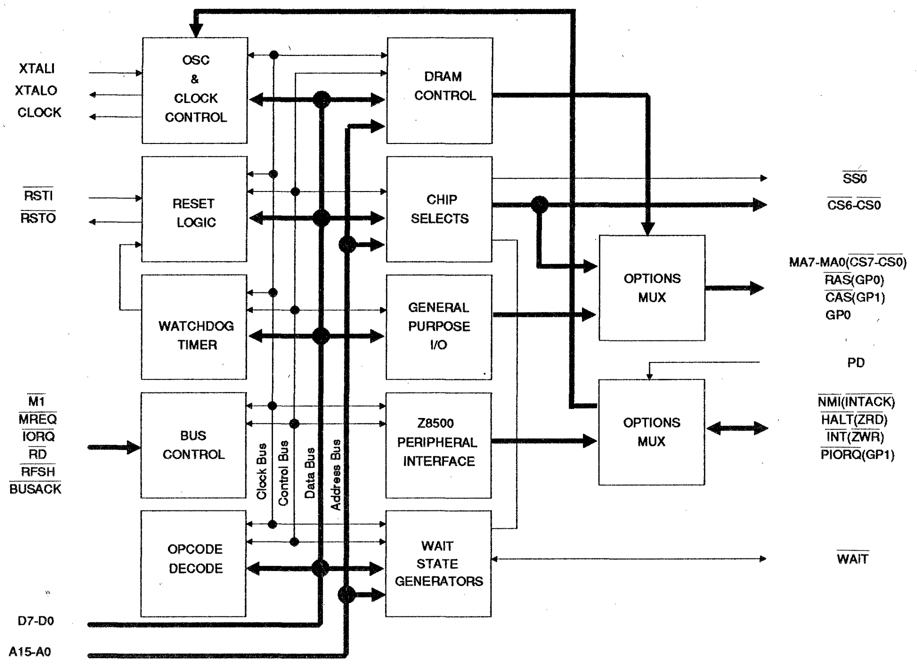


Figure 2. Z84C80 Block Diagram

PIN DESCRIPTIONS:

The device pinout is depicted in Figure 1 and the block diagram is shown in Figure 2.

A_{15}-A_0. *Address Bus* (Input, active High). A_{15}-A_0 form a 16-bit address bus to provide the means for controlling memory data bus exchanges (up to 64K bytes) and I/O device exchanges.

$\overline{BUSACK}$. *Bus Acknowledge* (Input, active Low). As an output from the CPU, $\overline{BUSACK}$ indicates that the CPU has relinquished control of the system bus (address, data, and some control) to external circuitry.

$\overline{CAS}$. *Column Address Strobe* (Output, active Low). This signal (along with $\overline{PIORQ}$) is multiplexed with the GP_1 output. As a $\overline{CAS}$ strobe signal, this output is used by the DRAM interface controller to signal the DRAM device to latch the current contents of the multiplexed address bus (MA_7-MA_0) in order to address the correct column within the DRAM.

CLOCK. System Clock (Output, active High). This output is used to provide the standard single-phase clock for either a Z80 NMOS or CMOS system.

$\overline{CS_7}$-$\overline{CS_0}$. *Chip Selects* (Outputs, acitve Low). These outputs are used to select memory and/or I/O devices for data exchanges. The output is active depending upon the decoded machine cycle and the contents of the address bus. The $\overline{CS_7}$-$\overline{CS_0}$ signals are multiplexed with MA_7-MA_0 signals. The $\overline{CS_6}$-$\overline{CS_0}$ signals are also available separately. When the DRAM Interface Controller is enabled, these pins serve as the multiplexed address bus while the seperate $\overline{CS_6}$-$\overline{CS_0}$ pins can be used for chip selects. When the DRAM Interface Controller is disabled, the chip select outputs appear on these multiplexed pins as well as the seperate $\overline{CS_6}$-$\overline{CS_0}$ pins.

D_7-D_0. *Data Bus* (Input, active High). D_7-D_0 form an 8-bit data bus that is used for programming the Z84C80 GLU during I/O transfers. It is also used for decoding the instruction sequences for wait state insertion during the RETI sequence.

GP_0. *General Purpose Output 0* (Output, active High). This signal is available on pin 39, and is also multiplexed with the $\overline{RAS}$ output . When the dynamic memory interface is not enabled, this output also becomes available to the user on the $\overline{RAS}$ pin. This output can be controlled through the General Purpose Output Control Register.

GP_1. *General Purpose Output 1* (Output, active High). This signal is multiplexed with two other pins, the $\overline{PIORQ}$ output and the $\overline{CAS}$ output. When the Z08500 peripheral interface and the interrupt acknowledge wait state generator are not enabled, this output becomes available to the user on the $\overline{PIORQ}$ pin. When the dynamic memory interface is not enabled, this output becomes available to the user on the $\overline{CAS}$ pin. This output can be controlled through the General Purpose Output Control Register.

$\overline{HALT}$. *Halt Acknowledge* (Input, active Low). This pin is multiplexed with the $\overline{ZRD}$ output. When the Power-Down Interface is enabled (the Z8500 peripheral interface is not enabled), this pin is the $\overline{HALT}$ input to the device. it is used by the Z80 GLU to control entry to the power-down mode of operation for the Z80 CMOS CPU.

$\overline{INT}$. *Interrrupt Request* (Input, active Low). This pin is multiplexed with the $\overline{ZWR}$ line. When the Power-Down Interface is enabled (the Z8500 Peripheral Interface is not enabled), this pin is the $\overline{INT}$ (interrupt) input to the device. It is used by the Z80 GLU to control exit from the power-down mode of operation for the Z80 CMOS CPU.

$\overline{INTACK}$. *Z8500 Interrupt Acknowledge* (Output, active Low). This signal is multiplexed with the $\overline{NMI}$ input. When the Power-Down interface is not enabled (the Z8500 peripheral interface is enabled), this pin is the $\overline{INTACK}$ output from the device. This signal is used for the Z8500 peripherals during the interrupt acknowledge cycle.

$\overline{IORQ}$. *Input/Output Request* (Input, active Low). This signal is used to select the Z80 GLU device during programming and also to assist in the decoding of the I/O chip select outputs.

$\overline{M1}$. *M1 Cycle* (Input, active Low). This signal is used to decode the opcode fetch and interrupt acknowledge machine cycles for the wait state generators, code/data separation, and RETI logic.

MA_7-MA_0. *Multiplexed Address Bus* (Output, active High). These lines are multiplexed with the $\overline{CS_7}$-$\overline{CS_0}$ lines. When the DRAM Interface Controller is enabled, these pins serve as the multiplexed (row and column) address bus required to interface to 64K dynamic memories. During dynamic memory access cycles, these pins provide both halves of the required address. During the refresh cycles these pins provide an 8-bit refresh address to the dynamic memory.

PIN DESCRIPTIONS (cont):

$\overline{\text{MREQ}}$. *Memory Request* (Input, active Low). This signal is used to assist in the decoding of the chip select outputs for memory access and for the dynamic memory interface.

$\overline{\text{NMI}}$. *Non-Maskable Interrupt Request* (Input, active Low). This signal is multiplexed with the $\overline{\text{INTACK}}$ output. When the Power-Down Interface is enabled (the Z8500 Peripheral Interface is not enabled), this pin is the NMI input to the device. It is used by the Z80 GLU to control exit from the power-down mode operation for the Z80 CMOS CPU.

PD. *Power-Down Option* (Input, active High, Z84C80 only). When pulled to V_{cc}, the Power-Down feature is enabled. When pulled to GND, the Z8500 Peripheral Interface is enabled.

$\overline{\text{PIORQ}}$. *Peripheral Input/Output Request* (Output, active Low). This signal is multiplexed with the GP1 output. When either the Z8500 Peripheral Interface is enabled or the interrupt acknowledge wait state generator is enabled, this output is a delayed $\overline{\text{IORQ}}$ signal to Z80 Peripherals.

$\overline{\text{RAS}}$. *Row Address Strobe* (Output, active Low). This signal is multiplexed with the GP_0 output. As a $\overline{\text{RAS}}$ strobe signal, this output is used by the DRAM interface controller to signal the DRAM device to latch the current contents of the multiplexed address bus (MA_7-MA_0) in order to address the correct row within the DRAM.

$\overline{\text{RD}}$. *Read* (Input, active Low). This input is used to assist in the determination between I/O read and write cycles.

$\overline{\text{RFSH}}$. *Refresh* (Input, active Low). This signal is used to indicate when the system bus (address and control) are idle so that dynamic memory refresh can occur.

$\overline{\text{RSTI}}$. *Reset In* (Input, active Low). This is a reset request input from the external system.

$\overline{\text{RSTO}}$. *Reset Out* (Output, active Low). This is the synchronized reset output for the system.

$\overline{\text{SS}_0}$. *Static Select* (Output, active Low). This output is used to select a static memory device (typically a ROM). It is enabled by default but can be disabled under program control.

$\overline{\text{WAIT}}$. *Wait* (Bidirectional, Active Low, Open-drain). This pin serves to provide a wait output to the Z80 CPU as determined by the on-chip wait state generators of the Z80 GLU. It also serves as an input pin to determine if any external logic has caused wait states.

XTALI. *Crystal In* (Input, active High). This input can be connected to either a parallel resonant crystal, a ceramic resonator, or an external clock source. A fundamental parallel-type crystal is recommended.

XTALO. *Crystal Out* (Output). This output can be connected to either a parallel-resonant crystal or a ceramic resonantor. If XTALI is connected to a clock source, then this pin should be left OPEN.

$\overline{\text{ZRD}}$. *Z8500 Read* (Output, active Low). This pin is multiplexed with the $\overline{\text{HALT}}$ input. When the Power-Down Interface is not enabled (the Z8500 peripheral interface is enabled), this pin is the $\overline{\text{ZRD}}$ output from the device. This signal is used for the Z8500 peripherals during I/O read cycles and during the reset cycle.

$\overline{\text{ZWR}}$. *Z8500 Write* (Output, active Low). This pin is multiplexed with the $\overline{\text{INT}}$ line. When the Power-Down Interface is not enabled (the Z8500 peripheral interface is enabled), this pin is the $\overline{\text{ZWR}}$ output from the device. This signal is used for the Z8500 peripherals during I/O write cycles and during the reset cycle.

ARCHITECTURE:

Clock Oscillator. The clock oscillator circuit (see Figure 3) consists of three parts; the oscillator, the controller, and the driver. The oscillator circuit can accept either a parallel resonant crystal, a ceramic resonator, or a TTL compatible clock input for the main clock generation. The oscillator frequency is twice that of the CLOCK output (system operating frequency) and is rated to a maximum frequency of 12MHz.

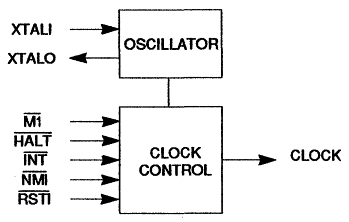


Figure 3: Clock Oscillator Block

The controller circuit performs two monitoring functions. First, it monitors the $\overline{\text{M1}}$ and $\overline{\text{HALT}}$ outputs from the Z80 CMOS CPU in order to provide control over the CLOCK output for entry into the power-down mode. Second, it also monitors the $\overline{\text{RSTI}}$, $\overline{\text{NMI}}$, and $\overline{\text{INT}}$ signals to provide control over the CLOCK output for exit from the power-down mode. This feature is enabled through a strap option with the PD pin of the device and is multiplexed with the Z8500 peripheral interface allowing utilization with only one of the two features.

The driver circuit provides a clock output with the necessary AC and DC characteristics to satisfy both the NMOS and CMOS Z80 CPUs. It also provides the clock signal with enough drive to connect directly to several peripherals.

Static Memory Interface. The Z80 GLU provides logic (see Figure 4) for a static memory (RAM or ROM) chip select output ($\overline{SS_0}$) that is based upon the address inputs to the device. A part of the upper byte of the address bus (A_{15}-A_{12}) is compared against the contents of the Static Memory Boundary Registers. If the comparison yields a result that is less than or equal to the programmed address value, then the chip select signal is enabled for the duration of the memory access cycle. Boundaries are written in 4K segments (address lines A_{15}-A_{12}).

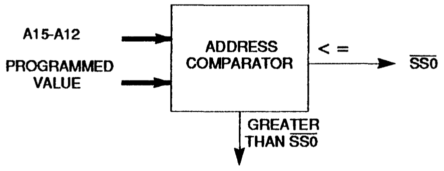


Figure 4: Static Memory Select Block

The default condition of the Static Memory Boundary Register after reset is to indicate that all memory is static (contents = 0FFH) and that the $\overline{SS_0}$ output is enabled. The lower limit of the static memory area addressed by $\overline{SS_0}$ is always assumed to be 0000H.

A wait state generator is available to insert from 0 to 3 additional wait states into access cycles for all static accesses (for interfacing with slow memory devices). For more detail, refer to the section on the wait state generators.

Dynamic Memory Interface Controller. This logic (see Figure 5) provides all the necessary control circuitry to interface directly to 64K DRAM memories. It provides all the required timing to generate $\overline{RAS}$ and $\overline{CAS}$ signals for all memory accesses not decoded as being for a static memory area and handles the timing required for $\overline{RAS}$ precharge time. It also provides for $\overline{RAS}$ only refresh of dynamic memories with support for all 64K devices. This is accomplished by an 8-bit counter on-chip that provides the refresh address instead of accepting it from the Z80 CPU.

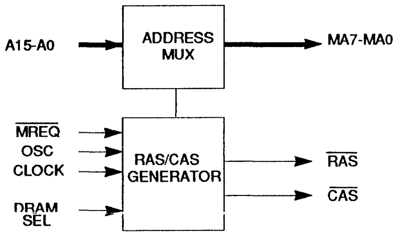


Figure 5: DRAM Controller Block

Memory and I/O Chip Selects. This section (see Figure 6) allows up to eight individual chip selects to be available to the user by programming the $\overline{CS_7}$-$\overline{CS_0}$ outputs to respond to either memory or I/O addresses. Each chip select has a separately programmable address range and can be enabled or disabled via software. The software would program the required Chip Select Control Register (CSCR7-CSCR0).

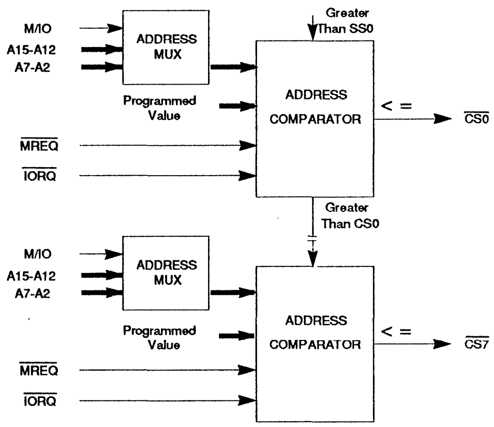


Figure 6: Chip Selects Control Block

When programmed as a memory chip select, values for address lines A_{15}-A_{12} are stored into the respective register. These values are used in comparison against the actual state of the address bus during the memory access cycle. If the comparison yields a result that is less than or equal to the programmed address value, then the chip select signal is enabled for the duration of the memory access cycle. Since address lines A_{15}-A_{12} are used, memory chip selects must be programmed in multiples of 4K bytes.

When programmed as an I/O chip select, values for address lines A7-A2 (A7-A4 for $\overline{CS7}$-$\overline{CS5}$) are stored into the respective register. These values are used in comparison against the actual state of the address bus during the I/O access cycle. When a comparison yields a result that is equal to the stored value, the appropriate chip select output is made active for the duration of the cycle. Since address lines A7-A2 are used, I/O chip selects can be programmed only for 4 contiguous port addresses (except $\overline{CS7}$-$\overline{CS5}$, which are programmed for 16 contiguous addresses).

Since I/O address comparisons are made on an "equal to" basis, all I/O selects should be programmed after any memory selections. The default condition after reset is with all chip select registers equal to 00H (addresses unspecified and outputs disabled).

Reset Synchronization Logic. The reset logic (see Figure 7) performs two functions; providing a power-on reset pulse that insures proper reset initialization and synchronization of an external reset request. The power-on reset circuit holds the $\overline{RSTO}$ output active for at least 16 clock cycles after the power has stabilized and the oscillator is running. The synchronization logic allows the $\overline{RSTI}$ input to be synchronized with $\overline{M1}$ and CLOCK signals to inhibit erroneous memory writes while entering a reset sequence. In this case, the $\overline{RSTO}$ output is held active for a period of 16 CLOCK cycles.

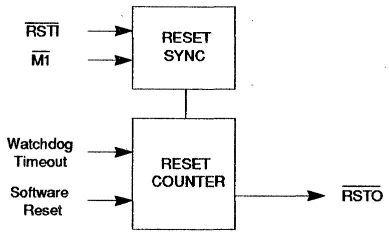


Figure 7: Reset Control Block

Watchdog Timer. The watchdog timer circuit (see Figure 8) consists of an 8-bit counter configured as a modulo 256 prescaler circuit, and a 16-bit count-down counter. The 16-bit count-down counter can be programmed for the time-out count that the user desires. If the 16-bit count-down timer is allowed to reach a count of zero, then the logic will issue an internal reset to the Reset Synchronization Logic and reset the entire system. The Watchdog Timer is kept from reaching a zero count by reloading the count before it reaches zero. This feature can be enabled or disabled via program control with the default condition after reset is disabled.

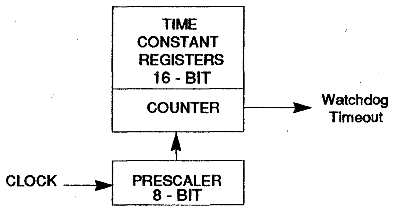


Figure 8: Watchdog Timer Block

Wait State Generators. There are five different wait state generators (see Figure 9) available to the user; one for static memory access cycles, one for I/O cycles (addresses 00H-03FH only), one of RETI cycle stretching, one for the interrupt acknowledge cycle stretching, and one for the opcode fetch cycle. The first three wait state generators are capable of adding from 0 to 3 additional wait states to their respective machine cycles. The fourth wait state generator (interrupt acknowledge) adds a varied amount of wait states depending upon whether or not the Z8500 peripheral interface is enabled. The opcode fetch wait state generator will add only one wait state. Each wait state generator is seperately programmable and can be enabled or disabled under program control. The default state after reset is with all wait state generators enabled for the maximum number of wait states.

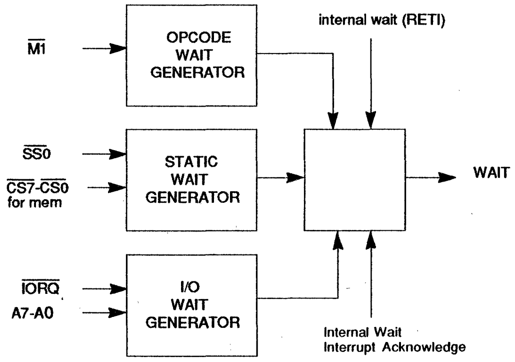


Figure 9: Wait State Generator Block

Interrupt Acknowledge Stretching. This logic unit (see Figure 10) monitors the $\overline{M1}$, $\overline{MREQ}$, and $\overline{IORQ}$ signals from the Z80 CPU to determine when an interrupt acknowledge cycle is being executed. When a long interrupt daisy chain settle time by using one of the on-chip wait state generators to insert additional wait states (beyond the two automatic wait states already inserted by the CPU) into the interrupt acknowledge cycle. If addtional wait states are inserted , then the $\overline{IORQ}$ signal is delayed by those number of wait states before being output to the peripherals (as $\overline{PIORQ}$).

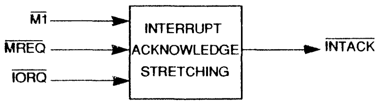


Figure 10: Interrupt Acknowledge Stretching

Z8500 Peripheral Interface. This logic (see Figure 11) provides the signals necessary to interface the Z80 CPU to the Z8500 peripheral family. The logic controls the read ($\overline{ZRD}$), write ($\overline{ZWR}$), and $\overline{INTACK}$ signals to the Z8500 peripherals and provides the necessary wait states for the Z80 CPU to achieve this interface. During the interrupt acknowledge cycle, two additional wait states are automatically inserted (beyond the two automatic wait states already inserted by the CPU) if this feature is enabled. To allow customization of the interrupt daisy chain, the interrupt acknowledge wait state generator can be used to provide up to four additional wait states beyond those already inserted.

This logic also controls the $\overline{ZRD}$ and $\overline{ZWR}$ lines for reset control of the Z8500 peripherals. This feature is multiplexed with the power-down clock control mode and controlled with the PD pin.

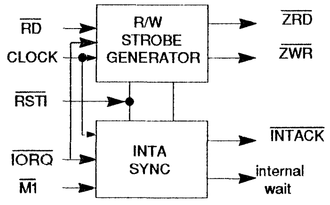


Figure 11: Z8500 Peripheral Interface Block

RETI Stretching. This logic (see Figure 12) monitors the contents of the data bus during the fetching of an instruction. If the instruction fetched is one of the special "ED" opcodes, then up to three additional wait states can be inserted into the next machine cycle. This feature allows for extension of the trailing end of the Z80 interrupt daisy chain (very useful for peripherals decoding the RETI instruction sequence).

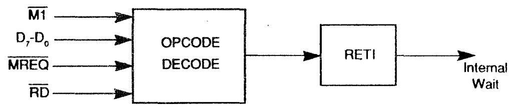


Figure 12: RETI Stretching

General Purpose Outputs. There are two general purpose outputs available to the user. These outputs are multiplexed with either the $\overline{PIORQ}$ output, or the $\overline{RAS}$ and $\overline{CAS}$ outputs. These outputs are controlled via software.

Reg.	Address	Name	Description	Reg.	Address	Name	Description
R0	0F0H	MCR	Master Control	R8	0F8H	CSCR5	Chip Select 5 Control
R1	0F1H	DMCR	Device Mode Control	R9	0F9H	CSCR6	Chip Select 6 Control
R2	0F2H	SMCR	Static Select 0 Control	R10	0FAH	CSCR7	Chip Select 7 Control
R3	0F3H	CSCR0	Chip Select 0 Control	R11	0FBH	WDTC1	Watchdog Time Constant 1
R4	0F4H	CSCR1	Chip Select 1 Control	R12	0FCH	WDTC0	Watchdog Time Constant 0
R5	0F5H	CSCR2	Chip Select 2 Control	R13	0FDH	WSCR	Wait State Control
R6	0F6H	CSCR3	Chip Select 3 Control	R14	0FEH	GPCR	General Purpose Control
R7	0F7H	CSCR4	Chip Select 4 Control	R15	0FFH		Reserved

Figure 13: Registers

PROGRAMMING:

The Z80 GLU device has 15 internal write registers for programming options and control (the 16th register is reserved for future use). Each register is addressed individually with the port addresses residing at locations 0F0H through 0FFH. All bits referenced as "Reserved" must be programmed as "0".

Port Address 0F0H - Register 0 - Master Control Register (see Figure 14). Master Control Register provides some global control functions for the Z80 GLU. The "Reset" bit allows a software reset for the chip. All registers are initialized to their default state. The D1 bit is used to reload the Watchdog Time Constant Registers (Registers 11 and 12) into the Watchdog Counter. This bit should be set periodically to make sure that the Watchdog Timer does not expire and reset the system. The D7 bit is the "Master Enable" for the Z80 GLU. It is a good practice to program all registers before setting this bit. This register is preset to 02H on reset.

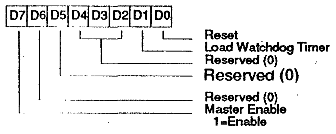


Figure 14: Master Control Register

Port Address 0F1H - Register 1 - Device Mode Control (see Figure 15). This register controls the functional options of the Z80 GLU devices. The DRAM Interface, Chip Selects, General Purpose Outputs, and Watchdog Timer are controlled by bit settings within this register. This register is present to all 0's on reset to configure the Z80 GLU device with the DRAM interface disabled and general purpose outputs enabled on the $\overline{CAS}$ and $\overline{RAS}$ outputs. The Watchdog Timer is also disabled on reset.

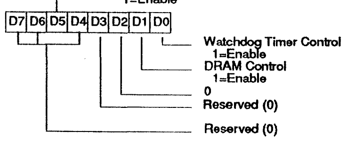


Figure 15: Device Mode Control Register

Port Address 0F2H - Register 2 - Static Select Memory Control ($\overline{SS_0}$, Figure 16). This register not only controls the $\overline{SS_0}$ output, but also allows the user to program the memory range to which this output should respond. Memory ranges are programmed on 4K boundaries with a logical comparison of less than or equal to the programmed address being made. The lower boundary for this comparison is always assumed to be 0000H. This register is preset to all 1's on reset to let the $\overline{SS_0}$ output respond to all memory addresses.

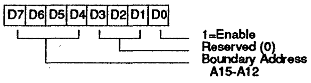


Figure 16: Static Select Memory Control Register

Port Addresses 0F3H-0FAH - Register 3-10 - Chip Select Control 0 - Chip Select Control 7 (see Figure 17). These registers control the additional chip select outputs from the Z80 GLU. The outputs can be programmed to respond to either memory or I/O device addresses. Each output is individually programmed and can be enabled or disabled under program control. These registers are preset to all 0's on reset to ensure that all the Chip Select outputs are inactive and disabled.

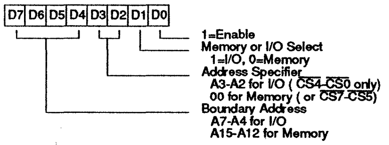


Figure 17: Chip Select Control Register

When programmed for memory devices, address line values for A_{15}-A_{12} should be programmed here. Comparisons on memory addresses are made for less than or equal to the contents of the current register and greater than the contents of the previous register ($\overline{SS_0}$ for $\overline{CS_0}$). When programmed for I/O devices, address line values for A_7-A_2 should be programmed here. Chip Selects 0 through 4 respond to four consecutive I/O port addresses while Chip Selects 5 through 7 respond to 16 consecutive I/O addresses (this allows direct interface devices such as the Z84C90 KIO). Comparisons on I/O addresses are made equal to the contents of this register. When programming the Chip Select Control Registers, it is advisable to disable the GLU (using bit 7 of Register 0) and re-enable it afterwards.

PROGRAMMING (cont):

Port Address 0FBH - Register 11 - Watchdog Time Constant 1 (see Figure 18). This register is used to hold the most significant byte of the time constant for the watchdog timer. The enable control for the watchdog timer is in the Device Mode Control Register (Register 1). This register is preset to all 0's on reset to indicate a maximum count.

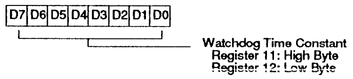


Figure 18: Watchdog Time Constant Register

Port Address 0FCH - Register 12 - Watchdog Time Constant 0 (see Figure 18). This register is used to hold the least significant byte of the time constant for the watchdog timer. The enable control for the watchdog timer is in the Device Mode Control Register (Register 1). This register is preset to all 0's on reset to indicate a maximum count.

Port Address 0FDH - Register 13 - Wait State Control (see Figure 19). Programming this register provides the user with control over the individual wait state generators of the Z80 GLU. Each of the first three wait state generators can add between 0 and 3 additional wait states into the respective machine cycles. The fourth wait state generator adds 2 to 4 additional wait states. Programming 0's into a wait state generator is the same as disabling that function. This register is preset to all 1's to add the maximum number of wait states into the respective cycles until the user can program the Z80 GLU. If the Z8500 peripheral interface is selected, then two additional wait states are automatically added to the interrrupt acknowledge cycle (allowing up to 6 wait states to be added).

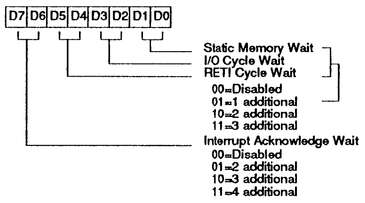


Figure 19: Wait State Control Register

Port Address 0FEH - Register 14 - General Purpose Control (see Figure 20). This register is used to provide the user with control over the general purpose outputs.

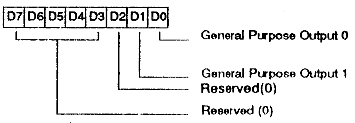


Figure 20: General Purpose Control Register

This register is preset to 03H on reset to ensure that the general purpose outputs are held high (for a possible DRAM interface). When DRAM interface is disabled, the programmed value in bit 0 (GP0) is output on pin 39 as well as pin 58.

TIMING:

The following timing diagrams show typical timing relationships when the Z84C80 is in a Z80 CPU environment. Together with the AC Characteristics, they also describe how the devices will behave when interfaced with other CPUs. For more detailed information, please refer to the AC Characteristics section. For more detailed information about the timing of the Z80 CPU, please refer to the Z80 Product Specification and Technical Manual.

I/O Cycle. Figure 21 illustrates the timing for I/O read and write cycles; this includes the programming of the Z80 GLU and other devices. The Z80 GLU does not receive a specific write signal; it internally generates its own by decoding an I/O operation (corresponding to I/O addresses 0F0H through 0FFH) with the absence of a read signal.

All chip select outputs programmed for operation with I/O devices are affected whenever the $\overline{IORQ}$ signal becomes active. The Z80 GLU will compare the contents of the lower half of the address bus (A_7-A_0) against pre-programmed values. If the compared address is the same as the programmed range, then the appropriate chip select is made active and will remain active until the $\overline{IORQ}$ signal becomes inactive. Chip Selects 7 through 5 respond to 16 consecutive I/O addresses (for interface with high port consumption devices such as the Z84C90 KIO) while Chip Selects 4 through 0 respond to 4 consecutive I/O addresses. Additional wait states can be placed into I/O cycles using the lower 64 I/O addresses (00H through 3FH) by utilizing the on-chip I/O wait state generator.

During the I/O cycles, the Z84C80 has the ability to provide either the Read ($\overline{ZRD}$) or Write ($\overline{ZWR}$) signal to a Z8500 peripheral device. The $\overline{ZWR}$ signal is generated with timing guaranteed to ensure that the data is valid before and during the entire time that the write signal is active.

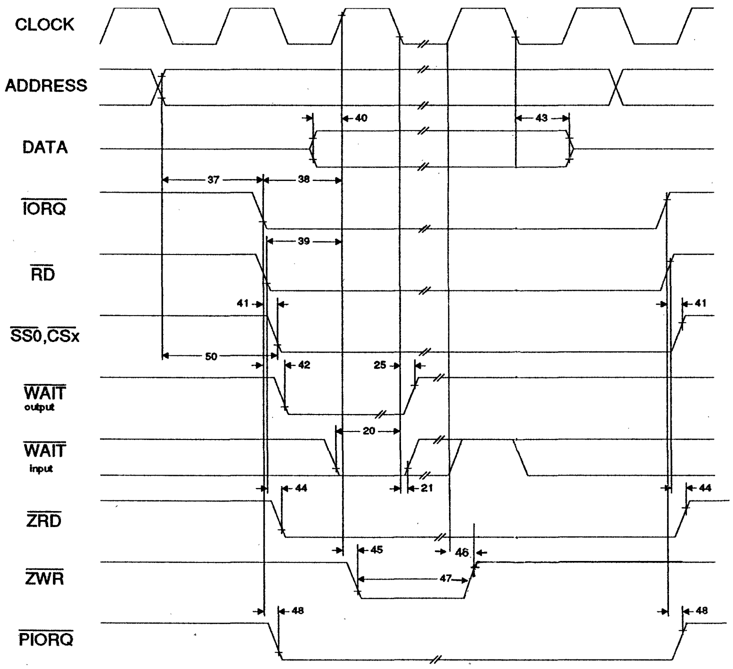


Figure 21: I/O Cycle Timing

Opcode Fetch Cycle. Figure 22 illustrates the sequence of events during an opcode fetch cycle. Several events can occur within the Z80 GLU during the opcode fetch cycle. Since all opcode fetch cycles are accesses to memory, either the static or dynamic memory interface can be activated. The decision for which interface is done by comparing the contents of the address bus at the beginning of the cycle. All memory addresses not corresponding to static memory interface (either $\overline{SS_0}$ or $\overline{CS_7}$-$\overline{CS_0}$) are considered to be dynamic in nature.

All chip select outputs programmed for operation with memory devices are affected whenever the $\overline{MREQ}$ signal becomes active. The Z80 GLU will compare the contents of the upper half of the address bus (A_{15}-A_8) against preprogrammed values. If the compared address falls within the programmed range, then the appropriate chip select is made active and will remain active until the $\overline{MREQ}$ signal becomes inactive.

The dynamic memory interface controller will assert the $\overline{RAS}$ signal as a result of the rising edge of T_2. This ensures that all addresses are stable and that the proper memory address decoding has occurred. If the decoded address does not indicate a selection for static memory, then the $\overline{CAS}$ signal is generated on the falling edge of T_2 to indicate that the contents of the multiplexed address bus now contain the column address.

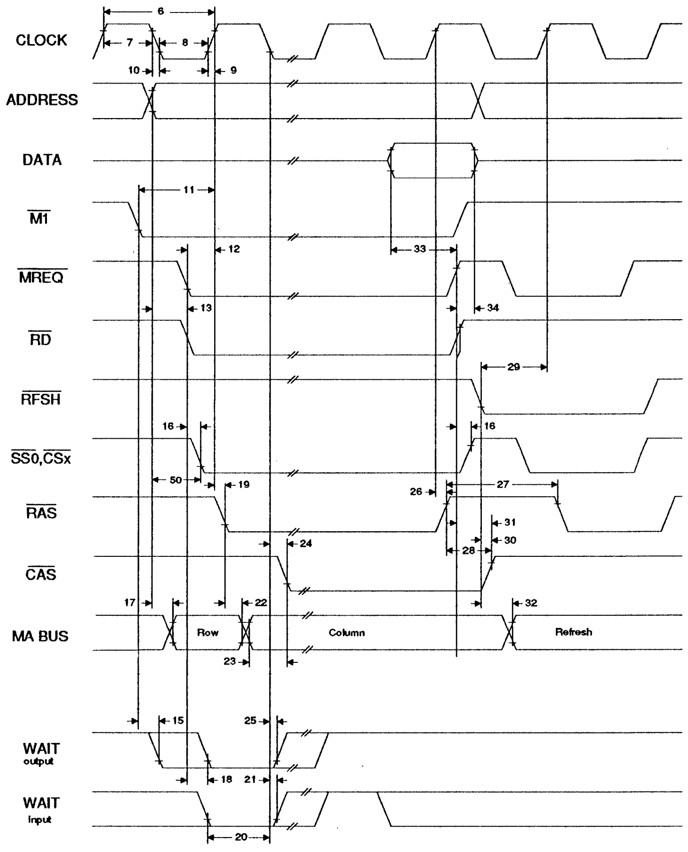


Figure 22: Opcode Fetch Cycle Timing

A third fucnction that can occur during the opcode fetch cycle is that of the RETI decode sequence. During the opcode fetch cycle, the Z80 GLU decodes the special "ED" instruction sequence of the Z80 CPU. If the "ED" opcode is detected, then the Z80 GLU automatically inserts one additional wait state into the next opcode fetch cycle. This provides the Z80 GLU with enough time to decode the following byte of the instruction to determine if it is the "4D" part of the RETI sequence. If the second byte is not a "4D," then the $\overline{\text{WAIT}}$ line is released and the CPU is allowed to continue its process. It the second byte is the "4D" part of the RETI sequence, then the logic can insert up to 2 more wait states to provide a longer interrupt daisy-chain settle time for the RETI instruction sequence.

Additional wait states can be inserted into the opcode fetch cycle from several sources. The opcode fetch wait state generator can insert one additional wait state so that memory access time can be defined by the normal memory cycle rather than the opcode fetch cycle. The static wait state generator can add wait states whenever one of the chip select outputs are active as a result of a requested memory address. The RETI wait state generator also adds wait states as previously described.

Memory Access Cycles. Figure 23 illustrates the timing for a normal memory access cycle. During the normal memory access cycle only one of two memory interfaces can be enabled, static or dynamic. The operation of these two memory interfaces are the same as during the opcode fetch cycle. Wait states can also be added here for static memory access (for use with slow memories).

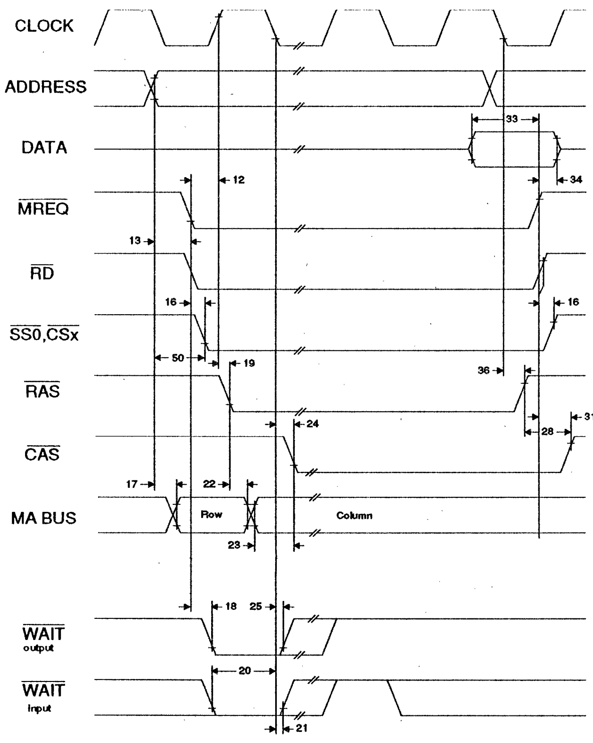


Figure 23: Memory Access Cycle Timing

Interrupt Acknowledge Cycles. Figure 24 depicts the timing sequence for a Z80 interrupt acknowledge cycle. The Z84C80 GLU device provides the user with the capability of "stretching" the interrupt acknowledge cycle of the Z80 CPU by adding wait states and delaying the $\overline{\text{IORQ}}$ (actually the $\overline{\text{PIORQ}}$ output) to the Z80 family of peripherals. This allows the user to have more interrupting peripherals within his system by lengthening his daisy chain settle time. The Z84C80 GLU also "stretches" the RETI decode time to help facilitate a long interrupt daisy chain (see Opcode Fetch Cycle description). During the interrupt acknowledge cycle, the Z84C80 has the ability to provide the proper timing for the interrupt acknowledge ($\overline{\text{INTACK}}$) and read ($\overline{\text{ZRD}}$) signals for Z8500 peripherals.

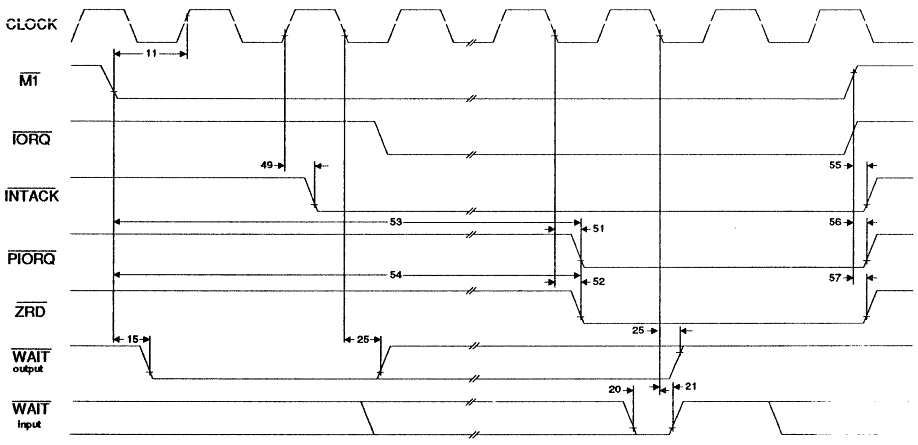


Figure 24: Interrupt Acknowledge Cycle Timing

Z84C00 CPU Power-Down Control. The Z84C80 GLU device can also provide control over the system clock for interface with the Z80 CMOS CPU and its peripheral family in order to facilitate the stand-by power feature of those devices. The power-down mode of operation is entered by the Z84C00 CPU during the T_4 low time of a $\overline{\text{HALT}}$ acknowledge cycle. Since the Z84C00 does not decode this by itself, additional logic is placed within the GLU chip to decode this machine cycle and stop the clock at the proper instant in time. This mode of operation is exited when the logic detects either an interrupt (maskable or non-maskable) or a reset to the system. The logic then releases the clock output and the CPU is allowed to continue its processing flow. This feature is multiplexed with the Z8500 peripheral interface (only one of the two can be enabled at any given time).

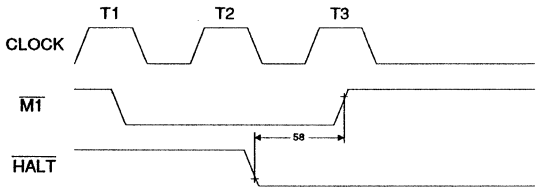


Figure 25: Power-Down Mode Entrance Timing

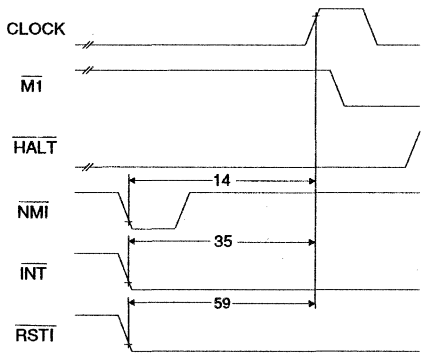


Figure 26: Power-Down Mode Exit Timing

Reset Cycle. There are three different reset cycles (see Figure 27) that are controlled by the Z80 GLU devices; power-on, external, and watchdog timer. During a power-on reset the Z80 GLU ensures that the reset output ($\overline{\text{RSTO}}$) is held low until the power supply has stabilized and the oscillator is properly operating. The $\overline{\text{RSTO}}$ signal is then released and the system is allowed to continue. If the Z8500 peripheral interface is enabled then the $\overline{\text{ZRD}}$ and $\overline{\text{ZWR}}$ signals are also held active to ensure a proper reset of the peripherals. When an external reset is required, the logic synchronizes the reset input ($\overline{\text{RSTI}}$) with the $\overline{\text{M1}}$ input signal to inhibit a possible reset during cycles which may cause erroneous writes to memory or I/O devices. If the system is in the power-down mode of operation, the reset input will release the system clock and then synchronize the input to perform the function. In either case, the $\overline{\text{RSTO}}$ signal is held active for 16 clock cycles to ensure a proper reset. If the watchdog timer is enabled and is also allowed to reach a zero count, then a reset cycle will be generated to bring the system back into a known state. Control over the $\overline{\text{ZRD}}$ and $\overline{\text{ZWR}}$ signals is also performed during a reset operation to ensure that both signals are active at the same time (the reset condition for this peripheral family).

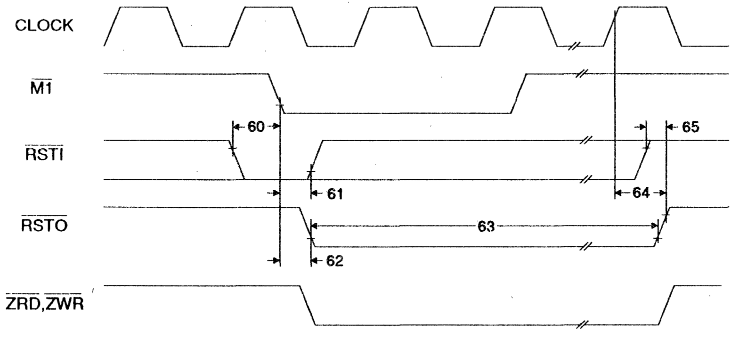


Figure 27: Reset Synchronization

ABSOLUTE MAXIMUM RATINGS:

Voltage on Vcc with respect to Vss
. . . . -0.3V to +7.0V
Voltages on all inputs with respect to Vss
. . . . -0.3V to Vcc+0.3V
Operating Ambient Temperature
. . . . See Ordering Information
Storage Temperature
. . . . -65 C to +150 C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS:

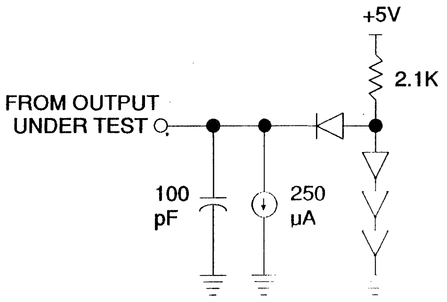
The DC Characteristics and Capacitance sections below apply to the following standard test conditions, unless otherwise noted. All voltages are referenced to GND (0V). Positive current flows into the referenced pin.

Available operating temperature ranges are:
- S = 0 C to +70 C

Voltage Supply Range: +5.0V +/- 10%

All AC parameters assume a load capaitance of 100 pF. Add 10 ns delay for each 50 pF increase in load up to a maximum of 200 pF for the data bus and 100 pF for the address and control lines. AC timing measurements are referenced to 1.5 volts (except for CLOCK, which is referenced to the 10% and 90% points).

The Ordering Information section lists temperature ranges and product numbers. Package drawings are in the Package Information section. Refer to the Literature List for additional documentation.



DC CHARACTERISTICS:

Symbol	Parameter	min	max	Unit	Condition
V_{IHC}	Input Clock High Voltage	2.2	V_{CC}+0.3	V	Driven by Ext. Clock
V_{ILC}	Input Clock Low Voltage	-0.3	.8	V	Driven by Ext. Clock
V_{IH}	Input High Voltage	2.2	V_{CC}+0.3	V	
V_{IL}	Input Low Votlage	-0.3	0.8	V	
V_{OHC}	Output Clock High Voltage	V_{CC}-0.6		V	
V_{OH}	Output High Voltage	2.4			I_{OH}=-250 µA
V_{OL}	Output Low Voltage		0.4	V	I_{OL}=2.0 mA
V_{OLW}	Output Low Voltage (Wait)		0.5	V	I_{OL}=5.0 mA
I_{IL}	Input Leakage Current		±10	µA	
I_{CC}	Power Supply Current		30	mA	f=8.0 MHz
			40	mA	f=10.0 MHz V_{CC}=5V V_{IH}=V_{CC}-0.2V V_{IL}=0.2V
C_I	Input Capacitance		5	pF	
C_O	Output Capacitance		10	pF	

AC CHARACTERISTICS:

No.	Symbol	Parameter	Z84C8006		
			min	max	notes
1	TcXTAL	XTAL Cycle Time	81		
2	TwXTALh	XTAL High Width	25		
3	TwXTALl	XTAL Low Width	25		
4	TrXTAL	XTAL Rise Time		15	
5	TfXTAL	XTAL Fall Time		15	
6	TcC	CLOCK Cycle Time	160		
7	TwCh	CLOCK High Width	65		
8	TwCl	CLOCK Low Width	65		
9	TrC	CLOCK Rise Time		15	
10	TfC	CLOCK Fall Time		15	
11	TsM1f(Cr)	$\overline{M1}$ ↓ to CLOCK ↑ Setup	20		
12	TsMREQf(Cr)	$\overline{MREQ}$ ↓ to CLOCK ↑ Setup	20		
13	TsA(MREQf)	Address to $\overline{MREQ}$ ↓ Setup	50		
14	TdNMIf(Cr)	$\overline{NMI}$ ↓ to CLOCK ↑ Delay		2*T_CC	
15	TdM1f(WAITf)	$\overline{M1}$ ↓ to $\overline{WAIT}$ ↓ Delay		55	note 1
16	TdMREQ(CS)	$\overline{MREQ}$ to SS_0,CS_x Delay		50	
17	TdA(RA)	Address to Row Address Delay		50	
18	TdMREQf(WAITf)	$\overline{MREQ}$ ↓ to $\overline{WAIT}$ ↓ Delay		55	note 2
19	TdCr(RAS)	CLOCK ↑ to $\overline{RAS}$ ↓ Delay		50	
20	TsWAIT(Cf)	$\overline{WAIT}$ to CLOCK ↓ Setup	25		
21	ThCf(WAIT)	CLOCK ↓ to $\overline{WAIT}$ Hold	10		
22	ThRASf(RA)	$\overline{RAS}$ ↓ to Row Address Hold	25		
23	TdCA(CASf)	Column Address to $\overline{CAS}$ ↓ Delay	25		
24	TdCf(CASf)	CLOCK ↓ to $\overline{CAS}$ ↓ Delay		45	
25	TdCf(WAITr)	CLOCK ↓ to $\overline{WAIT}$ ↑ Delay		RC+25	note 3
26	TdCr(RASr)	CLOCK ↑ to $\overline{RAS}$ ↑ Delay		45	
27	TdRASr(RASf)	$\overline{RAS}$ ↑ to $\overline{RAS}$ ↓ Delay	TcC		
28	TdRASr(CASr)	$\overline{RAS}$ ↑ to $\overline{CAS}$ ↑ Delay		45	
29	TsRFSHf(Cr)	$\overline{RFSH}$ ↓ to CLOCK ↑ Setup	15		
30	TdRFSHf(CASr)	$\overline{RFSH}$ ↓ to $\overline{CAS}$ ↑ Delay		45	
31	TdMREQr(CASr)	$\overline{MREQ}$ ↑ to $\overline{CAS}$ ↑ Delay		45	
32	TdRFSHf(RFA)	$\overline{RFSH}$ ↓ to Refresh Address Delay		45	
33	TsD(MRD)	Data to $\overline{MREQ}$ or $\overline{RD}$ ↑ Setup	15		
34	ThMRD(D)	$\overline{MREQ}$ or $\overline{RD}$ ↑ to Data Hold	10		
35	TdINTf(Cr)	$\overline{INT}$ ↓ to CLOCK ↑ Delay		2*T_CC	
36	TdCf(RASr)	CLOCK ↓ to $\overline{RAS}$ ↑ Delay		50	
37	TsA(IORQf)	Address to $\overline{IORQ}$ ↓ Setup	40		
38	TsIORQf(Cr)	$\overline{IORQ}$ ↓ to CLOCK ↑ Setup	20		
39	TsRD(Cr)	$\overline{RD}$ to CLOCK ↑ Setup	20		
40	TsD(Cr)	Write Data to CLOCK ↑ Setup	30		
41	TdIORQ(CS)	$\overline{IORQ}$ to CS_x Delay		50	
42	TdIORQf(WAITf)	$\overline{IORQ}$ ↓ to WAIT ↓ Delay		55	
43	ThCf(D)	CLOCK ↓ to Data Hold	10		
44	TdRD(ZRD)	$\overline{RD}$ to $\overline{ZRD}$ Delay		50	
45	TdCr(ZWRf)	CLOCK ↑ to $\overline{ZWR}$ ↓ Delay		50	
46	TdCf(ZWRr)	CLOCK ↑ to $\overline{ZWR}$ ↑ Delay		50	
47	TwZWRl	$\overline{ZWR}$ Pulse Width Low	1.0*TcC		
48	TdIORQ(PIORQ)	$\overline{IORQ}$ to $\overline{PIORQ}$ Delay		40	note 4
49	TdCr(INTACKf)	CLOCK ↑ to $\overline{INTACK}$ ↓ Delay		50	
50	TdA(CS)	Address to SS_0,CS_x Delay		50	note 5
51	TdCf(PIORQf)	CLOCK ↓ to $\overline{PIORQ}$ ↓ Delay		50	
52	TdCf(ZRDf)	CLOCK ↓ to $\overline{ZRD}$ ↓ Delay		50	
53	TdM1f(PIORQf)	$\overline{M1}$ ↓ to $\overline{PIORQ}$ ↓ Delay			note 6
54	TdM1f(ZRDf)	$\overline{M1}$ ↓ to $\overline{ZRD}$ ↓ Delay			note 7
55	TdM1r(INTACKr)	$\overline{M1}$ ↑ to $\overline{INTACK}$ ↑ Delay		50	
56	TdM1r(PIORQr)	$\overline{M1}$ ↑ to $\overline{PIORQ}$ ↑ Delay		50	
57	TdM1r(ZRDr)	$\overline{M1}$ ↑ to $\overline{ZRD}$ ↑ Delay		50	
58	TsHALTf(M1r)	$\overline{HALT}$ ↓ to $\overline{M1}$ ↑ Setup	20		

AC CHARACTERISTICS (cont):

No.	Symbol	Parameter	Z84C8010	
			min	max
59	TdRSTIf(Cr)	$\overline{\text{RSTI}}$ ↓ to CLOCK ↑ Delay		2*TcC
60	TsRSTIf(M1f)	$\overline{\text{RSTI}}$ ↓ to $\overline{\text{M1}}$ ↓ Setup	20	
61	ThM1f(RSTI)	$\overline{\text{M1}}$ ↓ to $\overline{\text{RSTI}}$ Hold	10	
62	TdM1f(RSTOf)	$\overline{\text{M1}}$ ↓ to $\overline{\text{RSTO}}$ ↓ Delay		50
63	TwRSTOl	$\overline{\text{RSTO}}$ Pulse Width Low	16*TcC	
64	TdCr(RSTOr)	CLOCK ↑ to $\overline{\text{RSTO}}$ ↑ Delay		50
65	TdRSTIr(RSTOr)	$\overline{\text{RSTI}}$ ↑ to $\overline{\text{RSTO}}$ ↑ Delay		50

All parameters in units of nanoseconds unless specified.

Notes:
1. This parameter is valid only when the opcode wait state generator is enabled and interrupt acknowledge cycles.
2. This parameter is valid only when the static memory wait state genereator is enabled.
3. This parameter value is dependent upon the RC time constant as determined by the external pull-up resistor.
4. For I/O cycles and non-stretched interrupt acknowledge cycles.
5. This parameter is valid only if parameters 13 and 37 are not met.
6. Z8500 Peipheral Interface disabled; 2*TcC+TwCh+TfC+n*Tw-80 where n=0,2,3, or 4.
   Z8500 Peripheral Interface enabled; 4*TcC+TwCh+TfC+n*Tw-80 where n=1,2,3, or 4.
7. 4*TcC+TwCh+TfC+n*Tw-80 where n=1,2,3, or 4.
8. There is an error with the $\overline{\text{M1}}$ signal which may cause the DRAM interface to function improperly during a refresh cycle. To correct this just add a 200-500 ohm resistor in series with the $\overline{\text{M1}}$ signal from the Z80 CPU.
9. The $\overline{\text{ZWR}}$ signal from the Z84C80 is one clock cycle wide. The 8500 peripherals expect this signal to be one and a half clock cycles wide. For example, the Z8530 specification for $\overline{\text{ZWR}}$ at 6 MHz is 250 n sec.. The $\overline{\text{ZWR}}$ signal pulse width from the 84C80 at 6 MHz is 166.6 n sec.. To work-around this problem, one possibility is to run the 8500 peripheral at a higher speed than the GLU. For example, use a 6 MHz 8530, and run the GLU at 4 MHz.

Zilog

Product Specification

January 1989

Z84C90 CMOS Z80®KIO Serial/Parallel/Counter/Timer

FEATURES:

- Two independent synchronous/asynchronous serial channels.
- Three 8-bit parallel ports.
- Four independent counter/timer channels.
- On-chip clock oscillator/driver.
- Software/Hardware Resets.
- Designed in CMOS for low power operations.
- Supports Z80 Family interrupt daisy chain.
- Programmable interrupt priorities.
- 8MHz bus clock frequency.
- Single +5 Volt Power Supply.

GENERAL DESCRIPTION:

Zilog's new Z84C90 Serial/Parallel/Counter/Timer (KIO) is a multi-channel, multi-purpose I/O device designed to provide the end-user with a cost effective and powerful solution to meet his perophereal needs. The Z84C90 combines the features of one Z84C30 CTC, one Z84C4xSIO, one Z84C20 PIO, a byte-wide bit-programmable I/O port, and a crystal oscillator into a single 84 pin PLCC package. The block diagram for the Z84C90 is shown in Figure 1 while the pinout is shown in Figure 2. Utilizing fifteen internal registers for data and programming information, the KIO can easily be configured to any given system environment. Although the optimum performance is obtained with a Z84C00 CPU, the KIO can just as easily be used with any other CPU.

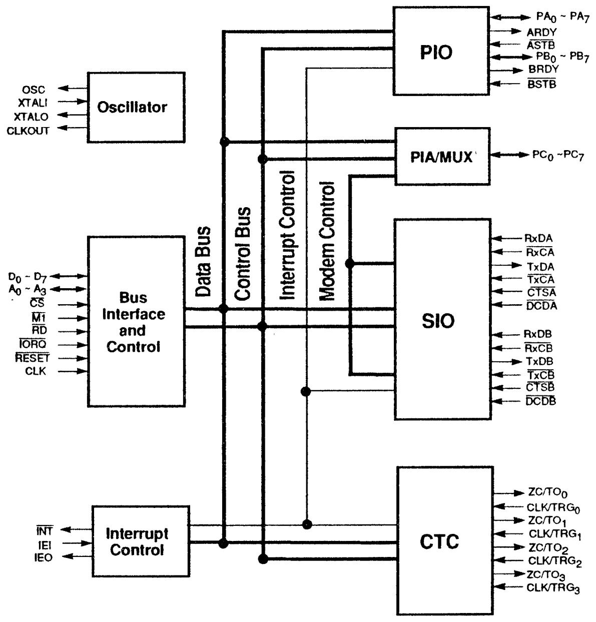


Figure 1: KIO Block Diagram

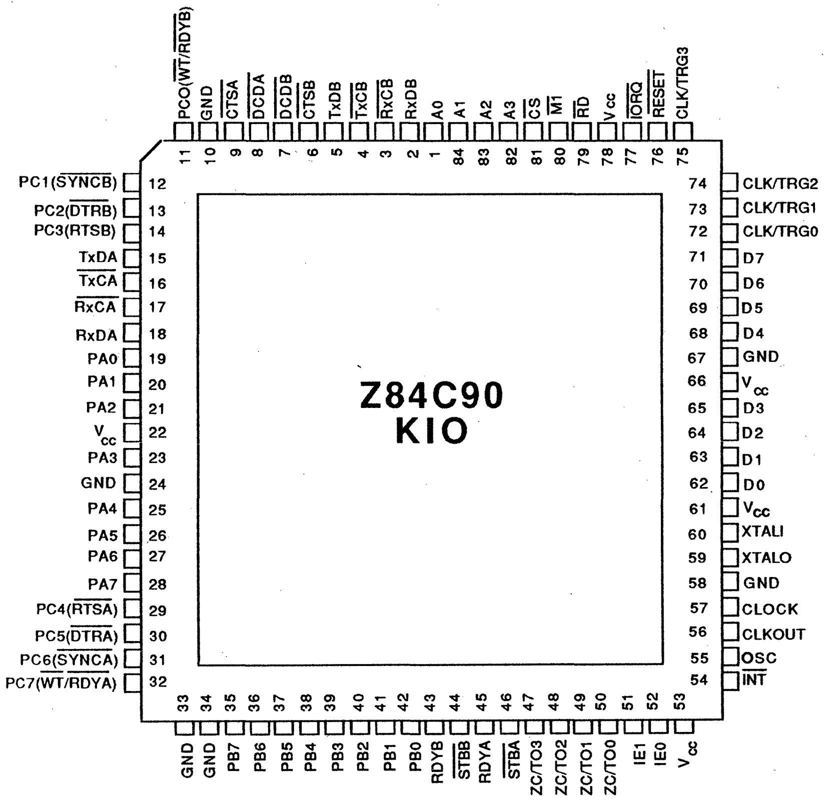


Figure 2: PLCC Pinout

Z84C20 Parallel Input/Output Logic Unit: This logic unit provides both TTL- and CMOS-compatible interfaces between peripheral devices and a CPU through the use of two 8-bit parallel ports. The CPU configures the logic to interface to a wide range of peripheral devices with no external logic. Typical devices that are compatible with this interface are keyboards, printers, and EPROM/PAL programmers.

The parallel ports (designated Port A and Port B) are byte-wide and completely compatible with the Z84C20 PIO (see Figure 3.). These two ports have several modes of operation; input, output, bidirectional, or bit control mode. Each port has two handshake signals (RDY and $\overline{STB}$) which can be used to control data transfers. The RDY (ready) indicates that the port is ready for data transfer while $\overline{STB}$ (strobe) is an input to the port that indicates when data transfer has occurred. Each of the ports can also be programmed to interrupt the CPU upon the occurrence of specified status conditions and generate unique interrupt vectors when the CPU responds. (For more information on the operation of this portion of the logic, please refer to the Z84C20 PIO Product Specification and Technical Manual.)

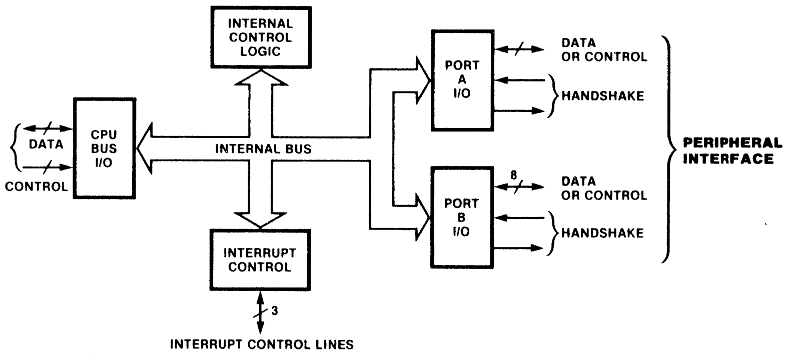


Figure 3: PIO Block Diagram

Parallel Interface Adapter (PIA) Logic Unit: This logic also offers an additional 8-bits of I/O, referred to as the PIA port (see Figure 4), to the user. This port, designated as Port C, is bit-programmable for data transfers; each bit can be individually programmed as either an input or an output. Bit direction control is accomplished through the programming of the PIA Control Register. When programmed as outputs, the output data latches are programmed with an I/O write cycle and their state can be read with an I/O read cycle. When programmed as inputs, the state of the external pin is read with the I/O read cycle. This port does not have handshake capabilities and offers no interrupt capabilities. This port is multiplexed to provide, when desired, the additional modem and CPU control signals for the serial I/O logic unit.

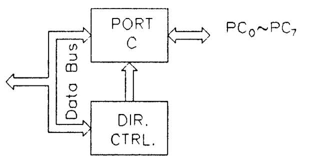


Figure 4: PIA Block Diagram

When a read from the PIA port is done, input data will be latched when $\overline{IORQ}$, $\overline{CS}$, and $\overline{RD}$ are all detected active. The data bus will display this data as a result of the rising edge of the CLOCK input after this occurrence. When a write to the PIA port is done, data will be written as a result of the rising edge of the CLOCK input after $\overline{IORQ}$ and $\overline{CS}$ have been detected active and $\overline{RD}$ has been detected inactive.

Counter/Timer Logic Unit: This logic unit provides the user with four individual 8-bit counter/timer channels that are compatible with the Z84C30 CTC (see Figure 5). The counter/timers can be programmed by the CPU for a broad range of counting and timing applications. Typical applications include event counting, interrupt and interval timing, and serial baud rate clock generation.

Each of the counter/timer channels, designated Channels 0 through 3, have an 8-bit prescaler (when used in timer mode) as well as its own 8-bit counter to provide a wide range of count resolution. Each of the channels also have their own clock/trigger input to quantify the counting process and an output to indicate zero crossing/timeout conditions. With only one interrupt vector programmed into this logic unit, each channel can generate a unique interrupt vector in response to the interrupt acknowledge cycle.

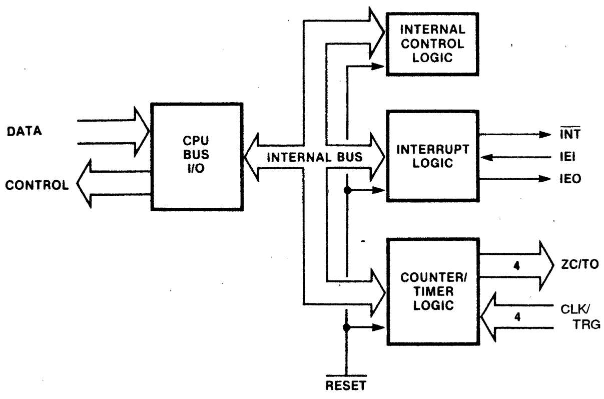


Figure 5: CTC Block Diagram

Serial I/O Logic Unit: This logic unit provides the user with two separate serial I/O channels that are completely compatible with the Z84C4x SIO (see Figure 6). Their basic functions as serial-to-parallel and parallel-to-serial converters can be programmed by a CPU for a broad range of serial communications applications. Each channel, designated Channel A and Channel B, is capable of supporting all common asynchronous and synchronous protocols (Monosync, Bisync, and SDLC/HDLC), byte- or bit-oriented.

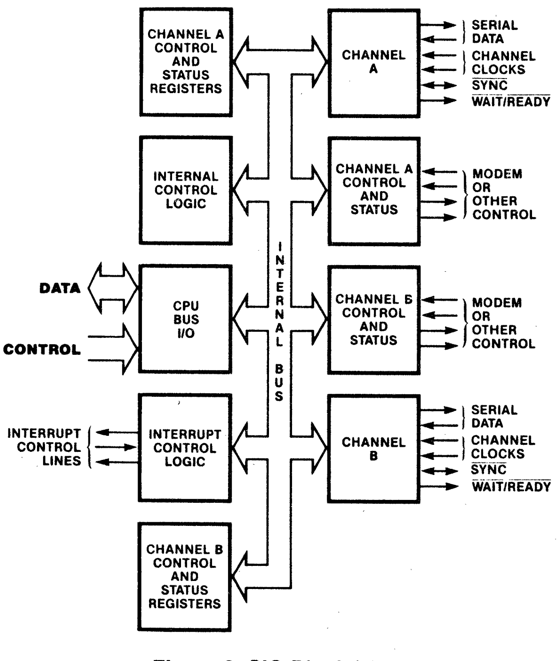


Figure 6: SIO Block Diagram

In the default state of the KIO, each serial channel supports full duplex communication with seperate transmit and receive data lines, two modem control signals ($\overline{CTS}$ and $\overline{DCD}$), and seprate transmit and receive clock inputs. Optionally, additional modem and CPU/DMA control signals can be obtained through the PIA port. (For more information on the operation of this portion of the logic, please refer to the Z84C40 SIO Product Specification and Technical Manual).

Clock Oscillator/Driver Logic Unit: A clock oscillator/driver is also available that will allow the user to eliminate that circuitry within his new design, or for use as another oscillator within the system. This logic will accept either a crystal, ceramic resonator, or TTL-compatible clock input and generate a MOS-compatible clock output and also an oscillator reference output. A fundamental parallel resonant crystal (Figure 7) is recommended. The preferred value of the two capacitors - C1 and C2 is 33 pf each.

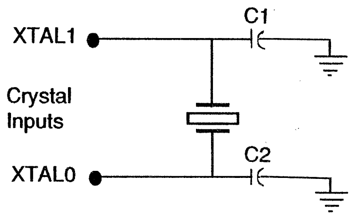


Figure 7: Crystal Connection

Command Logic Unit: This logic unit provides for much more than just controlling the interface between the KIO and the CPU. The main function provided by this unit is to allow the user to configure the internal interrupt daisy chain of the KIO into the order in which he would like the peripherals to interrupt. Any one of the three devices (SIO, CTC, PIO) can be the highest priority while another can be second and the remaining one third. The user can even configure the daisy chain such that no internal peripherals are involved in the chain. Programming of the daisy chain configuration is done by programming the Command Register with the appropriate 3-bit pattern in D_0-D_2 and D_3 set to "1".

A second function of this logic unit is to provide software controllable "hardware" resets to each of the individual devices. This allows an individual peripheral to be reset without having to reset the entire KIO. Requiring bit D_3 to be set to a "1" in order to program the daisy chain configuration allows the user to reset the individual devices without changing the daisy chain. The software reset commands for the individual devices still remain available to the user.

A third function of the Command Register allows the user to obtain use of the additional control signals of the SIO logic instead of the PIA Port. This is done by programming bit D_7 of the Command Register with "1".

PIN DESCRIPTIONS:

A_0-A_3. Address Bus (inputs, active high, 3-state). Use to select which port/register the current transaction cycle is for.

ARDY,BRDY. Port Ready (outputs, active high). these signals indicate that the port is ready for a data transfer. Inmode 0, it indicates that the port has data available for the peripheral device. In mode 1, it indicates that the port is ready to accept data from the peripheral device. In mode2, ARDY indicates that Port A has available for the peripheral device, but that it will not be placed onto PA_0-PA_7 until the $\overline{ASTB}$ signal is active, while BRDY indicates that Port A is able to accept data from a peripheral device. Note that Port B does not support mode 2 operation and can only be used in mode 3 operation when Port A is programmed for mode 2. These signals are not used in mode 3 operation.

$\overline{ASTB}$, $\overline{BSTB}$. Port Strobe (inputs, active low). These signals indicate that the peripheral device has performed a transfer. In mode 0, it indicates that the peripheral device has accepted the data present on the port pins. In mode 1, it causes the data on the port pins to be latched into Port A. In mode 2, the $\overline{ASTB}$ signal causes the data in the output data latch of Port A to be placed onto the Port A pins while the $\overline{BSTB}$ signal will cause the data present on the Port A pins to be latched into the Port A input data latch. The end of the current transaction is noted by the rising edge of these signals. Note that Port B does not support mode 2 operation and can only be used in mode 3 operation when Port A is programmed for mode 2. These signals are not used in mode 3 operation.

CLK/TRG$_0$-CLK/TRG$_3$. External Clock/Timer Trigger (inputs, user selectable active high or low). These four pins correspond to the four counter/timer channels of the KIO. In counter mode, each active edge will cause the downcounter to decrement. In timer mode, an active edge will start the timer.

CLKOUT. Clock Out (output, active high). This output is a divide-by-two of the oscillator (XTAL) input.

CLOCK. System Clock (input, active high). This clock should be the same as (or a derivative of) the CPU clock. If the CLKOUT is to be used as the system clock, then these two pins should be connected together.

$\overline{CS}$. Chip Select (input, active low). Used to activate the internal register decoding mechanism and allow the KIO to perform a data transfer to/from the CPU.

$\overline{CTSA}$, $\overline{CTSB}$. Clear to Send (inputs, active low). These signals are modem control signals to their serial channels. When programmed for Auto Enables, a low on these pins will enable their respective transmitters. If not programmed as Auto Enables, these pins may be used as general-purpose input signals.

D_0-D_7. Data Bus (bidirectional, active high, 3-state). Used for data exchanges between the CPU and the KIO for programming and data transfer. The KIO also monitors the data bus during the RETI instruction cycle to resolve its

$\overline{DCDA}$, $\overline{DCDB}$. Data Carrier Detect (inputs, active low). These signals are modem control signals to their serial channels. When programmed for Auto Enables, a low on these pins will enable their respective receivers. If not programmed as Auto Enables, these pins may be used as general-purpose input signals.

$\overline{DTRA}$, $\overline{DTRB}$. Data Terminal Ready (outputs, active low). These signals are modem control signals for their serial channels. They will follow the state programmed into their respective serial channels. They are multiplexed with Port C, bits 5 and 2 respectively.

IEI. Interrupt Enable In (input, active high). This signal is used with IEO to form a priority daisy chain when there is more than one interrupt-driven device. A high on this line indicates that no higher priority device is requesting an interrupt.

IEO. Interrupt Enable Out (output, active high). This signal is used with IEI to form a priority daisy chain when there is more than one interrupt-driven device. A high on this line indicates that this device and no higher priority device is requesting an interrupt. A low will block any lower priority devices from requesting an interrupt.

$\overline{INT}$. Interrupt Request (output, active low, open-drain). When any of the devices within the KIO requests interrupt servicing, this line will be active.

$\overline{IORQ}$. I/O Request (input, active low). $\overline{IORQ}$ is used with $\overline{RD}$, A0-A3, and $\overline{CS}$ to transfer data between the KIO and the CPU. When $\overline{IORQ}$, $\overline{RD}$, and $\overline{CS}$ are all active, the device selected by A0-A3 transfers data to the CPU. When $\overline{IORQ}$ and $\overline{CS}$ are active, but $\overline{RD}$ is inactive, the device selected by A0-A3 is written into by the CPU, When IORQ and $\overline{M1}$ are both active the KIO will respond with an interrupt vector from the highest priority interrupting device.

$\overline{M1}$. Machine Cycle 1 (input, active low). When $\overline{M1}$ is active and $\overline{RD}$ is active, the Z80 CPU is fetching an instruction from memory; the KIO decodes this cycle to determine if the RETI instruction sequence is being executed. When $\overline{M1}$ and $\overline{IORQ}$ are both active, the KIO decodes the cycle to be an interrupt acknowledge and will respond with a vector from the highest priority interrupting device.

OSC. Oscillator (output, active high). This output is a reference clock for the oscillator.

PA_0-PA_7. Port A Bus (bidirectional, active high, 3-state). This 8-bit bus transfers data between the peripheral device and the port. PA_0 is the least significant bit of the bus.

PB_0-PB_7. Port B Bus (bidirectional, active high, 3-state). This 8-bit bus transfers data between the peripheral device and the port. PB_0 is the least significant bit of the bus. This port can also supply 1.5 mA at 1.5 volts to drive Darlington transistors.

PC_0-PC_7. Port C Bus (bidirectional, active high, 3-state). This 8-bit bus transfers data between the peripheral device and the port. PC_0 is the least significant bit of the bus. These pins are multiplexed to provide either an 8-bit parallel port or additional modem control signals for the serial channels.

$\overline{RD}$. Read (input, active low). when $\overline{RD}$ is active, a memory or I/O read operation is in progress. $\overline{RD}$ is used with A0-A3, $\overline{CS}$ and $\overline{IORQ}$ to transfer data between the KIO and CPU.

$\overline{RESET}$. Reset (input, active low). A low on this pin will force the KIO into a reset condition. This signal must be active for a minimum of three CLOCK cycles. The reset state of the KIO is with the PIO ports in Mode 1 operation and handshakes inactive and interrupts disabled; PIA port in input mode and active; CTC channel counting terminated and interrupts disabled; SIO channels disabled and marking with interrupts disabled. All control registers should be rewritten after a hardware reset.

$\overline{RTSA}$, $\overline{RTSB}$. Request to Send (outputs, active low). These signals are modem control signals for their serial channels. They will follow the inverse state programmed into their respective serial channels. They are multiplexed with Port C, bits 4 and 3 respectively.

$\overline{RxCA}$, $\overline{RxCB}$. Receive Clock (inputs, active low). These clock are used to assemble data in the receiver shift register for their serial channels. Data is sampled on the rising edge of the clock.

RxDA, RxDB. Receive Data (inputs, active high). These are the input data pins to the receive shift register for their serial channels.

$\overline{SYNCA}$, $\overline{SYNCB}$. Synchronization (bidirectional, active low). In the asynchronous mode of operation, these pins act much like the $\overline{CTS}$ and $\overline{DCD}$ pins. Transitions affect the Sync/Hunt status bit for their respective serial channel but serve no other purpose. They are multiplexed with Port C, bits 6 and 1 respectively.

$\overline{TxCA}$, $\overline{TxCA}$. Transmit Clock (inputs, active low). These clocks are used to transmit data from the transmit shift register for their serial channels. Data is transmitted on the falling edge of the clock.

TxDA, TxDB. Transmit Data (outputs, active high). These are the output data pins from the transmitter for their serial channels.

$\overline{WT}$/RDYA, $\overline{WT}$/RDYB. Wait/Ready (outputs, open-drain when programmed as Wait, active high when programmed as Ready). These pins may be programmed as Ready lines for a DMA controller or Wait lines for interface to a CPU. As a Ready line, it indicates (when active) that transmitter or receiver is able to perform a transfer between the serial channel and the DMA. As a Wait line, in dictates (when active), that the CPU should wait until the transmitter or receiver can complete the requested transaction. They are multiplexed with Port C, bits 7 and 0 respectively.

XTALI. Crystal/Clock Connection (input, active high).

XTALO. Crystal Connection (output, active high).

ZC/TO_0-ZC/TO_3. Zero Count/Timeout (outputs, active high). These four pins correspond to the four counter/timer channels of the KIO. Each pin will become active when its corresponding downcounter reaches a zero count.

REGISTER ADDRESSES:

Register 0: PIO Port A Data
Register 1: PIO Port A Command
Register 2: PIO Port B Data
Register 3: PIO Port B Command
Register 4: CTC Channel 0
Register 5: CTC Channel 1
Register 6: CTC Channel 2
Register 7: CTC Channel 3
Register 8: SIO Channel A Data
Register 9: SIO Channel A Command/Status
Register 10: SIO Channel B Data
Register 11: SIO Channel B Command/Status
Register 12: PIA Port C Data
Register 13: PIA Port C Command
Register 14: KIO Command
Register 15: Reserved

REGISTER PROGRAMMING:

PIO Registers: For more detailed information, please consult the PIO Technical Manual.

Interrupt Vector Word (Figure 8). The PIO logic unit is designed to work with the Z80 CPU in interrupt Mode 2. This word must be programmed if interrupts are to be used and bit D_0 must be a zero.

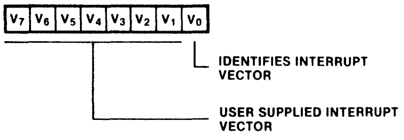


Figure 8: PIO Interrupt Vector Word

Mode Control Word (Figure 9). Selects the port operating mode. This word is required and may be written at any time.

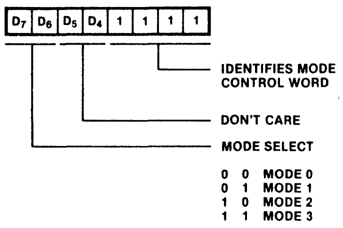


Figure 9: PIO Mode Control Word

I/O Register Control Word (Figure 10). When Mode 3 is selected, the Mode Control Word must be followed by the I/O Register Control Word. This word configures the I/O register, which defines which port lines are inputs or outputs. A "1" indicates input while a "0" indicates output. This word is required when in Mode 3.

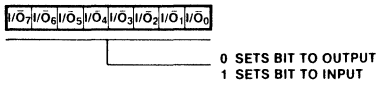
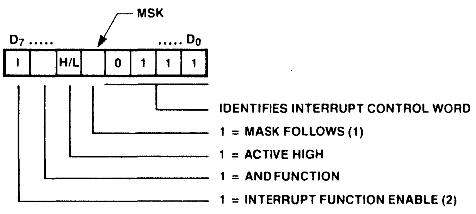


Figure 10: PIO I/O Register Control Word

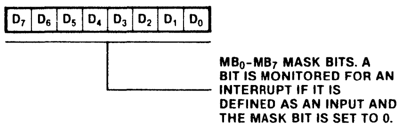
Interrupt Control Word (Figure 11). In Mode 3 operation, handshake signals are not used. Interrupts are generated as a logic function of the input signal levels. The Interrupt Control Word sets the logic conditions and the logic levels required for generating an interrupt. Two logic conditions or functions are available: AND (if all input bits change to the active level, an interrupt is triggered), and OR (if any one of the input bits change to the active logic level, an interrupt is triggered). The user can also program which input bits are to be considered as part of this logic function. Bit D_6 sets the logic function, bit D_5 sets the logic level, and bit D_4 specifies a mask control word to follow.



*NOTE:
1. Regardless of the operating mode, setting Bit D_4 = 1 causes any pending interrupts to be cleared.
2. The port interrupt is not enabled until the interrupt function enable is followed by an active $\overline{M1}$.

Figure 11: PIO Interrupt Control Word

Mask Control Word (Figure 12). This word sets the mask control register, thus allowing any unused bits to be masked off. If any bits are to be masked, then bit D4 of the interrupt Control Word must be set. When bit D4 of the Interrupt Control Word is set, then the next word programmed must be the Mask Control Word. To mask an input bit, the corresponding Mask Control Word bit must be a "1".



D7 D6 D5 D4 D3 D2 D1 D0
MB0-MB7 MASK BITS. A BIT IS MONITORED FOR AN INTERRUPT IF IT IS DEFINED AS AN INPUT AND THE MASK BIT IS SET TO 0.

Figure 12: PIO Mask Control Word

Interrupt Disable Word (Figure 13). This word can be used to enable or disable a port's interrupts without changing the rest of the port's interrupt conditions.

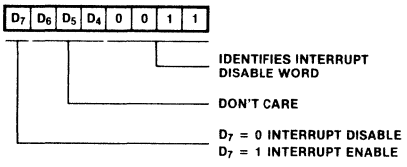


Figure 13: PIO Interrupt Disable Word

CTC Registers: For more detailed information, please consult the CTC Technical Manual.

Channel Control Word (Figure 14). This word sets the operating modes and parameters as described below. Bit D_0 must be a "1" to indicate that this is a Control Word.

Interrupt Enable. Bit D7 enables the interrupt logic so that an interrupt output ($\overline{INT}$) can be generated at zero count. Interrupts can be programmed in either mode and may be enabled or disabled at any time.

Mode. Bit D6 selects either Timer Mode or Counter Mode.

Prescale Factor. Bit D5 selects the prescale factor for use in the timer mode. Either divide-by-16 or divide-by-256 is available.

Clock/Trigger Edge Selector. Bit D4 selects the active edge of the CLK/TRG input pulses.

Timer Trigger. Bit D3 selects the trigger mode for timer operation. Either automatic or external trigger may be selected.

Time Constant. Bit D2 indicates that the next word programmed is time constant data for the downcounter.

Software Reset. Setting bit D1 indicates a software reset operation.

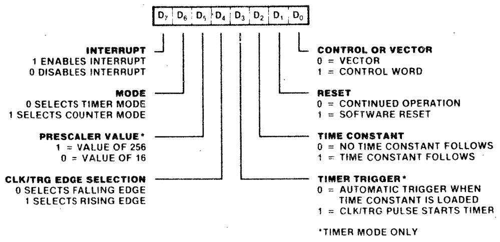


Figure 14: CTC Channel Control Word

Time Constant Word (Figure 15). Before a channel can start counting, it must receive a time constant word. The time constant value may be anywhere between 1 and 256, with "0" being accepted as a count of 256.

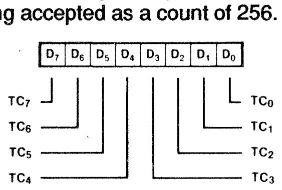


Figure 15: CTC Time Constant Word

Interrupt Vector Word (Figure 16). If one or more of the CTC channels have interrupts enabled, then the Interrupt Vector Word must be programmed. Only the five most significant bits of this word are programmed, and bit D_0 must be "0". Bits D_2-D_1 are automatically modified by the CTC channel when it responds with an interrupt vector.

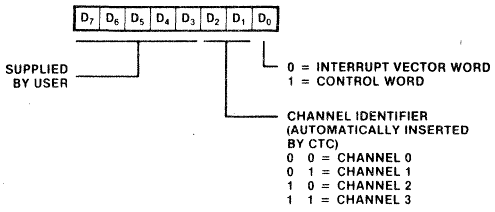


Figure 16: CTC Interrupt Vector Word

SIO Registers: For more detailed information, please consult the SIO Technical Manual.

Read Registers (Figure 17). The SIO channel B contains three read registers while channel A contains only two that can be read to obtain status information. To read the contents of a register (other than RRo), the program must first write a pointer to WRo in exactly the same manner as a write register operation. The next I/O read cycle will place the contents of the selected read register onto the data bus.

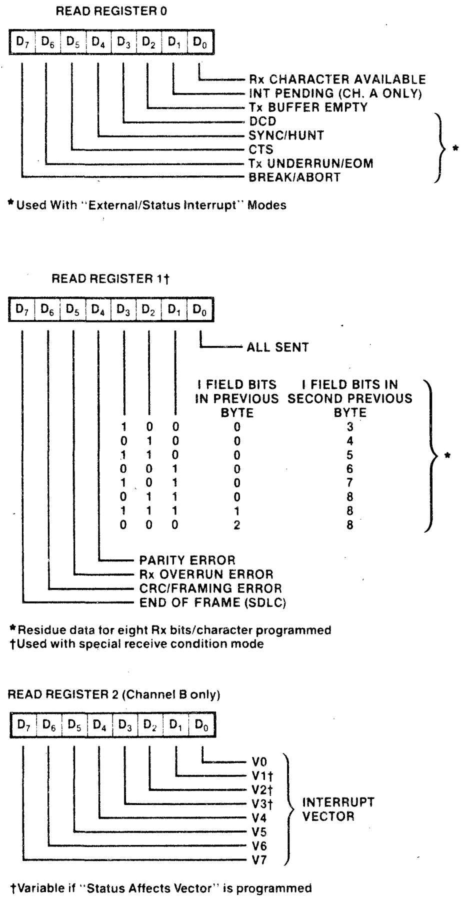
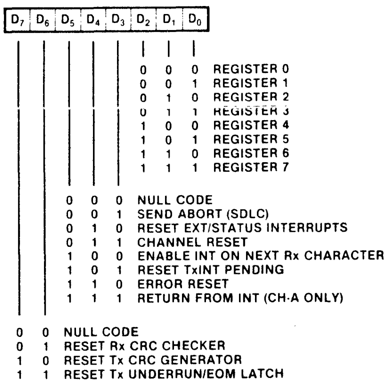


Figure 17: SIO Read Registers

Write Registers (Figure 18). The SIO channel B contains eight write registers while channel A contains only seven that are programmed to configure the operating modes and characteristics of each channel. With the exception of WRo, programming the write registers is a two step operation. The first operation is a pointer written to WRo that point to the selected register. The second operation is the actual control word that is written into the register to configure the SIO channel.

WRITE REGISTER 0

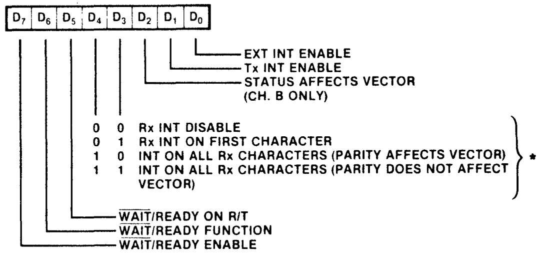


D2	D1	D0	
0	0	0	REGISTER 0
0	0	1	REGISTER 1
0	1	0	REGISTER 2
0	1	1	REGISTER 3
1	0	0	REGISTER 4
1	0	1	REGISTER 5
1	1	0	REGISTER 6
1	1	1	REGISTER 7

D5	D4	D3	
0	0	0	NULL CODE
0	0	1	SEND ABORT (SDLC)
0	1	0	RESET EXT/STATUS INTERRUPTS
0	1	1	CHANNEL RESET
1	0	0	ENABLE INT ON NEXT Rx CHARACTER
1	0	1	RESET TxINT PENDING
1	1	0	ERROR RESET
1	1	1	RETURN FROM INT (CH-A ONLY)

D7	D6	
0	0	NULL CODE
0	1	RESET Rx CRC CHECKER
1	0	RESET Tx CRC GENERATOR
1	1	RESET Tx UNDERRUN/EOM LATCH

WRITE REGISTER 1



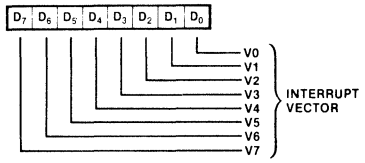
- D0 — EXT INT ENABLE
- D1 — Tx INT ENABLE
- D2 — STATUS AFFECTS VECTOR (CH. B ONLY)

D4	D3	
0	0	Rx INT DISABLE
0	1	Rx INT ON FIRST CHARACTER
1	0	INT ON ALL Rx CHARACTERS (PARITY AFFECTS VECTOR) *
1	1	INT ON ALL Rx CHARACTERS (PARITY DOES NOT AFFECT VECTOR) *

- D5 — WAIT/READY ON R/T
- D6 — WAIT/READY FUNCTION
- D7 — WAIT/READY ENABLE

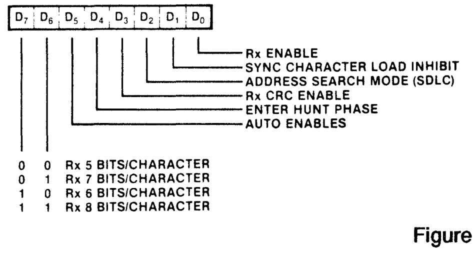
*Or on special condition

WRITE REGISTER 2 (Channel B only)



D0–D7: V0, V1, V2, V3, V4, V5, V6, V7 — INTERRUPT VECTOR

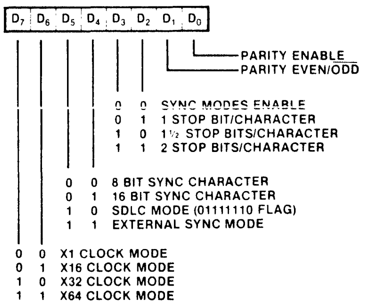
WRITE REGISTER 3



- D0 — Rx ENABLE
- D1 — SYNC CHARACTER LOAD INHIBIT
- D2 — ADDRESS SEARCH MODE (SDLC)
- D3 — Rx CRC ENABLE
- D4 — ENTER HUNT PHASE
- D5 — AUTO ENABLES

D7	D6	
0	0	Rx 5 BITS/CHARACTER
0	1	Rx 7 BITS/CHARACTER
1	0	Rx 6 BITS/CHARACTER
1	1	Rx 8 BITS/CHARACTER

WRITE REGISTER 4



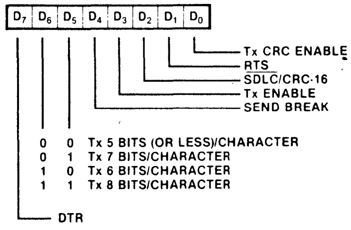
- D0 — PARITY ENABLE
- D1 — PARITY EVEN/ODD

D3	D2	
0	0	SYNC MODES ENABLE
0	1	1 STOP BIT/CHARACTER
1	0	1½ STOP BITS/CHARACTER
1	1	2 STOP BITS/CHARACTER

D5	D4	
0	0	8 BIT SYNC CHARACTER
0	1	16 BIT SYNC CHARACTER
1	0	SDLC MODE (01111110 FLAG)
1	1	EXTERNAL SYNC MODE

D7	D6	
0	0	X1 CLOCK MODE
0	1	X16 CLOCK MODE
1	0	X32 CLOCK MODE
1	1	X64 CLOCK MODE

WRITE REGISTER 5

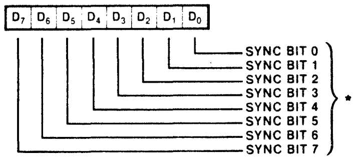


- D0 — Tx CRC ENABLE
- D1 — RTS
- D2 — SDLC/CRC-16
- D3 — Tx ENABLE
- D4 — SEND BREAK

D6	D5	
0	0	Tx 5 BITS (OR LESS)/CHARACTER
0	1	Tx 7 BITS/CHARACTER
1	0	Tx 6 BITS/CHARACTER
1	1	Tx 8 BITS/CHARACTER

- D7 — DTR

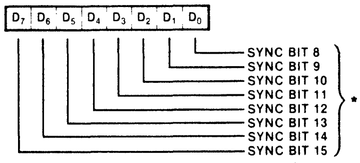
WRITE REGISTER 6



D0–D7: SYNC BIT 0, SYNC BIT 1, SYNC BIT 2, SYNC BIT 3, SYNC BIT 4, SYNC BIT 5, SYNC BIT 6, SYNC BIT 7 *

*Also SDLC address field

WRITE REGISTER 7



D0–D7: SYNC BIT 8, SYNC BIT 9, SYNC BIT 10, SYNC BIT 11, SYNC BIT 12, SYNC BIT 13, SYNC BIT 14, SYNC BIT 15 *

*For SDLC it must be programmed to "01111110" for flag recognition

Figure 18: SIO Write Registers

PIA Registers:

The PIA port can be configured for any combination of input and output bits. The direction is controlled by writing to the PIA Control Register. A "1" written to a bit position will indicate that the respective bit should be an input (Figure 19). All bits are inputs on reset.

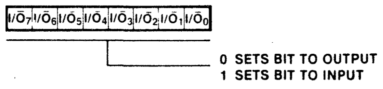


Figure 19: PIA Control Register

KIO Command Register:

The KIO Command Register is used to program software resets and to configure the internal interrupt daisy chain priority (Figure 20). This register should be programmed before all others. The reset control bits are momentary, writing a "1" will pulse an internal reset signal to the appropriate device.

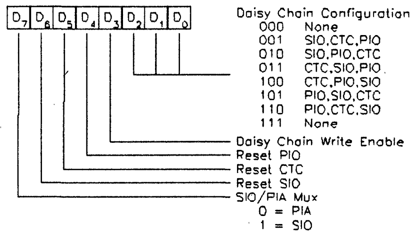


Figure 20: KIO Command Register

ABSOLUTE MAXIMUM RATINGS:

Voltage on Vcc with respect to Vss
. . . . -0.3V to +7.0V
Voltages on all inputs with respect to Vss
. . . . -0.3V to Vcc+0.3V
Operating Ambient Temperature
. . . . See Ordering Information
Storage Temperature
. . . . -65 C to +150 C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS:

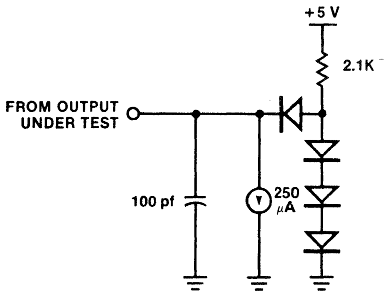
The DC Characteristics and Capacitance sections below apply to the following standard test conditions, unless otherwise noted. All voltages are referenced to GND (0V). Positive current flows into the referenced pin.

Available operating temperature ranges are:
- S = 0 C to +70 C
- E = -40 C to +100 C

Voltage Supply Range: +5.0V ± 10%

All AC parameters assume a load capaitance of 100 pF. Add 10 ns delay for each 50 pF increase in load up to a maximum of 200 pF for the data bus and 100 pF for the address and control lines. AC timing measurements are referenced to 1.5 volts (except for CLOCK, which is referenced to the 10% and 90% points).

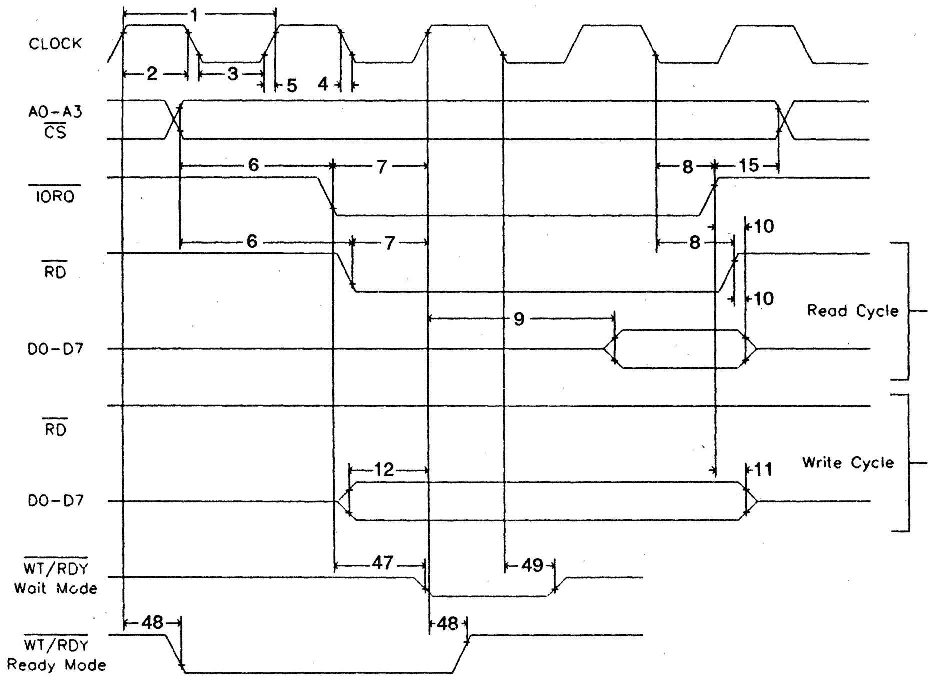
The Ordering Information section lists temperature ranges and product numbers. Package drawings are in the Package Information section. Refer to the Literature List for additional documentation.



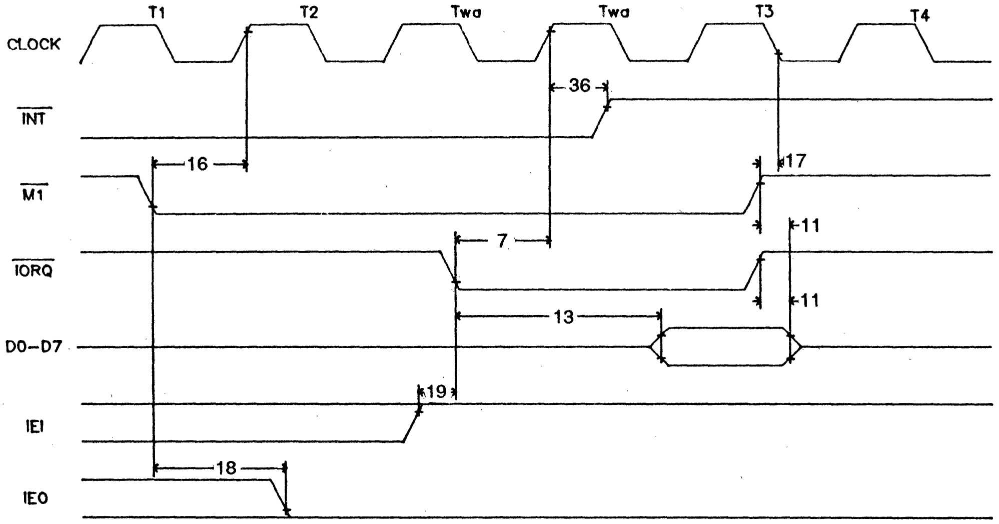
DC CHARACTERISTICS:

Symbol	Item	min	max	Unit	Condition
V_{ILC}	Clock Input Low Voltage	-0.3	+0.45	V	
V_{IHC}	Clock Input High Voltage	Vcc-0.6	Vcc+0.3	V	
V_{IL}	Input Low Voltage	-0.3	+0.8	V	
V_{IH}	Input High Voltage	2.2	Vcc	V	
V_{OL}	Output "L" Voltage		+0.4	V	I_{OL}= 2.0 mA
V_{OH1}	Output "H" Voltage 1	2.4		V	I_{OH}=-1.6 mA
V_{OH2}	Output "H" Voltage 2	Vcc-0.8		V	I_{OH}=-250 μA
I_{LI}	Input Leakage Current		±10.0	μA	Vin=0.4~Vcc
I_{OL}	3-State Leakage Current		±10.0	μA	Vin=0.4~Vcc
$I_{L(SY)}$	$\overline{\text{SYNC}}$ Pin Leakage Current	+10	-40	μA	Vin=0.4~Vcc
I_{OHD}	Darlington Drive Current	-1.5		mA	V_{OH}=1.5 V R_{EXT}=390 Ohms
Icc	Power Supply Current				
	6 MHz		15	mA	V_{CC}=5 V
	8 MHz		20	mA	V_{IH}=V_{CC}-.2 V
					V_{IL}=.2 V

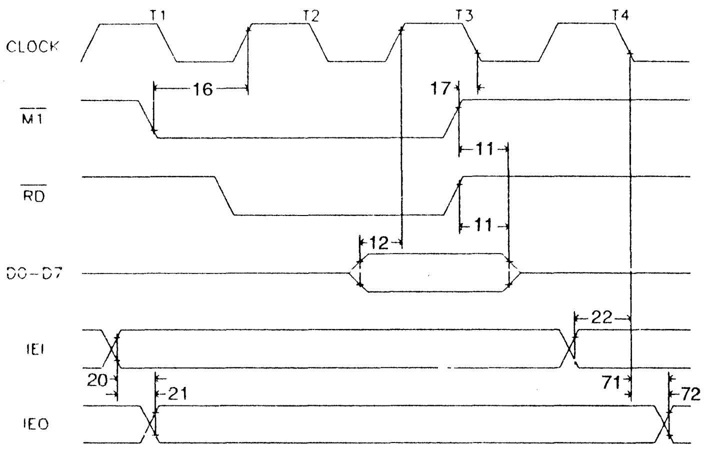
Over specified temperature and voltage ranges.



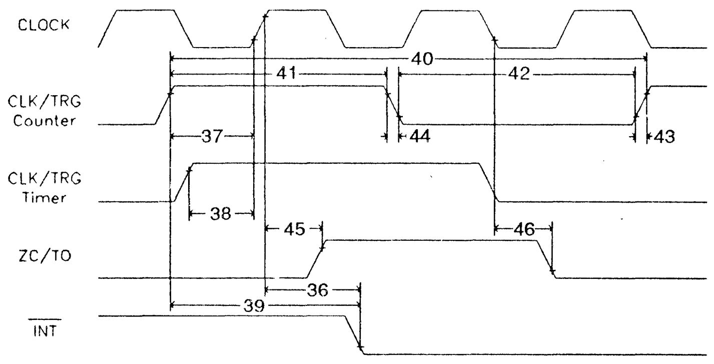
I/O Read/Write Timing ($\overline{M1}$ = 1)



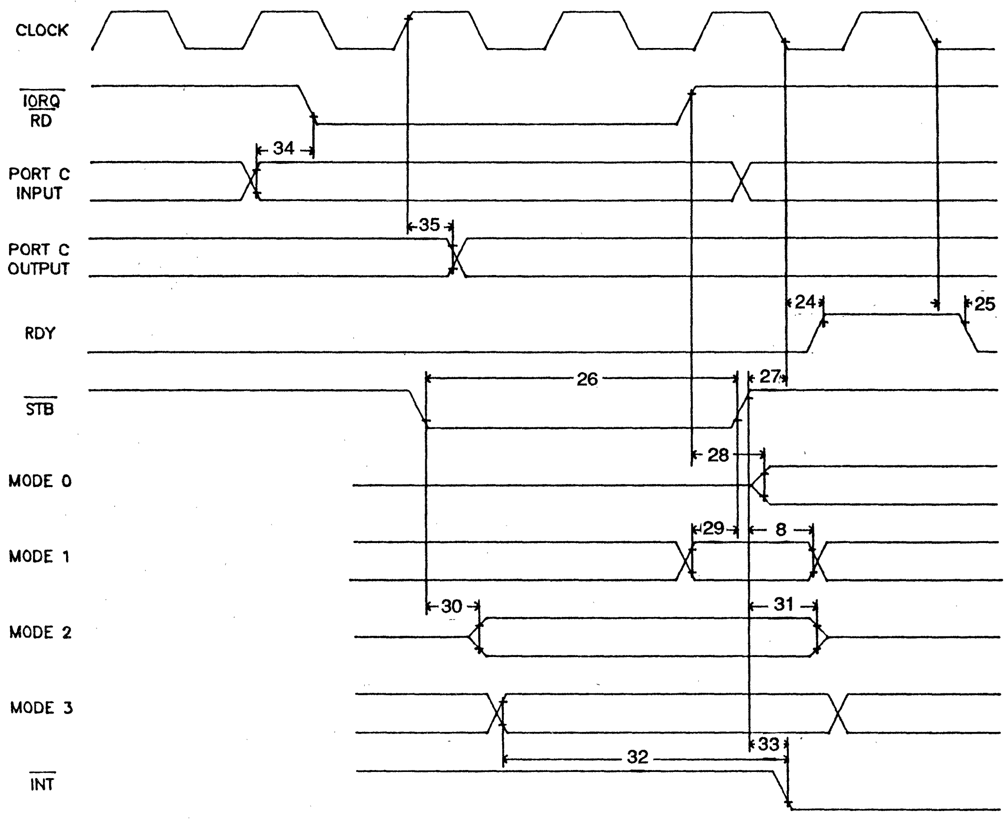
Interrupt Acknowledge Cycle



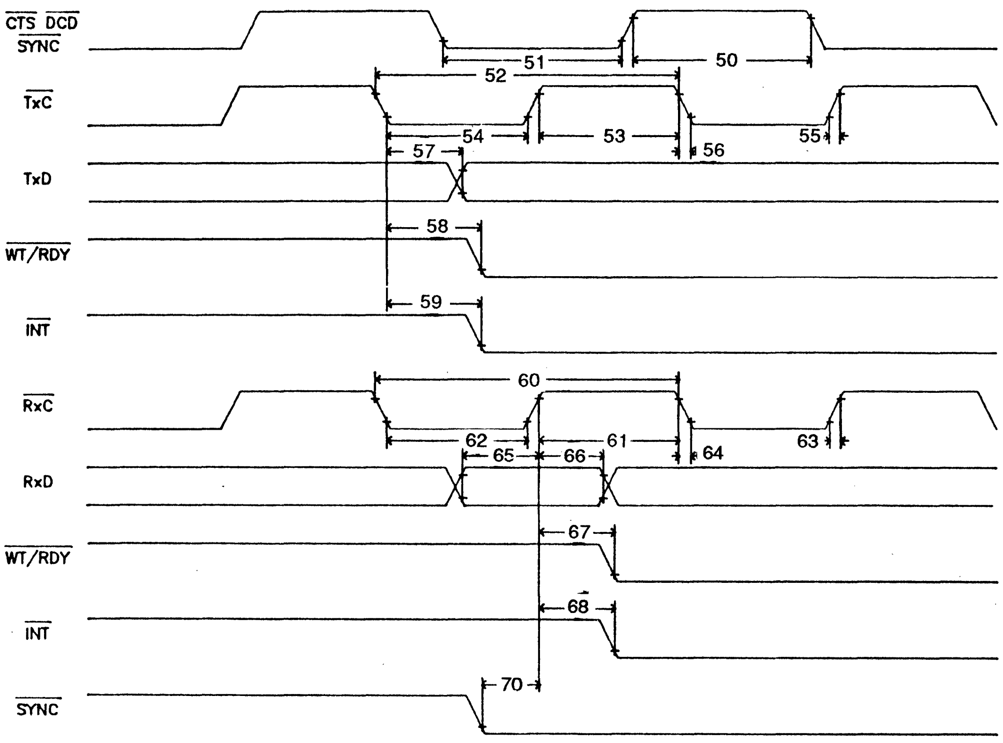
Opcode Fetch Cycle



Counter/Timer Timing



Port I/O Read/Write Timing



Serial I/O Timing

CAPACITANCE:

Symbol	Parameter	min	max	unit
C_{CLOCK}	Clock Capactiance		10	pF
C_{IN}	Input Capacitance		10	pF
C_{OUT}	Output Capacitance		15	pF

T_A=25 C, f=1 MHz

AC CHARACTERISTICS:

			Z84C9008			
No.	Symbol	Parameter	min	max	units	notes
1	TcC	Clock Cycle Time	125	DC	ns	
2	TwCh	Clock Pulse Width (High)	55	DC	ns	
3	TwCl	Clock Pulse Width (Low)	55	DC	ns	
4	TfC	Clock Fall Time		10	ns	
5	TrC	Clock Rise Time		10	ns	
6	TsA(Rlf)	Address, $\overline{CS}$ Setup to $\overline{RD}$,$\overline{IORQ}$ ↓	50		ns	
7	TsRl(Cr)	$\overline{RD}$,$\overline{IORQ}$ to CLOCK↑ Setup	60		ns	
8	Th	Hold Time for Specified Setup	15		ns	
9	TdCr(DO)	CLOCK ↑ to Data Out Delay		100	ns	
10	TdRlr(DOz)	$\overline{RD}$,$\overline{IORQ}$ ↑ to Data Float Delay		75	ns	
11	ThRDr(D)	$\overline{M1}$,$\overline{RD}$,$\overline{IORQ}$ ↑ to Data Hold	15		ns	
12	TsD(Cr)	Data In to CLOCK ↑ Setup	30		ns	
13	TdIOf(DO)	$\overline{IORQ}$ ↓ to Data Out Delay (INTACK Cycle)		90	ns	
14	ThIOr(D)	$\overline{IORQ}$ ↑ to Data Hold	15		ns	
15	ThIOr(A)	$\overline{IORQ}$ ↑ to Address Hold	15		ns	
16	TsM1f(Cr)	$\overline{M1}$ ↓ to CLOCK ↑ Setup	40		ns	
17	TsM1r(Cf)	$\overline{M1}$ ↑ to CLOCK ↓ Setup (M1 Cycle)	-15		ns	
18	TdM1f(IEOf)	$\overline{M1}$ ↓ to IEO ↓ Delay (Interrupt immediately preceding $\overline{M1}$ ↓)		100	ns	
19	TsIEI(IOf)	IEI to $\overline{IORQ}$ ↓ Setup	30		ns	
20	TdIElf(IEOf)	IEI ↓ to IEO ↓ Delay		70	ns	
21	TdIElr(IEOr)	IEI ↑ to IEO ↑ Delay (after ED Decode)		70	ns	
22	TsIEI(Cr)	IEI to CLOCK ↓ Setup (for 4D decode)	50		ns	
23	TsIOr(Cr)	$\overline{IORQ}$ ↑ to CLOCK ↑ Setup (to activate $\overline{RDY}$ on next clock)	100		ns	
24	TdCf(RDYr)	CLOCK ↓ to $\overline{RDY}$ ↑ Delay		100	ns	
25	TdCf(RDYf)	CLOCK ↓ to $\overline{RDY}$ ↓ Delay		100	ns	
26	TwSTB	$\overline{STB}$ Pulse Width	100		ns	
27	TsSTBr(Cr)	$\overline{STB}$ ↑ to CLOCK ↓ Setup (to activate $\overline{RDY}$ on next clock)	100		ns	
28	TdIOr(PD)	$\overline{IORQ}$ ↑ to Port Data Valid (Mode 0)		140	ns	
29	TsPD(STBr)	Port A,B Data to $\overline{STB}$ ↑ Setup	140		ns	
30	TdSTBf(PD)	$\overline{STB}$ ↓ to Port A,B Data Valid Delay (Mode 2)		150	ns	
31	TdSTBr(PDz)	$\overline{STB}$ ↑ to Port A,B Data Float Delay (Mode 2)		140	ns	
32	TdPD(INTf)	Port A,B Data Match to $\overline{INT}$ ↓ Delay (Mode 3)		360	ns	
33	TdSTBr(INTf)	$\overline{STB}$ ↑ to $\overline{INT}$ ↓ Delay		290	ns	
34	TsPD(Rlf)	Port Data to $\overline{RD}$,$\overline{IORQ}$ ↓ Setup				
35	TdCr(PD)	Clock ↑ to Port Data Valid Delay		80	ns	
36	TdCr(INTf)	CLOCK ↑ to $\overline{INT}$ Delay			ns	(1)
37	TsCTRr(Cr)c	CLK/TRG ↑ to CLOCK ↑ Setup (for immediate count, counter mode)	90		ns	

AC CHARACTERISTICS:

			Z84C9008			
No.	Symbol	Parameter	min	max	units	notes
38	TsCTRr(Cr)t	CLK/TRG ↑ to CLOCK ↑ Setup (for enabling prescaler on following CLOCK ↑, timer mode)	90		ns	
39	TdCTRr(INTf)	CLK/TRG ↑ to $\overline{\text{INT}}$ ↓ Delay				
		TsCTRr(Cr) satisfied				(2)
		TsCTRr(Cr) not satisfied				(3)
40	TcCTR	CLK/TRG Cycle Time	250	DC	ns	
41	TwCTRh	CLK/TRG Width High	90	DC	ns	
42	TwCTRl	CLK/TRG Width Low	90	DC	ns	
43	TrCTR	CLK/TRG Rise Time		30	ns	
44	TfCTR	CLK/TRG Fall Time		30	ns	
45	TdCr(ZCr)	CLOCK ↑ to ZC/TO ↑ Delay		80	ns	
46	TdCf(ZCf)	CLOCK ↓ to ZC/TO ↓ Delay		80	ns	
47	TdIOf(W/Rf)	$\overline{\text{IORQ}}$ ↓ to $\overline{\text{WT/RDY}}$ ↓ Delay (Wait Mode)		130	ns	
48	TdCr(W/Rf)	CLOCK ↑ to $\overline{\text{WT/RDY}}$ Delay (Ready Mode)		80	ns	
49	TdCf(W/Rz)	CLOCK ↓ to $\overline{\text{WT/RDY}}$ Float Delay (Wait Mode)		90	ns	
50	TwPh	Pulse Width High	150		ns	
51	TwPl	Pulse Width Low	150		ns	
52	TcTxC	$\overline{\text{TxC}}$ Cycle Time	250	DC	ns	
53	TwTxCh	$\overline{\text{TxC}}$ Width High	85	DC	ns	
54	TwTxCl	$\overline{\text{TxC}}$ Width Low	85	DC	ns	
55	TrTxC	$\overline{\text{TxC}}$ Rise Time		60	ns	
56	TfTxC	$\overline{\text{TxC}}$ Fall Time		60	ns	
57	TdTxCf(TxD)	$\overline{\text{TxC}}$ ↓ to TxD Delay (x1 mode)		160	ns	
58	TdTxCf(W/Rf)	$\overline{\text{TxC}}$ ↓ to $\overline{\text{WT/RDY}}$ ↓ Delay (Ready Mode)		5-9		(4)
59	TdTxCf(INTf)	$\overline{\text{TxC}}$ ↓ to $\overline{\text{INT}}$ ↓ Delay		5-9		(4)
60	TcRxC	$\overline{\text{RxC}}$ Cycle Time	250	DC	ns	
61	TwRxCh	$\overline{\text{RxC}}$ Width High	85	DC	ns	
62	TwRxCl	$\overline{\text{RxC}}$ Width Low	85	DC	ns	
63	TrRxC	$\overline{\text{RxC}}$ Rise Time		60	ns	
64	TfRxC	$\overline{\text{RxC}}$ Fall Time		60	ns	
65	TsRxD(RxCr)	RxD to $\overline{\text{RxC}}$ ↑ Setup	0		ns	
66	ThRxCr(RxD)	$\overline{\text{RxC}}$ ↑ to RxD Hold Time	80		ns	
67	TdRxCr(W/Rf)	$\overline{\text{RxC}}$ ↑ to $\overline{\text{W/RDY}}$ ↓ Delay (Ready Mode)		10-13		(4)
68	TdRxCf(INTf)	$\overline{\text{RxC}}$ ↓ to $\overline{\text{INT}}$ ↓ Delay		10-13		(4)
69	TdRxCr(SYNCf)	$\overline{\text{RxC}}$ ↑ to $\overline{\text{SYNC}}$ ↓ Delay (output mode)		4-7		(4)
70	TsSYNCf(RxCr)	$\overline{\text{SYNC}}$ ↓ to $\overline{\text{RxC}}$ ↑ Setup (external sync mode)	-100		ns	
71	TdCf(IEOr)	Clock ↓ to IEO ↑ Delay		90	ns	
72	TdCf(IEOf)	Clock ↓ to IEO ↓ Delay		110	ns	

Notes:
1: TcC+100
2: TdCr(INTf)+TsCTRr(Cr)c or TdCr(INTf)+TsCTRr(Cr)t
3: TcC+TdCr(INTf)+TsCTRr(Cr)c or TcC+TdCr(INTf)+TsCTRr(Cr)t
4: Units equal to System Clock periods (T_cC)

January 1989

Z80180
Z180 MPU

FEATURES:

- Operating Frequency to 10 MHz
- On-Chip MMU Supports Extended Address Space
- Two DMA Channels
- On-Chip Wait State Generators
- Two UART Channels
- Two 16-Bit Timer Channels
- On-Chip Interrupt Controller
- On-Chip Clock Oscillator/Generator
- Clocked Serial I/O Port
- Code Compatible with Zilog Z80 CPU
- Extended Instructions
- 6 MHz Version Supports 6.144 MHz CPU Clock Operation

GENERAL DESCRIPTION:

Based on a microcoded execution unit and an advanced CMOS manufacturing technology, the Z80180 is an 8-bit MPU which provides the benefits of reduced system costs and low power operation while offering higher performance and maintaining compatibility with a large base of industry standard software written around the Zilog Z80 CPU.

Higher performance is obtained by virtue of higher operating frequencies, reduced instruction execution times, an enhanced instruction set, and an on-chip memory management unit (MMU) with the capability of addressing up to 1 Mbyte of memory.

Reduced system costs are obtained by incorporating several key system functions on-chip with the CPU. These key functions include I/O devices such as DMA, UART, and timer channels. Also included on-chip are several "glue" functions such as dynamic RAM refresh control, wait state generators, clock oscillator, and interrupt controller.

Not only does the Z80180 consume a low amount of power during normal operation, but it also provides two operating modes that are designed to drastically reduce the power consumption even further. The SLEEP mode reduces power by placing the CPU into a "stopped" state, thereby consuming less current, while the on-chip I/O device is still operating. The SYSTEM STOP mode places both the CPU and the on-chip peripherals into a "stopped" mode, thereby reducing power consumption even further.

When combined with other CMOS VLSI devices and memories, the Z80180 provides an excellent solution to system applications requiring high performance, and low power operation.

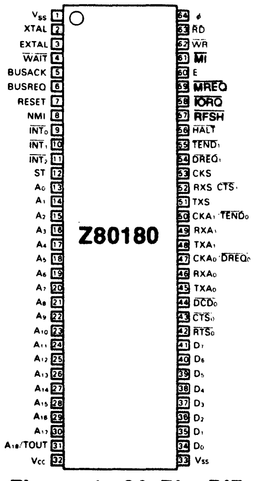


Figure 1. 64 Pin DIP

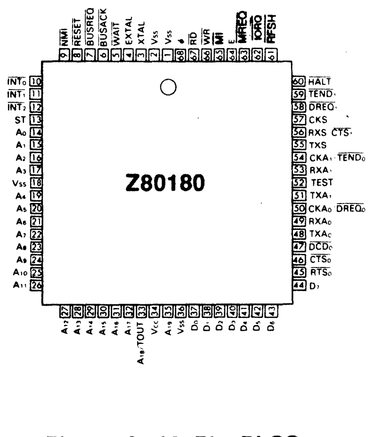


Figure 2. 68 Pin PLCC

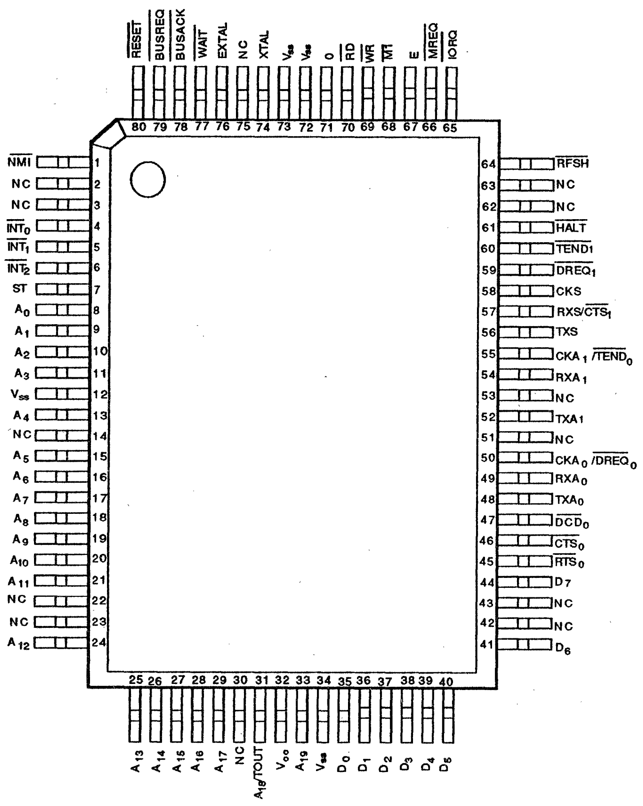


Figure 2b. 80-pin Quad Flat Pack

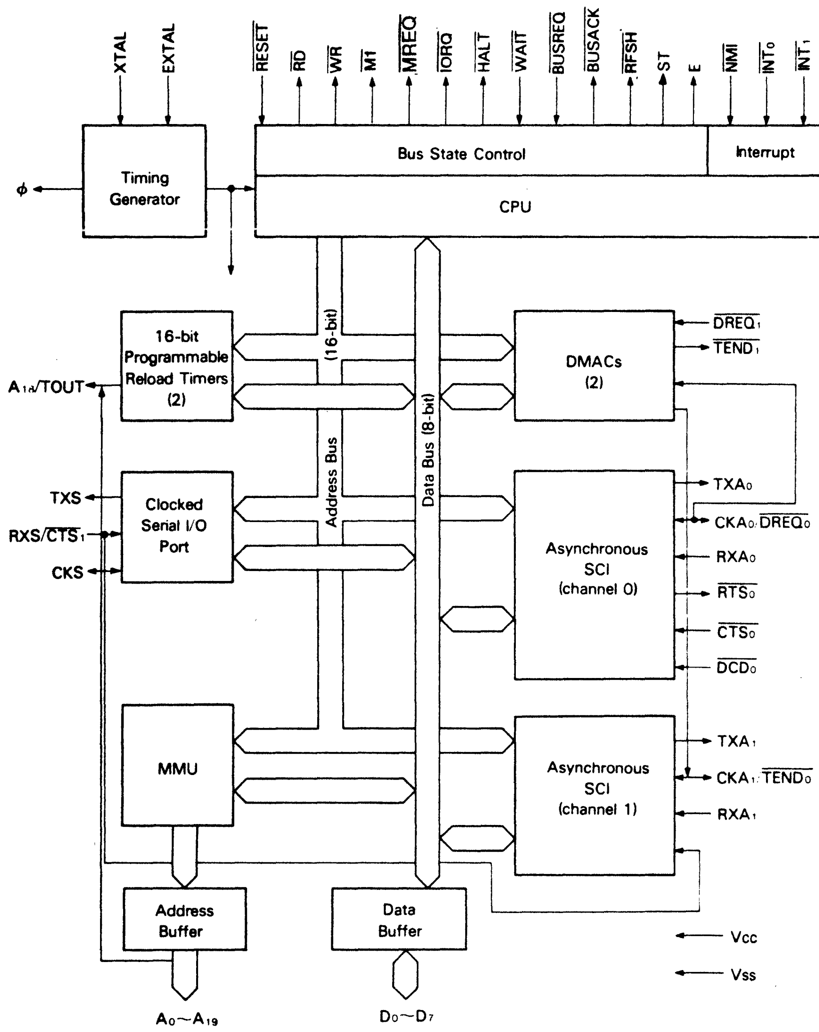


Figure 3. Block Diagram

PIN DESCRIPTION:

A_0-A_{19}. *Address Bus (Output, active High, 3-state).* A_0-A_{19} form a 20-bit address bus. The Address Bus provides the address for memory data bus exchanges, up to 1 Mbyte, and I/O data bus exchanges, up to 64K. The address bus enters a high impedance state during reset and external bus acknowledge cycles. Address line A_{18} is multiplexed with the output of PRT channel 1 (TOUT, selected as address output on reset) and address line A_{19} is not available in DIP versions of the Z80180.

$\overline{\text{BUSACK}}$. *Bus Acknowledge (Output, active Low).* $\overline{\text{BUSACK}}$ indicates the requesting device, the MPU address and data bus, and some control signals, have entered their high impedance state.

$\overline{\text{BUSREQ}}$. *Bus Request (Input, active Low).* This input is used by external devices (such as DMA controllers) to request access to the system bus. This request has a higher priority than $\overline{\text{NMI}}$ and is always recognized at the end of the current machine cycle. This signal will stop the CPU from executing further instructions and places the address and data buses, and other control signals, into the high impedance state.

CKA_0, CKA_1. *Asynchronous Clock 0 and 1 (Bidirectional, active High).* These pins are the transmit and receive clocks for the synchronous channels. CKA_0 is multiplexed with $\overline{\text{DREQ}_0}$ and CKA_1 is multiplexed with $\overline{\text{TEND}_0}$.

CKS. *Serial Clock (Bidirectional, active High).* This line is clock for the CSIO channel.

CLOCK. *System Clock (Output, active High).* The output is used as a reference clock for the MPU and the external system. The frequency of this output is equal to one-half that of the crystal or input clock frequency.

$\overline{\text{CTS}_0}$-$\overline{\text{CTS}_1}$. *Clear to Send 0 and 1 (Inputs, active Low).* These lines are modem control signals for the ASCI channels. $\overline{\text{CTS1}}$ is multiplexed with RXS.

D_0-D_7. *Data Bus (Bidirectional, active High, 3-state).* D_0-D_7 constitute an 8-bit bidirectional data bus, used for the transfer of information to and from I/O and memory devices. The data bus enters the high impedance state during reset and external bus acknowledge cycles.

$\overline{\text{DCD}_0}$. *Data Carrier Detect 0 (Input, active Low).* This is a programmable modem control signal for ASCI channel 0.

$\overline{\text{DREQ}_0}$, $\overline{\text{DREQ}_1}$. DMA Request 0 and 1 (Input, active Low). $\overline{\text{DREQ}}$ is used to request a DMA transfer from one of the on-chip DMA channels. The DMA channels monitor these inputs to determine when an external device is ready for a read or write operation. These inputs can be programmed to be either level or edge sensed. $\overline{\text{DREQ}_0}$ is multiplexed with CKA_0.

E. *Enable Clock (Output, active High).* Synchronous machine cycle clock output during bus transactions.

EXTAL. *External Clock/Crystal (Input, active High).* Crystal oscillator connection. An external clock can be input to the Z80180 on this pin when a crystal is not used. This input is Schmitt triggered.

$\overline{\text{HALT}}$. *Halt/Sleep Status (Output, active Low).* This output is asserted after the CPU has executed either the HALT or SLP instruction, and is waiting for either non-maskable or maskable interrupt before operation can resume. It is also used with the $\overline{\text{M1}}$ and ST signals to decode status of the CPU machine cycle.

$\overline{\text{INT}_0}$. *Maskable Interrupt Request 0 (Input, active Low).* This signal is generated by external I/O devices. The CPU will honor this request at the end of the current instruction cycle as long as the $\overline{\text{NMI}}$ and $\overline{\text{BUSREQ}}$ signals are inactive. The CPU acknowledges this interrupt request with an interrupt acknowledge cycle. During this cycle, both the $\overline{\text{M1}}$ and $\overline{\text{IORQ}}$ signals will become active.

$\overline{\text{INT}_1}$, $\overline{\text{INT}_2}$. *Maskable Interrupt Requests 1 and 2 (Inputs, active Low).* This signal is generated by external I/O devices. The CPU will honor these requests at the end of the current instruction cycle as long as the $\overline{\text{NMI}}$, $\overline{\text{BUSREQ}}$, and $\overline{\text{INT}_0}$ signals are inactive. The CPU will acknowledge these interrupt requests with an interrupt acknowledge cycle. Unlike the acknowledgement for $\overline{\text{INT}_0}$, during this cycle neither the $\overline{\text{M1}}$ or $\overline{\text{IORQ}}$ signals will become active.

$\overline{\text{IORQ}}$. *I/O Request (Output, active Low, 3-state).* $\overline{\text{IORQ}}$ indicates that the address bus contains a valid I/O address for an I/O read or I/O write operation. $\overline{\text{IORQ}}$ is also generated, along with $\overline{\text{M1}}$, during the acknowledgement of the $\overline{\text{INT}_0}$ input signal to indicate that an interrupt response vector can be placed onto the data bus. This signal is analogous to the $\overline{\text{IOE}}$ signal of the Z64180.

$\overline{\text{M1}}$. *Machine Cycle 1 (Output, active Low).* Together with $\overline{\text{MREQ}}$, $\overline{\text{M1}}$ indicates that the current cycle is the opcode fetch cycle of an instruction execution. Together with $\overline{\text{IORQ}}$, $\overline{\text{M1}}$ indicates that the current cycle is for an interrupt acknowledge. It is also used with the $\overline{\text{HALT}}$ and ST signal to decode status of the CPU machine cycle. This signal is analogous to the $\overline{\text{LIR}}$ signal of the Z64180.

$\overline{\text{MREQ}}$. *Memory Request (Output, active Low, 3-state).* $\overline{\text{MREQ}}$ indicates that the address bus holds a valid address for a memory read or memory write operation. This signal is analogous to the $\overline{\text{ME}}$ signal of the Z64180.

$\overline{\text{NMI}}$. *Non-maskable Interrupt (Input, negative edge triggered).* $\overline{\text{NMI}}$ has a higher priority than $\overline{\text{INT}}$ and is always recognized at the end of an instruction, regardless of the state of the interrupt enable flip-flops. This signal forces CPU execution to continue at location 0066H.

$\overline{RD}$. *Read (Output, active Low, 3-state).* $\overline{RD}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O or memory device should use this signal to gate data onto the CPU data bus.

$\overline{RFSH}$. *Refresh (Output, active Low).* Together with $\overline{MREQ}$, $\overline{RFSH}$ indicates that the current CPU machine cycle and the contents of the address bus should be used for refresh of dynamic memories. The low order 8 bits of the address bus (A_7-A_0) contain the refresh address.

This signal is analogous to the $\overline{REF}$ signal of the Z64180.

$\overline{RTS_0}$. *Request to Send 0 (Output, active Low).* This is a programmable modem control signal for ASCI channel 0.

RXA_0, RXA_1. *Receive Data 0 and 1 (Inputs, active High).* These signals are the receive data to the ASCI channels.

RXS. *Clocked Serial Receive Data (Input, active High).* This line is the receiver data for the CSIO channel. RXS is multiplexed with the $\overline{CTS1}$ signal for ASCI channel 1.

ST. *Status (Output, active High).* This signal is used with the $\overline{M1}$ and $\overline{HALT}$ output to decode the status of the CPU machine cycle.

Note that the output from M1 is affected by the status of the M1E bit in OMCR register. Table 1 shows the status while M1E = 1.

ST	$\overline{HALT}$	$\overline{M1}$	Operation
0	1	0	CPU operation (1st op-code fetch)
1	1	0	CPU operation (2nd op-code and 3rd op-code fetch)
1	1	1	CPU operation (MC except for op-code fetch)
0	X	1	DMA operation
0	0	0	HALT mode
1	0	1	SLEEP mode (including SYSTEM STOP mode)

NOTE X: Don't care
MC: Machine cycle

Table 1. Status Summary

$\overline{TEND_0}$, $\overline{TEND_1}$. *Transfer End 0 and 1 (Outputs, active Low).* This output is asserted active during the last write cycle of a DMA operation. It is used to indicate the end of the block transfer. $\overline{TEND_0}$ in multiplexed with CKA_1.

TOUT. *Timer Out (Output, active High).* TOUT is the pulse output from PRT channel 1. This line is multiplexed with A_{18} of the address bus.

TXA_0, TXA_1. *Transmit Data 0 and 1 (Outputs, active High).* These signals are the transmitted data from the ASCI channels. Transmitted data changes are with respect to the falling edge of the transmit clock.

TXS. *Clocked Serial Transmit Data (Output, active High).* This line is the transmitted data from the CSIO channel.

$\overline{WAIT}$. *Wait (Input, active Low).* $\overline{WAIT}$ indicates to the MPU that the addressed memory or I/O devices are not ready for a data transfer. This input is used to induce additional clock cycles into the current machine cycle. The $\overline{WAIT}$ input is sampled on the falling edge of T_2 (and subsequent wait states). If the input is sampled low, then additional wait states are inserted until the $\overline{WAIT}$ input is sampled high, at which time execution will continue.

$\overline{WR}$. *Write (Output, active Low, 3-state).* $\overline{WR}$ indicates that the CPU data bus holds valid data to be stored at the addressed I/O or memory location.

XTAL. *Crystal (Input, active High).* Crystal oscillator connection. This pin should be left open if an external clock is used instead of a crystal. The oscillator input is not a TTL level (reference DC characteristics).

Multiplexed pin descriptions

$A_{18}/\overline{TOUT}$	During RESET, this pin is initialized as A_{18} pin. If either TOC1 or TOC0 bit of the Timer Control Register (TCR) is set to 1, TOUT function is selected. If TOC1 and TOC0 bits are cleared to 0, A_{18} function is selected.
$CKA_0/\overline{DREQ_0}$	During RESET, this pin is initialized as CKA_0 pin. If either DM1 or SM1 in DMA Mode Register (DMODE) is set to 1, $\overline{DREQ_0}$ function is always selected.
$CKA_1/\overline{TEND_0}$	During RESET, this pin is initialized as CKA_1 pin. If CKA1D bit in ASCI control register ch 1 (CNTLA1) is set to 1, $\overline{TEND_0}$ function is selected. If CKA1D bit is set to 0, CKA_1 function is selected.
$RXS/\overline{CTS_1}$	During RESET, this pin is initialized as RXS pin. If CTS1E bit in ASCI status register ch1 (STAT1) is set to 1, $\overline{CTS_1}$ function is selected. If CTS1E bit is set to 0, RXS function is selected.

ARCHITECTURE:

The Z80180 combines a high performance CPU core with a variety of system and I/O resources useful in a broad range of applications. The CPU core consists of five functional blocks: clock generator, bus state controller (including dynamic memory refresh), interrupt controller, memory management unit (MMU), and the central processing unit (CPU). The integrated I/O resources make up the remaining four functional blocks: direct memory access (DMA) control (2 channels), asynchronous serial communications interface (ASCI, 2 channels), programmable reload timers (PRT, 2 channels), and a clock serial I/O (CSIO) channel.

Clock Generator. This logic generates the system clock from either an external crystal or clock input. The external clock is divided by two and provided to both internal and external devices.

Bus State Controller. This logic performs all of the status and bus control activity associated with both the CPU and some on-chip peripherals. This includes wait state timing, reset cycles, DRAM refresh, and DMA bus exchanges.

Interrupt Controller. This block monitors and prioritizes the variety of internal and external interrupts and traps to provide the correct responses from the CPU. To remain compatible with the Z80 CPU, three different interrupt modes are supported.

Memory Management Unit. The MMU allows the user to "map" the memory used by the CPU (logically only 64K) into the 1M Byte addressing range supported by the Z80180. The organization of the MMU object code compatibility with the Z80 CPU while offerring access to an extended memory space. This is accomplished by using an effective "common area - banked area" scheme.

Central Processing Unit. The CPU is microcoded to provide a core that is object code compatible with the Z80 CPU. It also provides a superset of the Z80 instruction set, including 8-bit multiply and divide. This core has been enhanced to allow many of the instructions to execute in fewer clock cycles.

DMA Controller. The DMA controller provides high speed transfers between memory and I/O devices. Transfer operations supported are memory to memory, memory to/from I/O, and I/O to I/O. Transfer modes supported are request, burst, and cycle steal. DMA transfers can access the full 1 Mbyte addressing range with a block length up to 64K bytes, and can cross over 64K boundaries.

Asynchronous Serial Communications Interface (ASCI). The ASCI logic provides two individual full-duplex UARTs. Each channel includes a programmable baud rate generator and modem control signals. The ASCI channels can also support a multiprocessor communications format.

Programmable Reload Timer (PRT). This logic consists of two separate channels, each containig a 16-bit counter (timer) and count reload register. The time base for the counters is derived from the system clock (divided by 20) before reaching the counter. PRT channel 1 provides an optional output to allow for waveform generation.

Clocked Serial I/O (CSIO). The CSIO channel provides a half-duplex serial transmitter and receiver. This channel can be used for simple high-speed data connection to another microprocessor or microcomputer.

OPERATION MODES:

The Z80180 can be configured to operate like the 64180. This is accomplished by allowing the user to have control over the $\overline{M1}$, $\overline{IORQ}$, $\overline{WR}$, and $\overline{RD}$ signals. The Operation Mode Control Register (OMCR) determines the $\overline{M1}$ options; the timing of the $\overline{IORQ}$, RD, and $\overline{WR}$ signals; and the RETI operation.

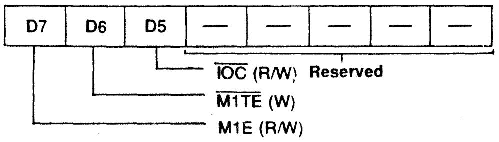


Figure 4. Operation Mode Control Register (I/O Address = 3 EH)

MIE ($\overline{M1}$ Enable): This bit controls the M1 output and is set to a 1 during reset.

When M1E=1, the $\overline{M1}$ output is asserted LOW during the opcode fetch cycle, the $\overline{INT_0}$ acknowledge cycle, and the first machine cycle of the $\overline{NMI}$ acknowledge. This will also cause the $\overline{M1}$ signal to be active during both fetches of the RETI instruction sequence, which may cause corruption of the external interrupt daisy chain. Hence, this bit should be set to 0 for the Z80180. When M1E=0, the $\overline{M1}$ output is normally inactive and asserted LOW only during the refetch of the RETI instruction sequence and during the $\overline{INT_0}$ acknowledge cycle.

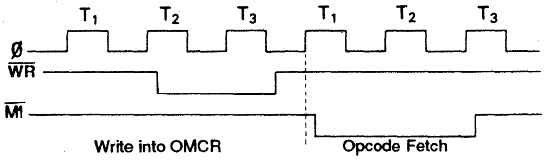


Figure 5. M1 Temporary Enable Timing

$\overline{M1TE}$ (M1 Temporary Enable): This bit controls the temporary assertion of the $\overline{M1}$ signal. It is always read back as a 1 and is set to 1 during reset. This function is used to "arm" the internal interrupt structure of the Z80PIO. When a control word is written to the Z80PIO to enable interrupts, no enable actually takes place until the PIO sees an active $\overline{M1}$ signal. When $\overline{M1TE}$=1, there is no change in the operation of the $\overline{M1}$ signal and M1E controls its function. When $\overline{M1TE}$=0, the $\overline{M1}$ output will be asserted during the next opcode fetch cycle regardless of the state programmed into the M1E bit. This is only momentary (one time) and the user need not reprogram a 1 to disable the function (See Figure 5).

$\overline{IOC}$: this bit controls the timing of the $\overline{IORQ}$ and $\overline{RD}$ signals. It is set to 1 by reset.

When $\overline{IOC}$=1, the $\overline{IORQ}$ and $\overline{RD}$ signals function the same as the Z64180.

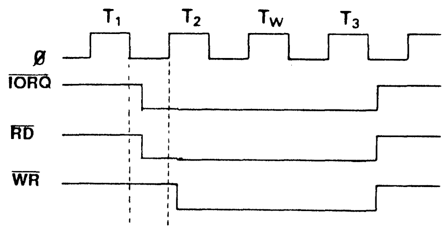


Figure 6. I/O Read and Write Cycles with $\overline{IOC}$ =1

When $\overline{IOC}$=0, the timing of the $\overline{IORQ}$ and $\overline{RD}$ signals match the timing required by the Z80 family of peripherals. The $\overline{IORQ}$ and $\overline{RD}$ signals will go active as a result of the rising edge of T2. This allows the Z80180 to satisfy the setup times required by the Z80 peripherals on those two signals.

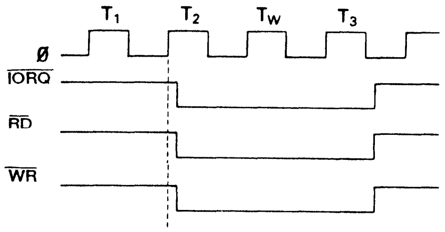


Figure 7. I/O Read and Write Cycles with $\overline{IOC}$ = 0

For the rest of this manual, it is assumed that M1E=0 and $\overline{IOC}$=0. The user must program the Operation Mode Control Register before the first I/O instruction is executed.

TIMING:

This section explains the Z80180 CPU timing for the following operations:

Instruction (op-code) fetch timing.
Operand and data read/write timing.
I/O read/write timing.
Basic instruction (fetch and execute) timing.
RESET timing.
BUSREQ/BUSACK bus exchange timing.

The basic CPU operation consists of one or more "Machine Cycles" (MC). A machine cycle consists of three system clocks, T1, T2, and T3 while accessing memory or I/O, or it consists of one system clock (T1) during CPU internal operations. The system clock is half the frequency of the Crystal oscillator (e.g., an 8 MHz crystal produces 4 MHz or 250 nsec). For interfacing to slow memory or peripherals, optional wait states (Tw) may be inserted between T2 and T3.

Instruction (op-code) Fetch Timing. Fig. 8 shows the instruction (op-code) fetch timing with no wait states. An op-code fetch cycle is externally indicated when the $\overline{M1}$ output pin is LOW.

In the first half of T_1, the address bus (A_0-A_{19}) is driven from the contents of the Program Counter (PC). Note that this is the translated address output of the Z80180 on-chip MMU.

In the second half of T_1, the $\overline{MREQ}$ (Memory Request) and $\overline{RD}$ (Read) signals are asserted LOW, enabling the memory.

The op-code on the data bus is latched at the rising edge of T_3 and the bus cycle terminates at the end of T_3.

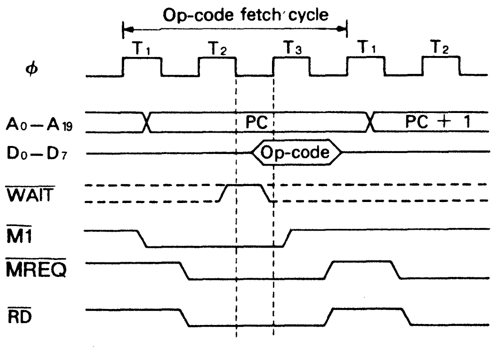


Figure 8. Opcode Fetch timing (Without Wait State)

Fig. 9 illustrates the insertion of wait states (Tw) into the op-code fetch cycle. Wait states (Tw) are controlled by the external $\overline{WAIT}$ input combined with an on-chip programmable wait state generator.

At the falling edge of T_2 the combined $\overline{WAIT}$ input is sampled. If $\overline{WAIT}$ input is asserted LOW, a wait state (Tw) is inserted. The address bus, $\overline{MREQ}$, $\overline{RD}$ and $\overline{M1}$ are held stable during wait states. When the WAIT is sampled inactive HIGH at the falling edge of Tw, the bus cycle enters T_3 and completes at the end of T_3.

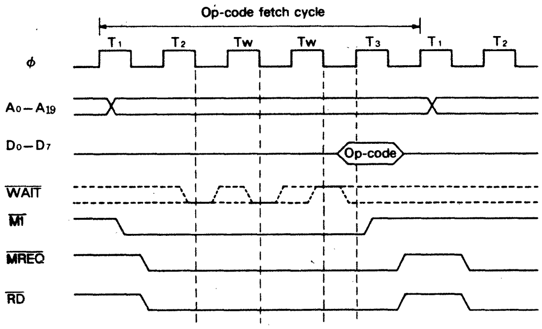


Figure 9. Opcode Fetch Timing (With Wait State)

Operand and Data Read/Write Timing. The instruction operand and data read/write timing differs from op-code fetch timing in two ways. First, the $\overline{M1}$ output is held inactive. Second, the read cycle timing is relaxed by one-half clock cycle since data is latched at the falling edge of T_3.

Instruction operands include immediate data, displacement, and extended addresses, and have the same timing as memory data reads.

During memory write cycles the $\overline{MREQ}$ signal goes active in the second half of T_1. At the end of T_1, the data bus is driven with the write data.

At the start of T_2, the $\overline{WR}$ signal is asserted LOW enabling the memory. $\overline{MREQ}$ and $\overline{WR}$ go inactive in the second half of T3 followed by disabling of the write data on the data bus.

Wait states (Tw) are inserted as previously described for op-code fetch cycles. Fig. 10 illustrates the read/write timing without wait states (Tw), while Fig. 11 illustrates read/write timing with wait states (Tw).

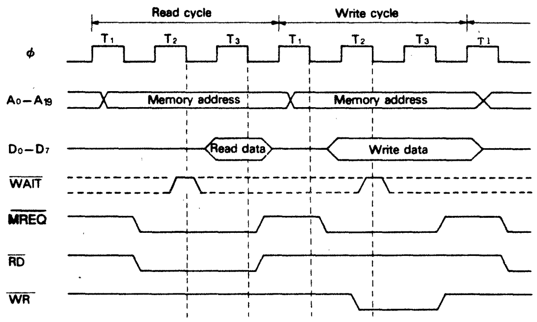


Figure 10. Memory Read/Write Timing (Without Wait State)

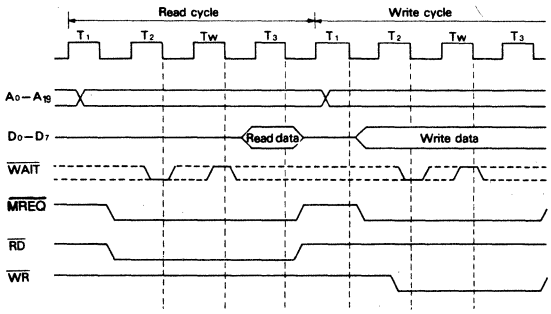


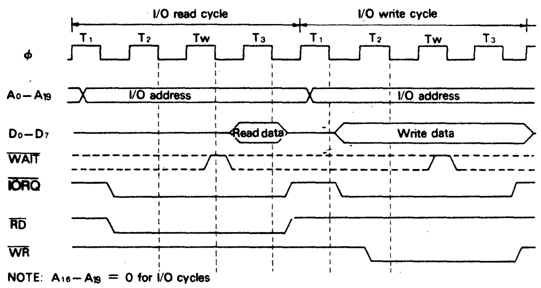
Figure 11. Memory Read/Write Timing (With Wait State)

I/O Read/Write Timing. I/O instructions cause data read/write transfers which differ from memory data transfers in the following three ways:

1. The $\overline{IORQ}$ (I/O Request) signal is asserted LOW instead of the MREQ signal.
2. The 16-bit I/O address is not translated by the MMU.
3. A_{16}-A_{19} are held LOW.

At least one wait state (Tw) is always inserted for I/O read and write cycles (except internal I/O cycles).

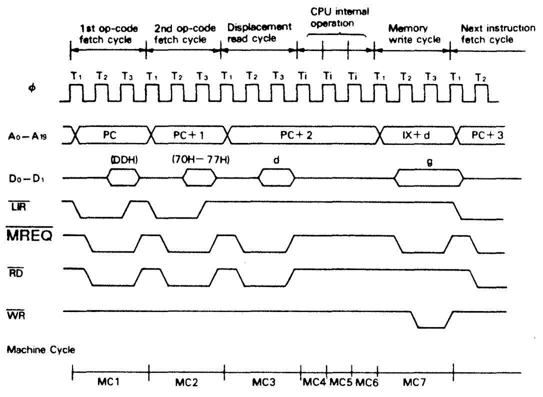
Fig. 12 shows I/O read/write timing with the automatically inserted wait state (Tw).



NOTE: $A_{16}-A_{19}$ = 0 for I/O cycles

Figure 12. I/O Read/Write Timing

Basic Instruction Timing. An instruction may consist of a number of machine cycles including op-code fetch, operand fetch, and data read/write cycles. An instruction may also include cycles for internal processes which make the bus idle.



NOTE: d = displacement
g = register contents

Figure 13. Instruction Timing

The example in Fig. 13 illustrates the bus timing for the data transfer instruction LD (IX+d),g. This instruction moves the contents of a CPU register (g) to the memory location with address computed by adding a signed 8-bit displacement (d) to the contents of an index register (IX).

The instruction cycle starts with the two machine cycles to read the two byte instruction op-code as indicated by $\overline{M1}$ LOW. Next, the instruction operand (d) is fetched.

The external bus is idle while the CPU computes the effective address. Finally, the computed memory location is written with the contents of the CPU register (g).

RESET Timing. Fig. 14 shows the Z80180 hardware RESET timing. If the $\overline{RESET}$ pin is LOW for six or more than six clock cycles, processing is terminated and the Z80180 restarts execution from (logical and physical) address 00000H.

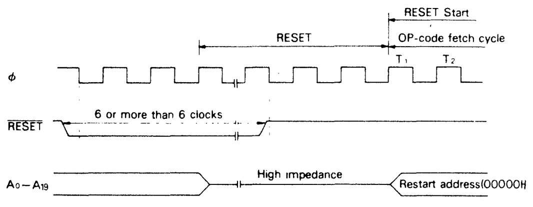


Figure 14. Reset Timing

$\overline{BUSREQ}/\overline{BUSACK}$ Bus Exchange Timing. The Z80180 can coordinate the exchange of control, address and data bus ownership with another bus master. The alternate bus master can request the bus release by asserting the $\overline{BUSREQ}$ (Bus Request) input LOW. After the Z80180 releases the bus, it relinquishes control to the alternate bus master by asserting the $\overline{BUSACK}$ (Bus Acknowledge) output LOW.

The bus may be released by the Z80180 at the end of each machine cycle. In this context, a machine cycle consists of a minimum of 3 clock cycles (more if wait states are inserted) for op-code fetch, memory read/write, and I/O read/write cycles. Except for these cases, a machine cycle corresponds to one clock cycle.

When the bus is released, the address (A_0-A_{19}), data (D_0-D_7), and control ($\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, and $\overline{WR}$) signals are placed in the high impedance state.

Note that dynamic RAM refresh is not performed when the Z80180 has released the bus. The alternate bus master must provide dynamic memory refreshing if the bus is released for long periods of time.

Fig. 15 illustrates $\overline{BUSREQ}/\overline{BUSACK}$ bus exchange during a memory read cycle. Fig. 16 illustrates bus exchange when the bus release is requested during a Z80180 CPU internal operation. $\overline{BUSREQ}$ is sampled at the falling edge of the system clock prior to T_3, T_i and Tx (BUS RELEASE state). If $\overline{BUSREQ}$ is asserted LOW at the falling edge of the clock state prior to Tx, another Tx is executed.

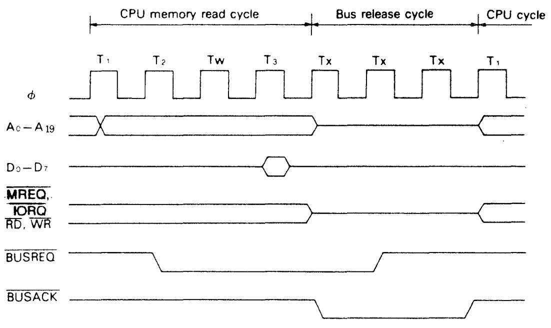


Figure 15. Bus Exchange Timing

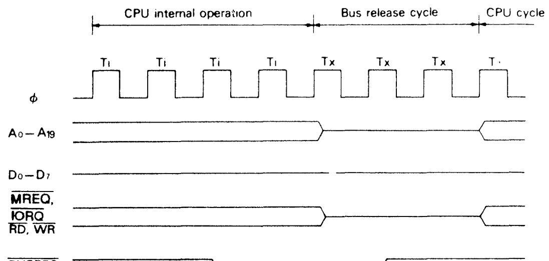


Figure 16. Bus Exchange Timing

WAIT State Generator

To ease interfacing with slow memory and I/O devices, the Z80180 uses wait states (Tw) to extend bus cycle timing. A wait state(s) is inserted based on the combined (logical OR) state of the external $\overline{WAIT}$ input and an internal programmable wait state (Tw) generator. Wait states (Tw) can be inserted in both CPU execution and DMA transfer cycles.

When the external $\overline{WAIT}$ input is asserted LOW, wait state(s) (Tw) are inserted between T_2 and T_3 to extend the bus cycle duration. The $\overline{WAIT}$ input is sampled at the falling edge of the system clock in T_2 or Tw. If the $\overline{WAIT}$ input is asserted LOW at the falling edge of the system clock in Tw, another Tw is inserted into the bus cycle. Note that $\overline{WAIT}$ input transitions must meet specified set-up and hold times. This can easily be accomplished by externally synchronizing $\overline{WAIT}$ input transitions with the rising edge of the system clock.

Dynamic RAM refresh is not performed during wait states (Tw) and thus system designs which use the automatic refresh function must consider the affects of the occurrence and duration of wait states (Tw). Figure 17 shows WAIT timing.

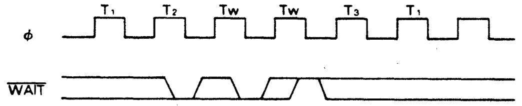


Figure 17. $\overline{WAIT}$ Timing

Programmable Wait State Insertion. In addition to the $\overline{WAIT}$ input, wait states (Tw) can also be inserted by program using the Z80180 on-chip wait state generator. Wait state (Tw) timing applies for both CPU execution and on-chip DMAC cycles.

By programming the four significant bits of the DMA/WAIT Control Register (DCNTL) the number of wait states, (Tw) automatically inserted in memory and I/O cycles, can be separately specified.

HALT and Low Power Operation Modes

The Z80180 can operate in 4 different modes. HALT mode, IOSTOP mode and 2 low power operation modes - SLEEP and SYSTEM STOP. Note that in all operating modes, the basic CPU clock (XTAL, EXTAL) must remain active.

HALT mode. HALT mode is entered by execution of the HALT instruction (op-code = 76H) and has the following characteristics.

(1) The internal CPU clock remains active.

(2) All internal and external interrupts can be received.

(3) Bus exchange ($\overline{BUSREQ}$ and $\overline{BUSACK}$) can occur.

(4) Dynamic RAM refresh cycle ($\overline{RFSH}$) insertion continues at the programmed interval.

(5) I/O operations (ASCI, CSI/O and PRT) continue.

(6) The DMAC can operate.

(7) The $\overline{HALT}$ output pin is asserted LOW.

(8) The external bus activity consists of repeated "dummy" fetches of the op-code following the HALT instruction.

Essentially, the Z80180 operates normally in HALT mode, except that instruction execution is stopped.

HALT mode can be exited in the following two ways.

RESET Exit from HALT mode. If the $\overline{RESET}$ input is asserted LOW for at least 6 clock cycles, HALT mode is exited and the normal RESET sequence (restart at address 00000H) is initiated.

Interrupt Exit from HALT mode. When an internal or external interrupt is generated, HALT mode is exited and the normal interrupt response sequence is initiated.

If the interrupt source is masked (individually by enable bit, or globally by IEF1 state), the Z80180 remains in HALT mode. However, $\overline{NMI}$ interrupt will initiate the normal $\overline{NMI}$ interrupt response sequence independent of the state of IEF_1.

HALT timing is shown in Fig 18.

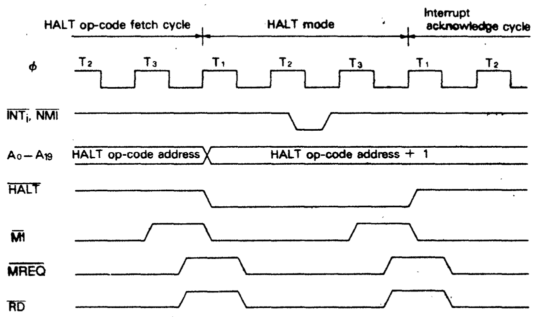


Figure 18. HALT Timing

SLEEP mode. SLEEP mode is entered by execution of the 2 byte SLP instruction. SLEEP mode has the following characteristics.

(1) The internal CPU clock stops, reducing power consumption.

(2) The internal crystal oscillator does not stop.

(3) Internal and external interrupt inputs can be received.

(4) DRAM refresh cycles stop.

(5) I/O operations using on-chip peripherals continue.

(6) The internal DMAC stop.

(7) $\overline{\text{BUSREQ}}$ can be received and acknowledged.

(8) Address outputs go HIGH and all other control signal output become inactive HIGH.

(9) Data Bus, 3-state.

SLEEP mode is exited in one of two ways as shown below.

RESET Exit from SLEEP mode. If the $\overline{\text{RESET}}$ input is held LOW for at least 6 clock cycles, it will exit SLEEP mode and begin the normal RESET sequence with execution starting at address (logical and physical) 00000H.

Interrupt Exit from SLEEP mode. The SLEEP mode is exited by detection of an external (NMI, INT_0-INT_2) or internal (ASCI, CSI/O, PRT) interrupt.

In case of $\overline{\text{NMI}}$, SLEEP Mode is exited and the CPU begins the normal $\overline{\text{NMI}}$ interrupt response sequence.

In the case of all other interrupts, the interrupt response depends on the state of the global interrupt enable flag (IEF_1) and the individual interrupt source enable bit.

If the individual interrupt condition is disabled by the corresponding enable bit, occurrence of that interrupt is ignored and the CPU remains in the SLEEP state.

Assuming the individual interrupt condition is enabled, the response to that interrupt depends on the global interrupt enable flag (IEF_1). If interrupts are globally enabled (IEF_1=1) and an individually enabled interrupt occurs, SLEEP mode is exited and the appropriate normal interrupt response sequence is executed.

If interrupts are globally disabled (IEF_1=0) and an individually enabled interrupt occurs, SLEEP mode is exited and instruction execution begins with the instruction following the SLP instruction. Note that this provides a technique for synchronization with high speed external events without incurring the latency imposed by an interrupt response sequence.

Figure 19 shows SLEEP timing.

IOSTOP mode. IOSTOP mode is entered by setting the IOSTOP bit of the I/O Control Register (ICR) to 1. In this case, on-chip I/O (ASCI, CSI/O, PRT) stops operating. However, the CPU continues to operate. Recovery from IOSTOP mode is by resetting the IOSTOP bit in ICR to 0.

SYSTEM STOP mode. SYSTEM STOP mode is the combination of SLEEP and IOSTOP modes. SYSTEM STOP mode is entered by setting the IOSTOP bit in ICR to 1 followed by execution of the SLP instruction. In this mode, on-chip I/O and CPU stop operating, reducing power consumption. Recovery from SYSTEM STOP mode is the same as recovery from SLEEP mode, noting that internal I/O sources (disabled by IOSTOP) cannot generate a recovery interrupt.

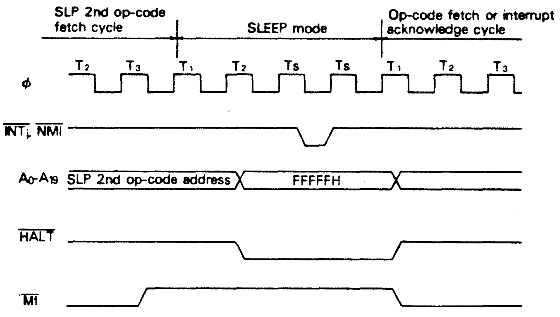


Figure 19. SLEEP Timing

Trap and Interrupts

The Z80180 CPU has twelve interrupt sources, 4 external and 8 internal, with fixed priority. (Reference Figure 20).

Higher Priority	(1)	TRAP (Undefined Op-code Trap)	Internal Interrupt
	(2)	$\overline{\text{NMI}}$ (Non Maskable Interrupt)	
	(3)	$\overline{\text{INT}_0}$ (Maskable Interrupt Level 0)	External Interrupt
	(4)	$\overline{\text{INT}_1}$ (Maskable Interrupt Level 1)	
	(5)	$\overline{\text{INT}_2}$ (Maskable Interrupt Level 2)	
	(6)	Timer 0	
	(7)	Timer 1	
	(8)	DMA channel 0	
	(9)	DMA channel 1	Internal Interrupt
	(10)	Clocked Serial I/O Port	
Lower Priority	(11)	Asynchronous SCI channel 0	
	(12)	Asynchronous SCI channel 1	

Figure 20. Interrupt Sources

TRAP Interrupt. The Z80180 generates a non-maskaable TRAP interrupt when an undefined op-code fetch occurs. This feature can be used to increase software reliability, implement an "extended" instruction set, or both. TRAP may occur during op-code fetch cycles and also if an undefined op-code is fetched during the interrupt acknowledge cycle for INT_0 when Mode 0 is used.

When a TRAP interrupt occurs the Z80180 operates as follows.

(1) The TRAP bit in the Interrupt TRAP/Control (ITC) register is set to 1.

(2) The current PC (Program Counter) value, reflecting location of the undefined op-code, is saved on the stack.

(3) The Z80180 vectors to logical address 0. Note that if logical address 0000H is mapped to physical address 00000H, the vector is the same as for RESET. In this case, testing the TRAP bit in ITC will reveal whether the restart at physical address 00000H was caused by RESET or TRAP.

External Interrupts. The Z80180 has four external hardware interrupt inputs.

(1) $\overline{\text{NMI}}$ - Non-maskable Interrupt
(2) $\overline{\text{INT}_0}$ - Maskable Interrupt Level 0
(3) $\overline{\text{INT}_1}$ - Maskable Interrupt Level 1
(4) $\overline{\text{INT}_2}$ - Maskable Interrupt Level 2

$\overline{\text{NMI}}$, $\overline{\text{INT}_1}$ and $\overline{\text{INT}_2}$ have fixed interrupt response modes. $\overline{\text{INT}_0}$ has 3 different software programmable interrupt response modes - Mode 0, Mode 1 and Mode 2.

$\overline{\text{NMI}}$ - Non-Maskable Interrupt. The $\overline{\text{NMI}}$ interrupt input is edge sensitive and cannot be masked by software. When $\overline{\text{NMI}}$ is detected, the Z80180 operates as follows.

(1) DMAC operation is suspended by the clearing of the DME (DMA Main Enable) bit in DCNTL.

(2) The PC is pushed onto the stack.

(3) The contents of IEF_1 are copied to IEF_2. This saves the interrupt reception state that existed prior to $\overline{\text{NMI}}$.

(4) IEF_1 is cleared to 0. This disables all external and internal maskable interrupts (i.e. all interrupts except $\overline{\text{NMI}}$ and TRAP).

(5) Execution commences at logical address 0066H.

The last instruction of an $\overline{\text{NMI}}$ service routine should be RETN (Return from Non-maskable Interrupt). This restores the stacked PC, allowing the interrupted program to continue.

$\overline{\text{INT}_0}$ - Maskable Interrupt Level 0
The next highest priority external interrupt after $\overline{\text{NMI}}$ is $\overline{\text{INT}_0}$. $\overline{\text{INT}_0}$ is sampled at the falling edge of the clock state prior to T_3 or T1 in the last machine cycle. If $\overline{\text{INT}_0}$ is asserted LOW at the falling edge of the clock state prior to T_3 or T_1 in the last machine cycle, $\overline{\text{INT}_0}$ is accepted. The interrupt is masked if either the IEF1 flag or the ITE0 (Interrupt Enable 0) bit in ITC are reset to 0.

The $\overline{\text{INT}_0}$ interrupt is unique in that 3 programmable interrupt response modes are available - Mode 0, Mode 1 and Mode 2. The specific mode is selected with the IM 0, IM 1 and IM 2 (Set Interrupt Mode) instructions. During $\overline{\text{RESET}}$, the Z80180 is initialized to use Mode 0 for INT0. The 3 interrupt response modes for INT_0 are:

(1) Mode 0 - Instruction fetch from data bus.

(2) Mode 1 - Restart at logical address 0038H.

(3) Mode 2 - Low byte vector table address fetch from data bus.

$\overline{\text{INT}_0}$ Mode 0
During the interrupt acknowledge cycle, an instruction is fetched from the data bus (D0-D7) at the rising edge of T3. Often, this instruction is one of the eight single byte RST (RESTART) instructions which stack the PC and restart execution at a fixed logical address. However, multibyte instructions can be processed if the interrupt acknowledging device can provide a multibyte response. Unlike all other interrupts, the PC is not automatically stacked.

Note that TRAP interrupt will occur if an invalid instruction is fetched during Mode 0 interrupt acknowledge.

$\overline{\text{INT}_0}$ Mode 1
When $\overline{\text{INT}_0}$ is received, the PC is stacked and instruction execution restarts at logical address 0038H. Both IEF1 and IEF2 flags are reset to 0, disabling all maskable interrupts. The interrupt service routine should normally terminate with the EI (Enable Interrupts) instruction followed by the RETI (Return from Interrupt) instruction, to reenable the interrupts.

$\overline{\text{INT}_0}$ Mode 2
This method determines the restart address by reading the contents of a table residing in memory. The vector table consists of up to 128 two-byte restart addresses stored in low byte, high byte order.

The vector table address is located on 256 byte boundaries in the 64K byte logical address space programmed in the 8-bit Interrupt Vector Register (I).

During the $\overline{\text{INT}_0}$ Mode 2 acknowledge cycle, the low-order 8 bits of the vector is fetched from the data bus at the rising edge of T3 and the CPU acquires the 16-bit vector.

Next, the PC is stacked. Finally, the 16-bit restart address is fetched from the vector table and execution begins at that address.

Note that external vector acquisition is indicated by both $\overline{M1}$ and $\overline{IORQ}$ LOW. Two wait states (Tw) are automatically inserted for external vector fetch cycles.

$\overline{INT_1}$, $\overline{INT_2}$

The operation of external interrupts $\overline{INT_1}$ and $\overline{INT_2}$ is a vector mode similar to $\overline{INT_0}$ Mode 2. The difference is that $\overline{INT_1}$ and $\overline{INT_2}$ generate the low-order byte of vector table address using the IL (Interrupt Vector Low) register rather than fetching it from the data bus. This is also the interrupt response sequence used for all internal interrupts (except TRAP).

Internal Interrupts. Internal interrupts (except TRAP) ise the same vectored response mode as INT1 and INT2. Internal interrupts are globally masked by IEF1 = 0. Individual internal interrupts are enabled/disabled by programming each individual I/O (PRT, DMAC, CSI/O, ASCI) control register. The lower vector of INT_1, INT_2 and internal interrupt are summarized in Table 2.

Interrupt Source	Priority	IL			Fixed Code				
		b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
$\overline{INT_1}$	Highest	•	•	•	0	0	0	0	0
$\overline{INT_2}$	↑	•	•	•	0	0	0	1	0
PRT channel 0		•	•	•	0	0	1	0	0
PRT channel 1		•	•	•	0	0	1	1	0
DMA channel 0		•	•	•	0	1	0	0	0
DMA channel 1		•	•	•	0	1	0	1	0
CSI/O		•	•	•	0	1	1	0	0
ASCI channel 0	↓	•	•	•	0	1	1	1	0
ASCI channel 1	Lowest	•	•	•	1	0	0	0	0

• Programmable

Table 2. Vector Table

RETI Instruction Sequence:

When the EDH/4DH sequence is fetched by the Z80180, it is recognized as the RETI instruction sequence. The Z80180 will then refetch the RETI instruction with 4 T-states in the EDH cycle to allow the Z80 peripherals time to decode that cycle (See Figure 21). This allows the internal interrupt structure of the peripheral to properly decode the instruction and behave accordingly.

The M1E bit of the Operation Mode Control Register (OMCR) should be set to 0 so that $\overline{M1}$ signal is active only during the refetch of the RETI instruction sequence. This is the desired operation when Z80 peripherals are connected to the Z80180.

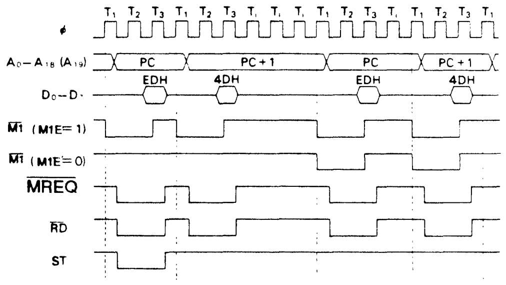


Figure 21. RETI Instruction Sequence

The RETI instruction takes 22 T-states and 10 machine cycles.

Interrupt Control Registers and Flags. The Z80180 has three registers and two flags which are associated with interrupt processing.

Function	Name	Access Method
(1) Interrupt Vector High	I	LD A,I and LD I, A instructions
(2) Interrupt Vector Low	IL	I/O instruction (addr=33H)
(3) Interrupt/Trap Control	ITC	I/O instruction (addr=34H)
(4) Interrupt Enable Flag 1,2	IEF_1,IEF_2	EI and DI

Interrupt Enable/Disable Operation

Two flags, IEF_1 and IEF_2, are used to signal the Z80180 CPU interrupt status. IEF_1 controls the overall enabling and disabling of all internal and external maskable interrupts (i.e. all interrupts except NMI and TRAP).

If $IEF_1 = 0$, all maskable interrupts are disabled. IEF_1 can be reset to 0 by the DI (Disable Interrupts) instruction and set to 1 by the EI (Enable Interrupts) instruction.

The purpose of IEF_2 is to correctly manage the occurrence of $\overline{NMI}$. During $\overline{NMI}$, the prior interrupt reception state is saved and all maskable interrupts are automatically disabled (IEF_1 copied to IEF_2 and then IEF_1 cleared to 0). At the end of the $\overline{NMI}$ interrupt service routine, execution of the RETN (Return from Non-maskable Interrupt) will automatically restore the interrupt receiving state (by copying IEF2 to IEF1) prior to the occurrence of $\overline{NMI}$.

IEF_2 state can be reflected in the P/V bit of the CPU Status Register by executing LD A, I or LD A, R instructions.

CPU Operation	IEF_1	IEF_2	REMARKS
RESET	0	0	Inhibits the interrupt except $\overline{NMI}$ and TRAP.
$\overline{NMI}$	0	IEF	Copies the contents of IEF_1 to IEF_2.
RETN	IEF_2	not affected	Returns from the $\overline{NMI}$ service routine.
Interrupt except $\overline{NMI}$ and TRAP	0	0	Inhibits the interrupt except $\overline{NMI}$ and TRAP.
RETI	not affected	not affected	
TRAP	not affected	not affected	
EI	1	1	
DI	0	0	
LD A, I	not affected	not affected	Transfers the contents of IEF_2 to P/V flag.
LD A, R	not affected	not affected	Transfers the contents of IEF_2 to P/V flag

Table 3. State of IEF_1 and IEF_2

Internal I/O Registers

The Z80180 internal I/O Registers occupy 64 I/O addresses (including reserved addresses). These registers access the internal I/O modules (ASCI, CSI/O, PRT) and control functions (DMAC, DRAM refresh, interrupts, wait state generator, MMU and I/O relocation).

To avoid address conflicts with external I/O, the Z80180 internal I/O addresses can be relocated on 64 byte boundaries within the bottom 256 bytes of the 64K byte I/O address space.

By programming IOA7 and IOA6 in the I/O control register, internal I/O register addresses are relocatable within ranges from 0000H to 00FFH in the I/O address space.

REGISTER	MNEMONICS	ADDRESS	REMARKS
ASCI Control Register A Channel 0	CNTLA0	0 0	

bit	MPE	RE	TE	$\overline{\text{RTS0}}$	MPBR/ EFR	MOD2	MOD1	MOD0
during RESET	0	0	0	1	invalid	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- MOD2, MOD1, MOD0 — MODE Selection
- MPBR/EFR — Multi Processor Bit Receive/ Error Flag Reset
- $\overline{\text{RTS0}}$ — Request To Send
- TE — Transmit Enable
- RE — Receive Enable
- MPE — Multi Processor Enable

REGISTER	MNEMONICS	ADDRESS	REMARKS
ASCI Control Register A Channel 1	CNTLA1	0 1	

bit	MPE	RE	TE	CKA1D	MPBR/ EFR	MOD2	MOD1	MOD0
during RESET	0	0	0	1	invalid	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- MOD2, MOD1, MOD0 — MODE Selection
- MPBR/EFR — Multi Processor Bit Receive/ Error Flag Reset
- CKA1D — CKA1 Disable
- TE — Transmit Enable
- RE — Receive Enable
- MPE — Multi Processor Enable

MOD2, 1, 0	
0 0 0	Start + 7 bit Data + 1 Stop
0 0 1	Start + 7 bit Data + 2 Stop
0 1 0	Start + 7 bit Data + Parity + 1 Stop
0 1 1	Start + 7 bit Data + Parity + 2 Stop
1 0 0	Start + 8 bit Data + 1 Stop
1 0 1	Start + 8 bit Data + 2 Stop
1 1 0	Start + 8 bit Data + Parity + 1 Stop
1 1 1	Start + 8 bit Data + Parity + 2 Stop

REGISTER	MNEMONICS	ADDRESS	REMARKS
ASCI Control Register B Channel 0	CNTLB0	0 2	

bit	MPBT	MP	$\overline{\text{CTS}}$/ PS	PEO	DR	SS2	SS1	SS0
during RESET	invalid	0	•	0	0	1	1	1
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- SS2, SS1, SS0 — Clock Source and Speed Select
- DR — Divide Ratio
- PEO — Parity Even or Odd
- $\overline{\text{CTS}}$/PS — Clear To Send/Prescale
- MP — Multi Processor
- MPBT — Multi Processor Bit Transmit

• $\overline{\text{CTS}}$: Depending on the condition of $\overline{\text{CTS}}$ Pin.
PS : Cleared to 0.

(to be continued)

REGISTER	MNEMONICS	ADDRESS	REMARKS
ASCI Control Register B Channel 1 : CNTLB1		0 3	

bit	MPBT	MP	$\overline{CTS}$/ PS	PEO	DR	SS2	SS1	SS0
during RESET	invalid	0	0	0	0	1	1	1
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- SS2, SS1, SS0 — Clock Source and Speed Select
- DR — Divide Ratio
- PEO — Parity Even or Odd
- $\overline{CTS}$/PS — Clear To Send/Prescale
- MP — Multi Processor
- MPBT — Multi Processor Bit Transmit

General divide ratio	PS=0 (divide ratio=10)		PS=1 (divide ratio=30)	
SS2,1,0	DR=0 (×16)	DR=1 (×64)	DR=0 (×16)	DR=1 (×64)
000	ϕ÷ 160	ϕ÷ 640	ϕ÷ 480	ϕ÷ 1920
001	÷ 320	÷ 1280	÷ 960	÷ 3840
010	÷ 640	÷ 2560	÷ 1920	÷ 7680
011	÷ 1280	÷ 5120	÷ 3840	÷ 15360
100	÷ 2560	÷ 10240	÷ 7680	÷ 30720
101	÷ 5120	÷ 20480	÷ 15360	÷ 61440
110	÷ 10240	÷ 40960	÷ 30720	÷ 122880
111	External clock (frequency < ϕ ÷ 40)			

REGISTER	MNEMONICS	ADDRESS	REMARKS
ASCI Status Register Channel 0 : STAT0		0 4	

bit	RDRF	OVRN	PE	FE	RIE	$\overline{DCD0}$	TDRE	TIE
during RESET	0	0	0	0	0	*	**	0
R/W	R	R	R	R	R/W	R	R	R/W

- TIE — Transmit Interrupt Enable
- TDRE — Transmit Data Register Empty
- $\overline{DCD0}$ — Data Carrier Detect
- RIE — Receive Interrupt Enable
- FE — Framing Error
- PE — Parity Error
- OVRN — Over Run Error
- RDRF — Receive Data Register Full

* $\overline{DCD_0}$: Depending on the condition of $\overline{DCD_0}$ Pin.

**

$\overline{CTS_0}$ Pin	TDRE
L	1
H	0

REGISTER	MNEMONICS	ADDRESS	REMARKS
ASCI Status Register Channel 1 : STAT1		0 5	

bit	RDRF	OVRN	PE	FE	RIE	CTS1E	TDRE	TIE
during RESET	0	0	0	0	0	0	1	0
R/W	R	R	R	R	R/W	R/W	R	R/W

- TIE — Transmit Interrupt Enable
- TDRE — Transmit Data Register Empty
- CTS1E — $\overline{CTS1}$ Enable
- RIE — Receive Interrupt Enable
- FE — Framing Error
- PE — Parity Error
- OVRN — Over Run Error
- RDRF — Receive Data Register Full

(to be continued)

REGISTER	MNEMONICS	ADDRESS	REMARKS
ASCI Transmit Data Register Channel 0	: TDR0	0 6	
ASCI Transmit Data Register Channel 1	: TDR1	0 7	
ASCI Receive Data Register Channel 0	: TSR0	0 8	
ASCI Receive Data Register Channel 1	: TSR1	0 9	
CSI/O Control Register	: CNTR	0 A	

bit	EF	EIE	RE	TE	–	SS2	SS1	SS0
during RESET	0	0	0	0	1	1	1	1
R/W	R	R/W	R/W	R/W		R/W	R/W	R/W

Speed Select (SS2, SS1, SS0)
Transmit Enable (TE)
Receive Enable (RE)
End Interrupt Enable (EIE)
End Flag (EF)

SS2,1,0	Baud Rate	SS2,1,0	Baud Rate
000	φ÷ 20	100	φ÷ 320
001	÷ 40	101	÷ 640
010	÷ 80	110	÷1280
011	÷160	111	External (frequency < ÷ 20)

REGISTER	MNEMONICS	ADDRESS	REMARKS
CSI/O Transmit/Receive Data Register	: TRDR	0 B	
Timer Data Register Channel 0L	: TMDR0L	0 C	
Timer Data Register Channel 0H	: TMDR0H	0 D	
Timer Reload Register Channel 0L	: RLDR0L	0 E	
Timer Reload Register Channel 0H	: RLDR0H	0 F	
Timer Control Register	: TCR	1 0	

bit	TIF1	TIF0	TIE1	TIE0	TOC1	TOC0	TDE1	TDE0
during RESET	0	0	0	0	0	0	0	0
R/W	R	R	R/W	R/W	R/W	R/W	R/W	R/W

Timer Down Count Enable 1,0 (TDE1, TDE0)
Timer Output Control 1,0 (TOC1, TOC0)
Timer Interrupt Enable 1,0 (TIE1, TIE0)
Timer Interrupt Flag 1,0 (TIF1, TIF0)

TOC1,0	A_{18}/TOUT
00	Inhibited
01	Toggle
10	0
11	1

(to be continued)

REGISTER	MNEMONICS	ADDRESS	REMARKS
Timer Data Register Channel 1L	: TMDR1L	1 4	
Timer Data Register Channel 1H	: TMDR1H	1 5	
Timer Reload Register Channel 1L	: RLDR1L	1 6	
Timer Reload Register Channel 1H	: RLDR1H	1 7	
Free Running Counter	: FRC	1 8	read only
DMA Source Address Register Channel 0L	: SAR0L	2 0	
DMA Source Address Register Channel 0H	: SAR0H	2 1	
DMA Source Address Register Channel 0B	: SAR0B	2 2	Bits 0-2 (3) are used for SAR0B. A_{19}*, A_{18}, A_{17}, A_{16} | DMA Transfer Request X X 0 0 | $\overline{DREQ_0}$ (external) X X 0 1 | RDR0 (ASCI0) X X 1 0 | RDR1 (ASCI1) X X 1 1 | Not Used
DMA Destination Address Register Channel 0L	: DAR0L	2 3	
DMA Destination Address Register Channel 0H	: DAR0H	2 4	
DMA Destination Address Register Channel 0B	: DAR0B	2 5	Bits 0-2 (3) are used for DAR0B. A_{19}*, A_{18}, A_{17}, A_{16} | DMA Transfer Request X X 0 0 | $\overline{DREQ_0}$ (external) X X 0 1 | TDR0 (ASCI0) X X 1 0 | TDR1 (ASCI1) X X 1 1 | Not Used
DMA Byte Count Register Channel 0L	: BCR0L	2 6	
DMA Byte Count Register Channel 0H	: BCR0H	2 7	
DMA Memory Address Register Channel 1L	: MAR1L	2 8	
DMA Memory Address Register Channel 1H	: MAR1H	2 9	
DMA Memory Address Register Channel 1B	: MAR1B	2 A	Bits 0-2 (3) are used for MAR1B.
DMA I/O Address Register Channel 1L	: IAR1L	2 B	
DMA I/O Address Register Channel 1H	: IAR1H	2 C	

(to be continued)

* In the R1 and Z Mask, these DMAC registers are expanded from 4 bits to 3 bits in the package version of CP-68 and FP-80.

REGISTER	MNEMONICS	ADDRESS	REMARKS
DMA Byte Count Register Channel 1L	: BCR1L	2 E	
DMA Byte Count Register Channel 1H	: BCR1H	2 F	
DMA Status Register	: DSTAT	3 0	see below
DMA Mode Register	: DMODE	3 1	see below

DMA Status Register : DSTAT (3 0)

bit	DE1	DE0	$\overline{\text{DWE1}}$	$\overline{\text{DWE0}}$	DIE1	DIE0	–	DME
during RESET	0	0	1	1	0	0	1	0
R/W	R/W	R/W	W	W	R/W	R/W		R

DME: DMA Master Enable

DIE1, DIE0: DMA Interrupt Enable 1,0

$\overline{\text{DWE1}}$, $\overline{\text{DWE0}}$: DMA Enable Bit Write Enable 1,0

DE1, DE0: DMA Enable ch 1,0

DMA Mode Register : DMODE (3 1)

bit	–	–	DM1	DM0	SM1	SM0	MMOD	–
during RESET	1	1	0	0	0	0	0	1
R/W			R/W	R/W	R/W	R/W	R/W	

MMOD: Memory MODE Select

SM1, SM0: Ch 0 Source Mode 1,0

DM1, DM0: Ch 0 Destination Mode 1, 0

DM1, 0	Destination	Address
0 0	M	DAR0+ 1
0 1	M	DAR0− 1
1 0	M	DAR0 fixed
1 1	I/O	DAR0 fixed

SM1, 0	Source	Address
0 0	M	SAR0+ 1
0 1	M	SAR0− 1
1 0	M	SAR0 fixed
1 1	I/O	SAR0 fixed

MMOD	Mode
0	Cycle Steal Mode
1	Burst Mode

(to be continued)

REGISTER	MNEMONICS	ADDRESS	REMARKS
DMA/WAIT Control Register	: DCNTL	3 2	

bit	MWI1	MWI0	IWI1	IWI0	DMS1	DMS0	DIM1	DIM0
during RESET	1	1	1	1	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

DMA Ch 1 I/O Memory Mode Select

$\overline{\text{DREQi}}$ Select, i = 1,0

I/O Wait Insertion

Memory Wait Insertion

MWI1,0	The number of wait states	IWI1,0	The number of wait states
0 0	0	0 0	0
0 1	1	0 1	2
1 0	2	1 0	3
1 1	3	1 1	4

DMSi	Sense
1	Edge sense
0	Level sense

DIM1,0	Transfer Mode	Address Increment/Decrement	
0 0	M→I/O	MAR1 + 1	IAR1 fixed
0 1	M→I/O	MAR1 − 1	IAR1 fixed
1 0	I/O→M	IAR1 fixed	MAR1 + 1
1 1	I/O→M	IAR1 fixed	MAR1 − 1

REGISTER	MNEMONICS	ADDRESS
Interrupt Vector Low Register	: IL	3 3

bit	IL7	IL6	IL5	–	–	–	–	–
during RESET	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W					

Interrupt Vector Low

REGISTER	MNEMONICS	ADDRESS
INT/TRAP Control Register	: ITC	3 4

bit	TRAP	UFO	–	–	–	ITE2	ITE1	ITE0
during RESET	0	0	1	1	1	0	0	1
R/W	R/W	R				R/W	R/W	R/W

$\overline{\text{INT}}$ Enable 2,1,0

Undefined Fetch Object

TRAP

REGISTER	MNEMONICS	ADDRESS
Refresh Control Register	: RCR	3 6

bit	REFE	REFW	–	–	–	–	CYC1	CYC0
during RESET	1	1	1	1	1	1	0	0
R/W	R/W	R/W					R/W	R/W

Cycle Select

Refresh Wait State

Refresh Enable

CYC1,0	Interval of Refresh Cycle
0 0	10 States
0 1	20
1 0	40
1 1	80

(to be continued)

REGISTER	MNEMONICS	ADDRESS	REMARKS							
MMU Common Base Register	: CBR	3 8	bit	CB7*	CB6	CB5	CB4	CB3	CB2	CB1	CB0
			during RESET	0	0	0	0	0	0	0	0
			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
				MMU Common Base Register							
MMU Bank Base Register	: BBR	3 9	bit	BB7*	BB6	BB5	BB4	BB3	BB2	BB1	BB0
			during RESET	0	0	0	0	0	0	0	0
			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
				MMU Bank Base Register							
MMU Common/Bank Area Register	: CBAR	3 A	bit	CA3	CA2	CA1	CA0	BA3	BA2	BA1	BA0
			during RESET	1	1	1	1	0	0	0	0
			R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
				MMU Common Area Register				MMU Bank Area Register			
Operation Mode Control Register	: OMCR	3 E	bit	M1E	$\overline{\text{M1TE}}$	$\overline{\text{IOC}}$	–	–	–	–	–
			during RESET	1	1	1	1	1	1	1	1
			R/W	R/W	W	R/W					
				$\overline{\text{M1}}$ Enable	$\overline{\text{M1}}$ Temporary Enable	I/O Compatibility					
I/O Control Register	: ICR	3 F	bit	IOA7	IOA6	IOSTP	–	–	–	–	–
			during RESET	0	0	0	1	1	1	1	1
			R/W	R/W	R/W	R/W					
				I/O Address		I/O Stop					

These MMU registers are expanded from 7 bits to 8 bits in the PLCC package

Memory Management Unit (MMU)

The Z80180 has an on-chip MMU which performs the translation of the CPU 64K byte (16-bit addresses 0000H to FFFFH) logical memory address space into a 1024K byte (20-bit addresses 00000H to FFFFFH) physical memory address space. Address translation occurs internally in parallel with other CPU operation.

Logical Address Spaces. The 64K byte CPU logical address space is interpreted by the MMU as consisting of up to three separate logical address areas, Common Area 0, Bank Area, and Common Area 1.

As shown in Fig.22, a variety of logical memory configurations are possible. The boundaries between the Common and Bank Areas can be programmed with 4K byte resolution.

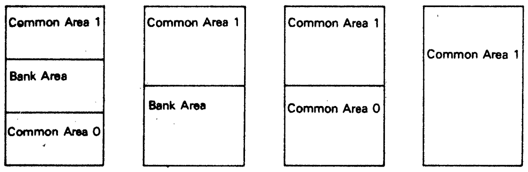


Figure 22. Logical Address Mapping Examples

Whether address translation takes place depends on the type of CPU cycle as follows.

(1) Memory Cycles
Address Translation occurs for all memory access cycles including instruction and operand fetches, memory data reads and writes, hardware interrupt vector fetch, and software interrupt restarts.

(2) I/O Cycles
The MMU is logically bypassed for I/O cycles. The 16-bit logical I/O address space corresponds directly with the 16-bit physical I/O address space. The four high-order bits (A16-A19) of the physical address are always 0 during I/O cycles.

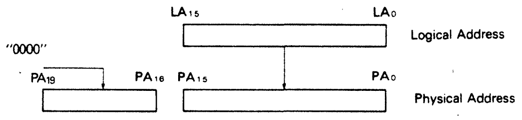


Figure 23. I/O Address Translation

(3) DMA Cycles
When the Z80180 on-chip DMAC is using the external bus, the MMU is physically bypassed. The 20-bit source and destination registers in the DMAC are directly output on the physical address bus (A0-A19).

Physical address translation. Fig. 24 shows the way in which physical addresses are generated based on the contents of CBAR, CBR and BBR. MMU comparators classify an access by logical area as defined by CBAR. Depending on which of the three potential logical areas (Common Area 1, Bank Area, or Common Area 0) is being accessed, the appropriate 8-bit base address is added to the high-order 4 bits of the logical address, yielding a 20-bit physical address. CBR is associated with Common Area 1 accesses. Common Area 0, if defined, is always based at physical address 00000H.

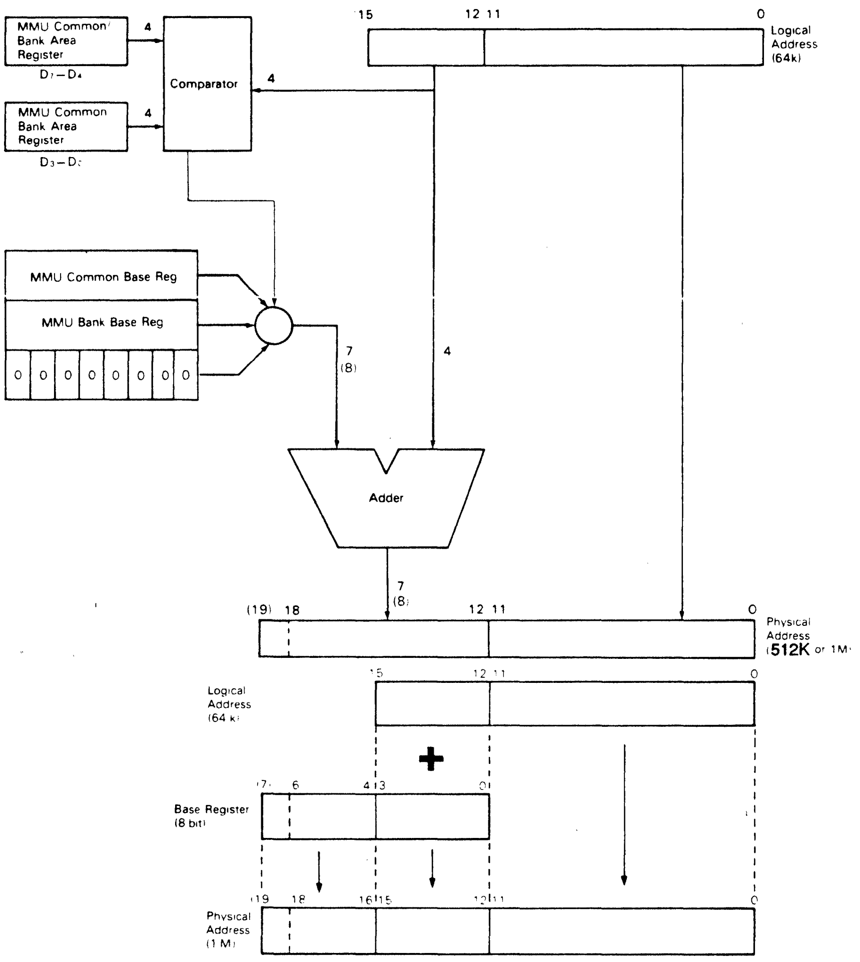


Figure 24. Physical Address Generation

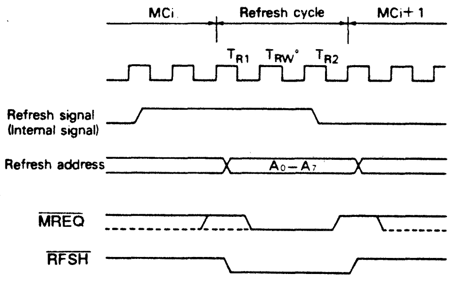
Dynamic RAM Refresh Control

The Z80180 incorporates a dynamic RAM refresh control circuit including 8-bit refresh address generation and programmable refresh timing. This circuit generates asynchronous refresh cycles inserted at the programmable interval independent of CPU program execution. For systems which do not use dynamic RAM, the refresh function can be disabled.

When the internal refresh controller determines that a refresh cycle should occur, the current instruction is interrupted at the first breakpoint between machine cycles. The refresh cycle is inserted by placing the refresh address on A_0-A_7 and the $\overline{RFSH}$ output is driven LOW.

Refresh cycles may be programmed to be either 2 or 3 clock cycles in duration by programming the REFW (Refresh Wait) bit in the Refresh Control Register (RCR). Note that the external $\overline{WAIT}$ input and the internal wait state generator are not effective during refresh.

Fig. 25 shows the timing of a refresh cycle with a refresh wait (T_{RW}) cycle.



NOTE: * If three refresh cycles are specified, T_{RW}, is inserted. Otherwise, T_{RW} is not inserted.
MC: Machine Cycle

Figure 25. Refresh Cycle Timing

DMA Controller (DMAC)

The Z80180 contains a two-channel DMA (Direct Memory Access) controller which supports high speed data transfer. Both channels (channel 0 and channel 1) have the following capabilities.

Memory Address Space. Memory source and destination addresses can be directly specified anywhere within the 1024K byte physical address space using 20-bit source and destination memory addresses. In addition, memory transfers can arbitrarily cross 64K byte physical address boundaries without CPU intervention.

I/O Address Space. I/O source and destination addresses can be directly specified anywhere within the 64K byte I/O address space (16-bit source and destination I/O addresses).

Transfer Length. Up to 64K bytes are transferred based on a 16-bit byte count register.

$\overline{DREQ}$ Input. Level and edge sense DREQ input detection are selectable.

$\overline{TEND}$ Output. Used to indicate DMA completion to external devices.

Transfer Rate. Each byte transfer can occur every 6 clock cycles. Wait states can be inserted in DMA cycles for slow memory or I/O devices. At the system clock (ø) = 6 MHz, the DMA transfer rate is as high as 1.0 megabytes/second (no wait states).

There is an additional feature disc for DMA interrupt request by DMA END. Each channel has the following additional specific capabilities.

Channel 0
Memory ↔ memory, memory ↔ I/O, memory ↔ memory mapped I/O transfers.

-Memory address increment, decrement, no-change.
-Burst or cycle steal memory to/from memory transfers.
-DMA to/from both ASCI channels.
-Higher priority than DMAC channel 1.

Channel 1
Memory ↔ I/O transfer.
Memory address increment, decrement

DMAC Registers
Each channel of the DMAC (channel 0, 1) has three registers specifically associated with that channel.

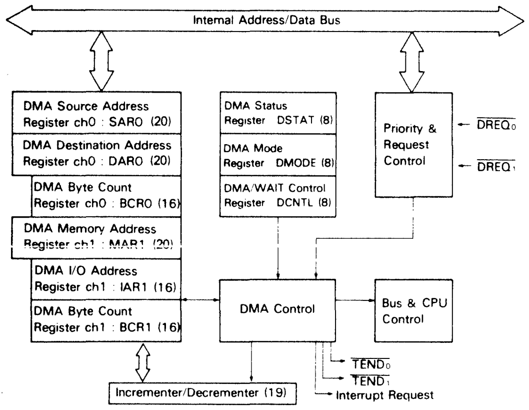
Channel 0
SAR0 - Source Address Register
DAR0 - Destination Address Register
BCR0 - Byte Count Register

Channel 1
MAR1 - Memory Address Register
IAR1 - I/O Address Register
BCR1 - Byte Count Register

The two channels share the following three additional registers in common.

DSTAT - DMA Status Register
DMODE - DMA Mode Register
DCNTL - DMA Control Register

DMAC Block Diagram. Fig.26 shows the Z64180 DMAC Block Diagram.



Internal Address/Data Bus
DMA Source Address Register ch0 : SAR0 (20)
DMA Destination Address Register ch0 : DAR0 (20)
DMA Byte Count Register ch0 : BCR0 (16)
DMA Memory Address Register ch1 : MAR1 (20)
DMA I/O Address Register ch1 : IAR1 (16)
DMA Byte Count Register ch1 : BCR1 (16)
DMA Status Register DSTAT (8)
DMA Mode Register DMODE (8)
DMA/WAIT Control Register DCNTL (8)
Priority & Request Control
DREQ0
DREQ1
DMA Control
Bus & CPU Control
TEND0
TEND1
Interrupt Request
Incrementer/Decrementer (19)

Figure 26. DMAC Block Diagram

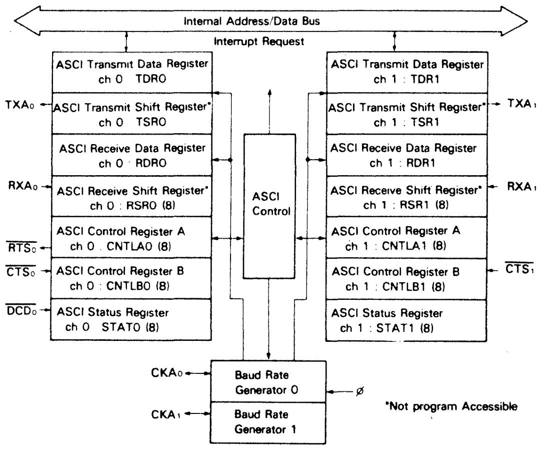
Asynchronous Serial Communication Interface (ASCI)

The Z80180 on-chip ASCI has two independent full-duplex channels. Based on full programmability of the following functions, the ASCI directly communicates with a wide variety of standard UARTs (Universal Asynchronous Receiver/Transmitter) including the Z8440 SIO and the Z8530 SCC.

The key functions for ASCI are shown below. Each channel is independently programmable.

-Full-duplex communication.
-7- or 8-bit data length.
-Program controlled 9th data bit for multiprocessor communication.
-1 or 2 stop bits.
-Odd, even, no parity.
-Parity, overrun, framing error detection.
-Programmable baud rate generator, /16 and /64 modes.
-Speed to 38.4K bits per second (CPU fc = 6.144 MHz).
-Modem control signals - Channel 0 has $\overline{\text{DCD0}}$, $\overline{\text{CTS0}}$ and $\overline{\text{RTS0}}$ Channel 1 has $\overline{\text{CTS1}}$.
-Programmable interrupt condition enable and disable.
-Operation with on-chip DMAC.

ASCI Block Diagram. Fig. 27 shows the ASCI Block Diagram.



Internal Address/Data Bus
Interrupt Request
ASCI Transmit Data Register ch 0 TDR0
ASCI Transmit Shift Register* ch 0 TSR0
ASCI Receive Data Register ch 0 RDR0
ASCI Receive Shift Register* ch 0 : RSR0 (8)
ASCI Control Register A ch 0 CNTLA0 (8)
ASCI Control Register B ch 0 : CNTLB0 (8)
ASCI Status Register ch 0 STAT0 (8)
ASCI Transmit Data Register ch 1 : TDR1
ASCI Transmit Shift Register* ch 1 : TSR1
ASCI Receive Data Register ch 1 : RDR1
ASCI Receive Shift Register* ch 1 : RSR1 (8)
ASCI Control Register A ch 1 : CNTLA1 (8)
ASCI Control Register B ch 1 : CNTLB1 (8)
ASCI Status Register ch 1 : STAT1 (8)
ASCI Control
TXA0
RXA0
RTS0
CTS0
DCD0
TXA1
RXA1
CTS1
CKA0
CKA1
Baud Rate Generator 0
Baud Rate Generator 1
ø
*Not program Accessible

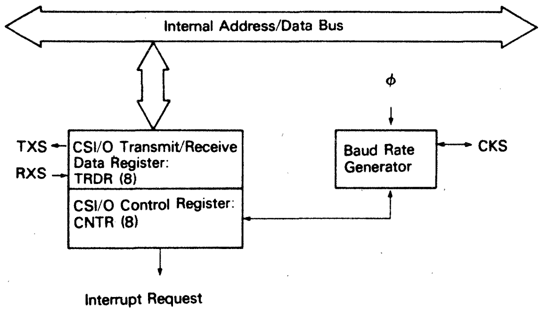
Figure 27. ASCI Block Diagram

Clocked Serial I/O Port (CSI/O)

The Z80180 includes a simple, high speed clock, synchronous serial I/O port. The CSI/O includes transmit/receive (half-duplex), fixed 8-bit data, and internal or external data clock selection. High speed operation (baud rate 200K bits/second at fC = 4 MHz) is provided. The CSI/O is ideal for implementing a multiprocessor communication link between mulitple Z80180s. These secondary devices may typically perform a portion of the system I/O processing, i.e. keyboard scan/decode, LDC interface, etc.

CSI/O Block Diagram. The CSI/O block diagram is shown in Fig. 28. The CSI/O consists of two registers - the Transmit/Receive Data Register (TRDR) and Control Register (CNTR).

CSI/O Transmit/Receive Data Register (TRDR: I/O Address = 0BH). TRDR is used for both CSI/O transmission and reception. Thus, the system design must insure that the constraints of half-duplex operation are met (Transmit and receive operation cannot occur simultaneously). For example, if a CSI/O transmission is attempted while the CSI/O is receiving data, a CSI/O will not work. Also note that TRDR is not buffered. Therefore, attempting to perform a CSI/O transmit while the previous transmit data is still being shifted out causes the shift data to be immediately updated, thereby corrupting the transmit operation in progress. Similarly, reading TRDR while a transmit or receive is in progress should be avoided.



Internal Address/Data Bus
ϕ
TXS
RXS
CSI/O Transmit/Receive Data Register: TRDR (8)
Baud Rate Generator
CKS
CSI/O Control Register: CNTR (8)
Interrupt Request

Figure 28. CSI/O Block Diagram

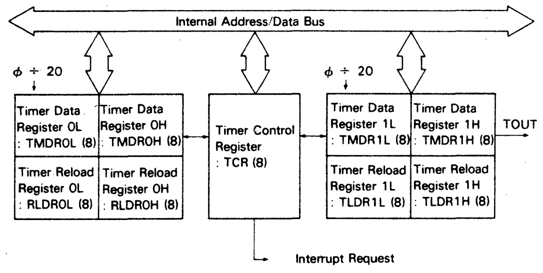
CSI/O Register Description

CSI/O Control/Status Register (CNTR: I/O Address = 0AH). CNTR is used to monitor CSI/O status, enable and disable the CSI/O, enable and disable interrupt generation, and select the data clock speed and source.

Programmable Reload Timer (PRT)

The Z80180 contains a two channel 16-bit Programmable Reload Timer. Each PRT channel contains a 16-bit down counter and a 16-bit reload register. The down counter is directly read and written and a down counter overflow interrupt can be programmably enabled or disabled. Also, PRT channel 1 has a TOUT output pin (pin 31 - multiplexed with A18) which can be set HIGH, LOW, or toggled. Thus, PRT1 can perform programmable output waveform generation.

PRT block diagram. The PRT block diagram is shown in Fig. 29. The two channels have seperate timer data and reload registers and a common status/control register. The PRT input clock for both channels is equal to the system clock divided by 20.



Internal Address/Data Bus
ϕ ÷ 20
ϕ ÷ 20
Timer Data Register 0L : TMDR0L (8)
Timer Data Register 0H : TMDR0H (8)
Timer Control Register : TCR (8)
Timer Data Register 1L : TMDR1L (8)
Timer Data Register 1H : TMDR1H (8)
TOUT
Timer Reload Register 0L : RLDR0L (8)
Timer Reload Register 0H : RLDR0H (8)
Timer Reload Register 1L : TLDR1L (8)
Timer Reload Register 1H : TLDR1H (8)
Interrupt Request

Figure 29. PRT Block Diagram

Secondary Bus Interface

E clock Output Timing. The Z80180 also has a secondary bus interface that allows it to easily interface with other peripheral families.

These devices require connection with the Z80180 synchronous E clock output. The speed (access time) required for the peripheral devices are determined by the Z80180 clock rate. Table 19, and Figures 80-82 define E clock output timing.

On-Chip Clock Generator

The Z80180 contains a crystal oscillator and system clock generator. A crystal can be directly connected or an external clock input can be provided. In either case, the system clock is equal to one-half the input clock. For example, a crystal or external clock input of 8 MHz corresponds with a system clock rate of 4 MHz.

The following table shows the AT cut crystal characteristics (Co, Rs) and the load capacitance (CL1, CL2) required for various frequencies of Z80180 operation.

Clock Frequency / Item	4MHz	4MHz < f ≦ 12MHz	12MHz < f ≦ 16MHz
Co	< 7 pF	< 7 pF	< 7 pF
Rs	< 60Ω	< 60Ω	< 60Ω
CL_1, CL_2	10 to 22 pF ±10%	10 to 22 pF ±10%	10 to 22 pF ±10%

Table 4.

If an external clock input is used instead of a crystal, the waveform (twice the clock rate) should exhibit a 50% ± 10% duty cycle. Note that the minimum clock input HIGH voltage level is Vcc-0.6V. The external clock input is connected to the EXTAL pin, while the XTAL pin is left open. Fig. 30 shows external clock interface.

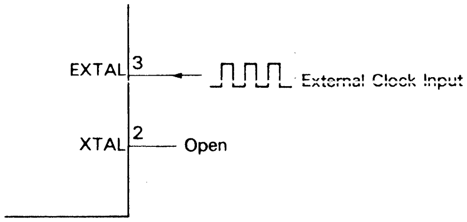


Figure 30. External Clock Interface

Miscellaneous

Free Running Counter (I/O Address = 18H)

Read only 8-bit free running counter without control registers and status registers. The contents of the 8-bit free running counter is counted down by one with an interval of 10 clock cycles. The free running counter continues counting down without being affected by the read operation.

If data is written into the free running counter, the interval of DRAM refresh cycle and baud rates for the ASCI and CSI/O are not guaranteed.

In IOSTOP mode, the free running counter continues counting down. It is initialized to FFH during RESET.

SOFTWARE ARCHITECTURE

Instruction Set. The Z80180 is object code compatible with the Z80 CPU, refer to the Z80 CPU Technical Manual or the Z80 Assembly Language Programming Manual for further details.

New Instructions	Operation
SLP	Enter SLEEP mode
MLT	8-bit multiply with 16-bit result
INO g, (m)	Input contents of immediate I/O address
OUTO (m), g	Output register contents to immediate I/O address
OTIM	Block output - increment
OTIMR	Block output - increment and repeat
OTDM	Block output - decrement
OTDMR	Block output - decrement and repeat
TSTIO m	Non-destructive AND, I/O port, and accumulator
TST g	Non-destructive AND, register, and accumulator
TST m	Non-destructive AND, immediate data and accumulator.
TST (HL)	Non-destructive AND, memory data, and accumulator.

SLP - Sleep. The SLP instruction causes the Z80180 to enter the SLEEP low power consumption mode. See section 2.4 for a complete description of the SLEEP state.

MLT - Multiply. The MLT performs unsigned multiplication on two 8-bit numbers yielding a 16-bit result. MLT may specify BC, DE, HL or SP registers. In all cases, the 8-bit operands are loaded into each half of the 16-bit register and the 16-bit result is returned in that register.

OTIM, OTIMR, OTDM, OTDMR - Block I/O. The contents of memory pointed to by HL is output to the I/O address in (C). The memory address (HL) and I/O address (C) are incremented in OTIM and OTIMR and decremented in OTDM and OTDMR, respectively. The B register is decremented. The OTIMR and OTDMR variants repeat the above sequence until register B is decremented to 0. Since the I/O address (C) is automatically incremented or decremented, these instructions are useful for block I/O (such as Z80180 on-chip I/O) initialization. When I/O is accessed, 00H is output in high-order bits of address automatically.

TSTIO m - Test I/O Port. The contents of the I/O port addressed by C are ANDed with immediately specified 8-bit data and the status flags are updated. The I/O port contents are not written (non-destructive AND). When I/O is accessed, 00H is output in higher bits of address automatically.

TST g - Test Register. The contents of the specified register are ANDed with the accumulator (A) and the status flags are updated. The accumulator and specified register are not changed (non-destructive AND).

TST m - Test Immediate. The contents of the immediately specified 8-bit data are ANDed with the accumulator (A) and the status flags are updated. The accumulator is not changed (non-destructive AND).

TST (HL) - Test Memory. The contents of memory pointed to by HL are ANDed with the accumulator (A) and the status flags are updated. The memory contents and accumulator are not changed (non-destructive AND).

INO g, (m) - Input, Immediate I/O address. The contents of immediately specified 8-bit I/O address are input into the specified register. When I/O is accessed, 00H is output in high-order bits of the address automatically.

OUTO (m), g - Output, immediate I/O address. The contents of the specified register are output to the immediately specified 8-bit I/O address. When I/O is accessed, 00H is output in high-order bits of the address automatically.

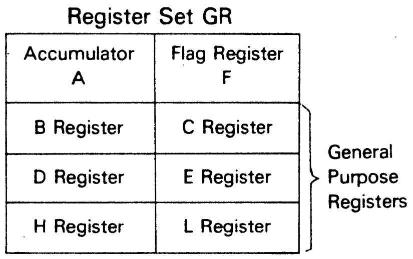
CPU Registers

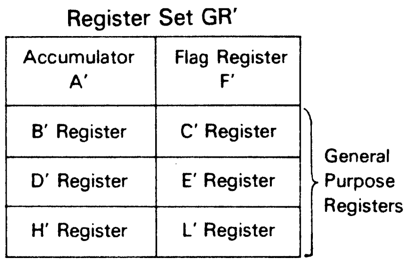
The Z80180 CPU registers consist of Register Set GR, Register Set GR' and Special Registers.

The Register Set GR consists of 8-bit Accumulator (A), 8-bit Flag Register (F), and three General Purpose Registers (BC, DE, and HL) which may be treated as 16-bit registers (BC, DE, and HL) or as individual 8-bit registers (B, C, D, E, H, and L) depending on the instruction to be executed. The Register Set GR' is alternate register set of Register Set GR and also contains Accumulator (A'), Flag Register (F') and three General Purpose Registers (BC', DE', and HL'). While the alternate Register Set GR' contents are not directly accessible, the contents can be programmably exchanged at high speed with those of Register Set GR.

The Special Registers consist of 8-bit Interrupt Vector Register (I), 8-bit R Counter (R), two 16-bit Index Registers (IX and IY), 16-bit Stack Pointer (SP), and 16-bit Program Counter (PC).

Fig. 31 shows CPU registers configuration.





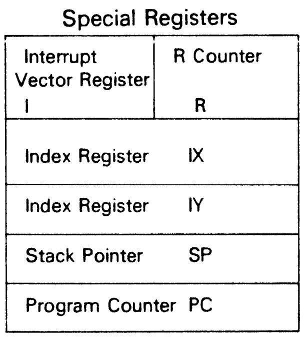


Figure 31. CPU Registers

Z80180 ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Value	Unit
Supply Voltage	V_{CC}	$-0.3 \sim +7.0$	V
Input Voltage	V_{in}	$-0.3 \sim V_{CC}+0.3$	V
Operating Temperature	T_{opr}	$0 \sim 70$	°C
Storage Temperature	T_{stg}	$-55 \sim +150$	°C

[NOTE] Permanent LSI damage may occur if maximum ratings are exceeded. Normal operation should be under recommended operating conditions. If these conditions are exceeded, it could affect reliability of LSI.

DC CHARACTERISTICS

(Vcc = 5V + 10%, Vss = 0V, over specified temperature range, unless otherwise noted.)

Symbol	Item	Condition	min	typ	max	Unit
V_{IH1}	Input "H" Voltage RESET, EXTAL, NMI		$V_{CC}-0.6$	–	$V_{CC}+0.3$	V
V_{IH2}	Input "H" Voltage Except RESET, EXTAL, NMI		2.0	–	$V_{CC}+0.3$	V
V_{IL1}	Input "L" Voltage RESET, EXTAL, NMI		−0.3	–	0.6	V
V_{IL2}	Input "L" Voltage Except RESET, EXTAL, NMI		−0.3	–	0.8	V
V_{OH}	Output "H" Voltage All outputs	$I_{OH} = -200\mu A$	2.4	–	–	V
		$I_{OH} = -20\mu A$	$V_{CC}-1.2$	–	–	
V_{OL}	Output "L" Voltage All Outputs	$I_{OL} = 2.2$ mA	–	–	0.45	V
I_{IL}	Input Leakage Current All Inputs Except XTAL, EXTAL	$Vin=0.5 \sim V_{CC}-0.5$	–	–	1.0	μA
I_{TL}	Three State Leakage Current	$Vin=0.5 \sim V_{CC}-0.5$	–	–	1.0	μA
I_{CC}*	Power Dissipation* (Normal Operation)	f= 6 MHz	–	15	40	mA
		f= 8 MHz	–	20	50	
		f=10MHz	–	25	60	
	Power Dissipation* (SYSTEM STOP mode)	f= 6 MHz	–	3.8	12.5	
		f= 8 MHz	–	5	15.0	
		f=10MHz	–	6.3	17.5	
Cp	Pin Capacitance	Vin= 0V, f= 1 MHz Ta= 25°C	–	–	12	pF

* $V_{IHmin} = V_{CC} - 1.0V$, $V_{ILmax} = 0.8V$ (all output terminals are at no load.)
Vcc=5.00V

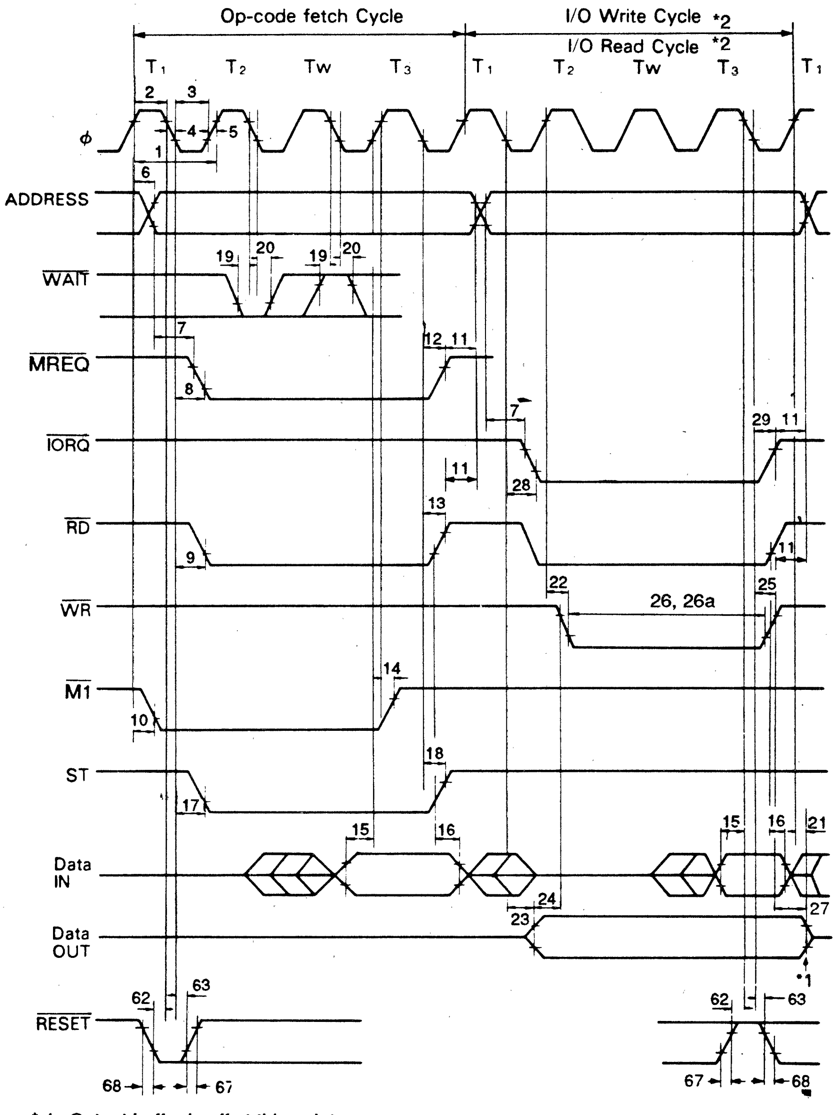
Z80180 AC CHARACTERISTICS

(Vcc = 5V + 10%, Vss = 0V, over specified temperature range, unless otherwise noted.)

No.	Symbol	Item	Z80180-6		Z80180-8		Z80180-10		Unit
			min	max	min	max	min	max	
1.	t_{cyc}	Clock Cycle Time	162	2000	125	2000	100	2000	ns
2.	t_{CHW}	Clock "H" Pulse Width	65	—	50	—	40	—	ns
3.	t_{CLW}	Clock "L" Pulse Width	65	—	50	—	40	—	ns
4.	t_{cf}	Clock Fall Time	—	15	—	15	—	10	ns
5.	t_{cr}	Clock Rise Time	—	15	—	15	—	10	ns
6.	t_{AD}	Ø ↑ to Address Valid Delay	—	90	—	80	—	70	ns
7.	t_{AS}	Address Valid to ($\overline{MREQ}$ ↓ or $\overline{IORQ}$ ↓)	30	—	20	—	10	—	ns
8.	t_{MED1}	Ø ↓ to $\overline{MREQ}$ ↓ Delay	—	60	—	50	—	50	ns
9.	t_{RDD1}	Ø ↓ to $\overline{RD}$ ↓ Delay $\overline{IOC}$ = 1	—	60	—	50	—	50	ns
		Ø ↑ to $\overline{RD}$ ↓ Delay $\overline{IOC}$ = 0	—	65	—	60	—	55	
10.	t_{M1D1}	Ø ↑ to $\overline{M1}$ ↓ Delay	—	80	—	70	—	60	ns
11.	t_{AH}	Address Hold Time from ($\overline{MREQ}$, $\overline{IOREQ}$, $\overline{RD}$, $\overline{WR}$)	35	—	20	—	10	—	ns
12.	t_{MED2}	Ø ↓ to $\overline{MREQ}$ ↑ Delay	—	60	—	50	—	50	ns
13.	t_{RDD2}	Ø ↓ to $\overline{RD}$ ↑ Delay	—	60	—	50	—	50	ns
14.	t_{M1D2}	Ø ↑ to $\overline{M1}$ ↑ Delay	—	80	—	70*	—	60	ns
15.	t_{DRS}	Data Read Set-up Time	40	—	30	—	25	—	ns
16.	t_{DRH}	Data Read Hold Time	0	—	0	—	0	—	ns
17.	t_{STD1}	Ø ↓ to ST ↓ Delay	—	90	—	70	—	60	ns
18.	t_{STD2}	Ø ↓ to ST ↑ Delay	—	90	—	70	—	60	ns
19.	t_{WS}	$\overline{WAIT}$ Set-up Time to Ø ↓	40	—	40	—	30	—	ns
20.	t_{WH}	$\overline{WAIT}$ Hold Time from Ø ↓	40	—	40	—	30	—	ns
21.	t_{WDZ}	Ø ↑ to Data Float Display	—	95	—	70	—	60	ns
22.	t_{WRD1}	Ø ↑ to $\overline{WR}$ ↓ Delay	—	65	—	60	—	50	ns
23.	t_{WDD}	Ø ↓ to Write Data Delay Time	—	90	—	80	—	60	ns
24.	t_{WDS}	Write Data Set-up Time to $\overline{WR}$ ↓	40	—	20	—	15	—	ns
25.	t_{WRD2}	Ø ↓ to $\overline{WR}$ ↑ Delay	—	80	—	60	—	50	ns
26.	t_{WRP}	$\overline{WR}$ Pulse Width (Memory Write Cycle)	170	—	130	—	110	—	ns
26a.		$\overline{WR}$ Pulse Width (I/O Write Cycle)	332	—	255	—	210	—	ns

	Symbol	Item	Z80180-6		Z80180-8		Z80180-10		Unit
			min	max	min	max	min	max	
27.	t_{WDH}	Write Data Hold Time from ($\overline{WR}$ ↑)	40	—	15	—	10	—	ns
28.	t_{IOD1}	Ø ↓ to $\overline{IORQ}$ ↓ Delay $\overline{IOC}$= 1	—	60	—	50	—	50	ns
		Ø ↑ to $\overline{IORQ}$ ↓ Delay $\overline{IOC}$= 0	—	65	—	60	—	55	
29.	t_{IOD2}	Ø ↓ to $\overline{IORQ}$ ↑ Delay	—	60	—	50	—	50	ns
30.	t_{IOD3}	$\overline{M1}$ ↓ to $\overline{IORQ}$ ↓ Delay	340	—	250	—	200	—	ns
31.	t_{INTS}	$\overline{INT}$ Set-up Time to Ø ↓	40	—	40	—	30	—	ns
32.	t_{INTH}	$\overline{INT}$ Hold Time from Ø ↓	40	—	40	—	30	—	ns
33.	t_{NMIW}	$\overline{NMI}$ Pulse Width	120	—	100	—	80	—	ns
34.	t_{BRS}	$\overline{BUSREQ}$ Set-up Time to Ø ↓	40	—	40	—	30	—	ns
35.	t_{BRH}	$\overline{BUSREQ}$ Hold Time from Ø ↓	40	—	40	—	30	—	ns
36.	t_{BAD1}	Ø ↑ to $\overline{BUSACK}$ ↓ Delay	—	95	—	70	—	60	ns
37.	t_{BAD2}	Ø ↓ to $\overline{BUSACK}$ ↑ Delay	—	95	—	70	—	60	ns
38.	t_{BZD}	Ø ↑ to Bus Floating Delay Time	—	125	—	90	—	80	ns
39.	t_{MEWH}	$\overline{MREQ}$ Pulse Width (HIGH)	110	—	90	—	70	—	ns
40.	t_{MEWL}	$\overline{MREQ}$ Pulse Width (LOW)	125	—	100	—	80	—	ns
41.	t_{RFD1}	Ø ↑ to $\overline{RFSH}$ ↓ Delay	—	90	—	80	—	60	ns
42.	t_{RFD2}	Ø ↑ to $\overline{RFSH}$ ↑ Delay	—	90	—	80	—	60	ns
43.	t_{HAD1}	Ø ↑ to $\overline{HALT}$ ↓ Delay	—	90	—	80	—	50	ns
44.	t_{HAD2}	Ø ↑ $\overline{HALT}$ ↑ Delay	—	90	—	80	—	50	ns
45.	t_{DRQS}	$\overline{DREQi}$ Set-up Time to Ø ↑	40	—	40	—	30	—	ns
46.	t_{DRQH}	$\overline{DREQi}$ Hold Time from Ø ↑	40	—	40	—	30	—	ns
47.	t_{TED1}	Ø ↓ to $\overline{TENDi}$ ↓ Delay	—	70	—	60	—	50	ns
48.	t_{TED2}	Ø ↓ to $\overline{TENDi}$ ↑ Delay	—	70	—	60	—	50	ns
49.	t_{ED1}	Ø ↑ to E ↑ Delay	—	95	—	70	—	60	ns
50.	t_{ED2}	Ø↓ or ↑to E ↓Delay	—	95	—	70	—	60	ns
51.	P_{WEH}	E Pulse Width (HIGH)	75	—	65	—	55	—	ns
52.	P_{WEL}	E Pulse Width (LOW)	180	—	130	—	110	—	ns

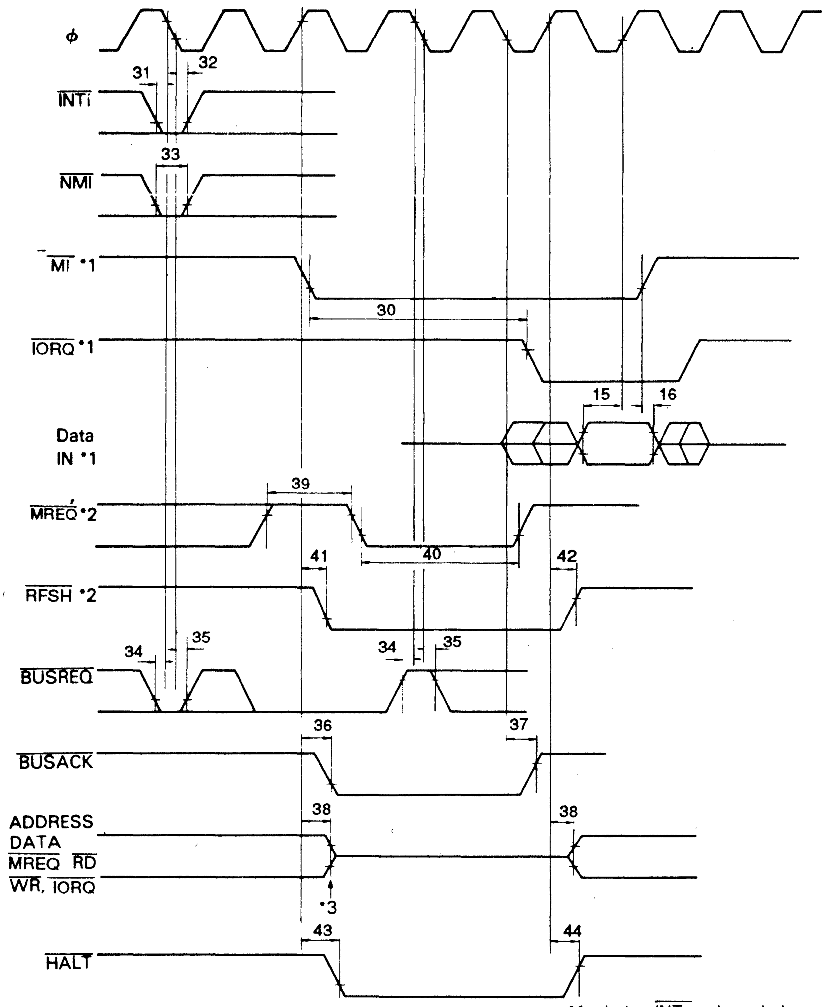
	Symbol	Item	Z80180-6		Z80180-8		Z80180-10		Unit
			min	max	min	max	min	max	
53.	t_{Er}	Enable Rise Time	—	20	—	20	—	20	ns
54.	t_{Ef}	Enable Fall Time	—	20	—	20	—	20	ns
55.	t_{TOD}	Ø ↓ to Timer Output Delay	—	300	—	200	—	150	ns
56.	t_{STDI}	CSI/O Transmit Data Delay Time (Internal Clock Operation)	—	200	—	200	—	150	ns
57.	t_{STDE}	CSI/O Transmit Data Delay Time (External Clock Operation)	—	7.5tcyc +300	—	7.5tcyc +200	—	7.5tcyc +150	ns
58.	t_{SRSI}	CSI/O Receive Data Set-up Time (Internal Clock Operation)	1	—	1	—	1	—	tcyc
59.	t_{SRHI}	CSI/O Receive Data Hold Time (Internal Clock Operation)	1	—	1	—	1	—	tcyc
60.	t_{SRSE}	CSI/O Receive Data Set-up Time (External Clock Operation)	1	—	1	—	1	—	tcyc
61.	t_{SRHE}	CSI/O Receive Data Hold Time (External Clock Operation)	1	—	1	—	1	—	tcyc
62.	t_{RES}	$\overline{\text{RESET}}$ Set-up Time to Ø ↓	120	—	100	—	80	—	ns
63.	t_{REH}	$\overline{\text{RESET}}$ Hold Time from Ø ↓	80	—	70	—	50	—	ns
64.	t_{OSC}	Oscillator Stabilization Time	—	20	—	20	—	TBD	ms
65.	t_{EXr}	External Clock Rise Time (EXTAL)	—	25	—	25	—	25	ns
66.	t_{EXf}	External Clock Fall Time (EXTAL)	—	25	—	25	—	25	ns
67.	t_{Rr}	$\overline{\text{RESET}}$ Rise Time	—	50	—	50	—	50	ms
68.	t_{Rf}	$\overline{\text{RESET}}$ Fall Time	—	50	—	50	—	50	ms
69.	t_{Ir}	Input Rise Time (except EXTAL, $\overline{\text{RESET}}$)	—	100	—	100	—	100	ns
70.	t_{If}	Input Fall Time (except EXTAL, $\overline{\text{RESET}}$)	—	100	—	100	—	100	ns



* 1 Output buffer is off at this point.

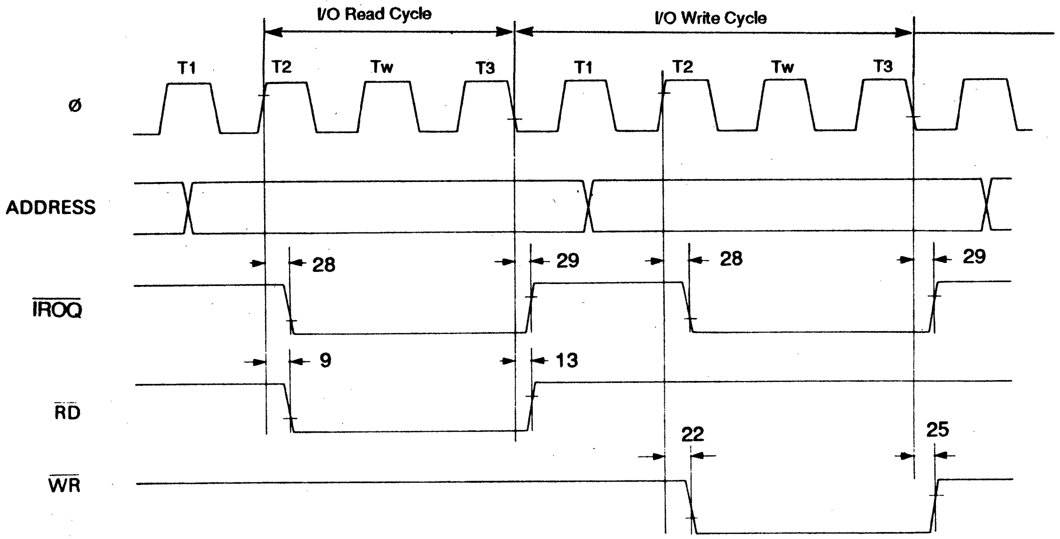
* 2 Memory Read/Write Cycle timing are the same as I/O Read/Write Cycle except there are no automatic wait states (Tw), and $\overline{\text{MREQ}}$ is active instead of $\overline{\text{IORQ}}$

CPU Timing [Op-code fetch Cycle / Memory Read Cycle / Memory Write Cycle / I/O Write Cycle / I/O Read Cycle]



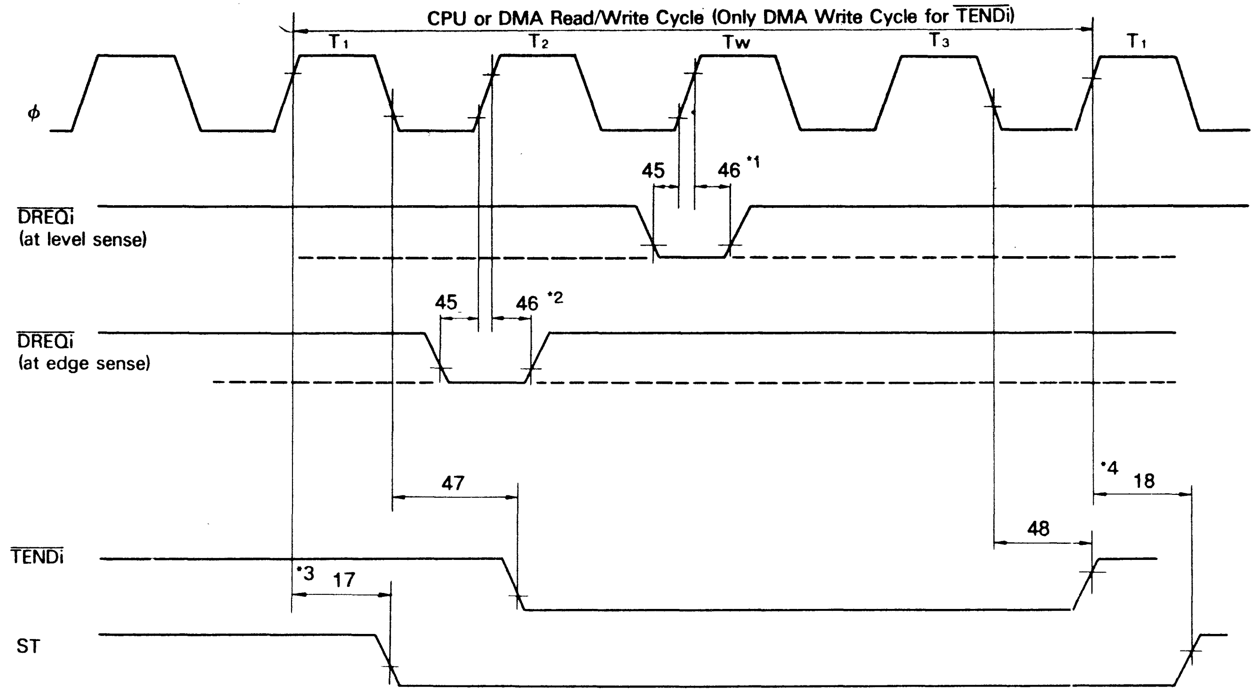
*1 during $\overline{\text{INT}_0}$ acknowledge cycle
*2 during refresh cycle
*3 Output buffer is off at this point.

CPU Timing (**$\overline{\text{INT}_0}$ Acknowledge cycle**
Refresh Cycle
BUS RELEASE Mode
HALT Mode
SLEEP Mode
SYSTEM STOP Mode)



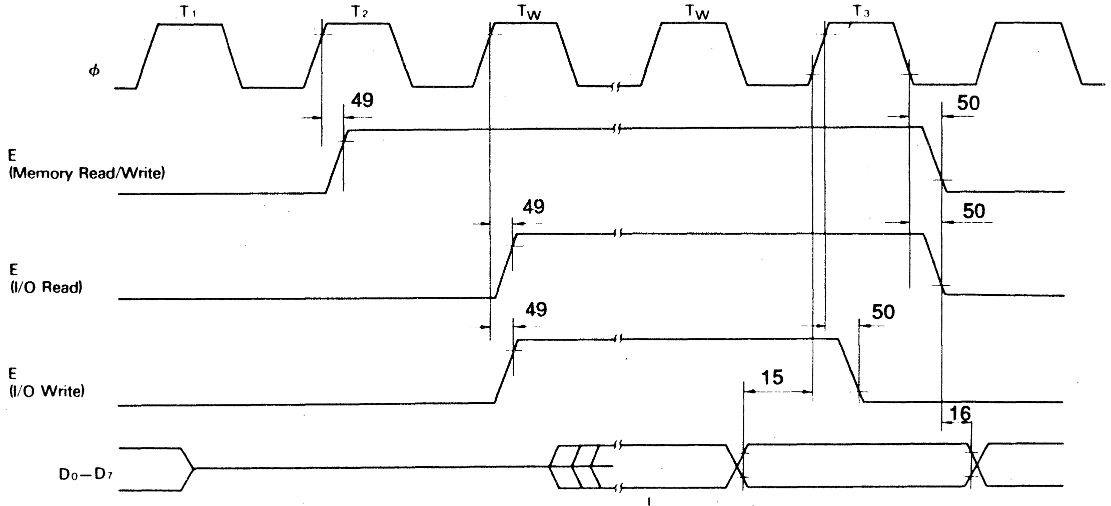
CPU Timing ($\overline{\text{IOC}}$=0)

I/O Read Cycle
I/O Write Cycle

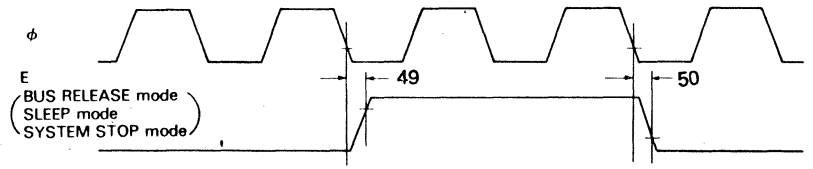

CPU or DMA Read/Write Cycle (Only DMA Write Cycle for $\overline{\text{TENDi}}$)
T1
T2
Tw
T3
T1
φ
45
46 *1
$\overline{\text{DREQi}}$ (at level sense)
45
46 *2
$\overline{\text{DREQi}}$ (at edge sense)
47
*4 18
48
$\overline{\text{TENDi}}$
*3 17
ST

DMA Control Signals

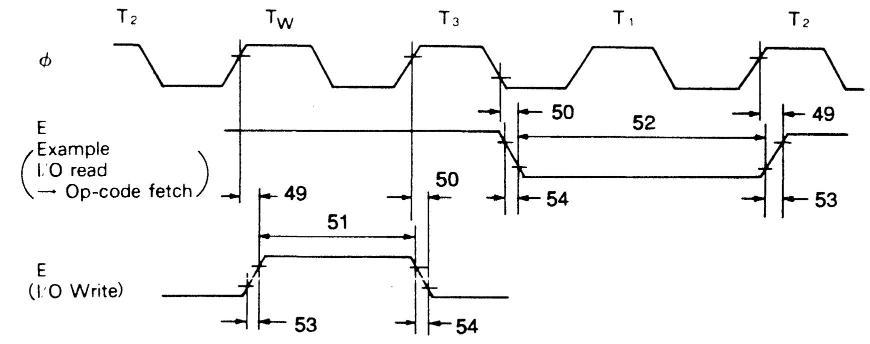
*1 t_{DRQS} and t_{DRQH} are specified for the rising edge of clock followed by T_3.
*2 t_{DRQS} and t_{DRQH} are specified for the rising edge of clock.
*3 DMA cycle starts.
*4 CPU cycle starts.



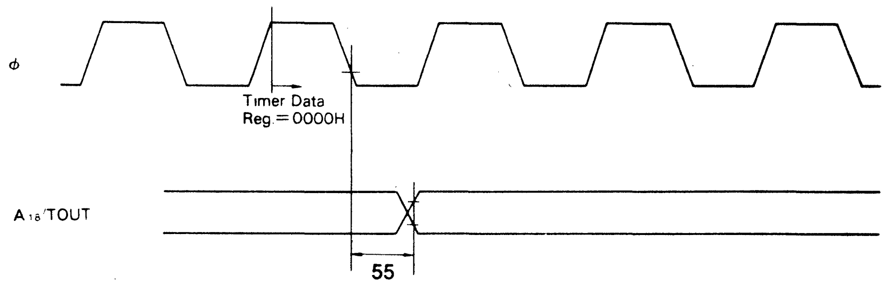
E Clock Timing (Memory Read/Write Cycle / I/O Read/Write Cycle)



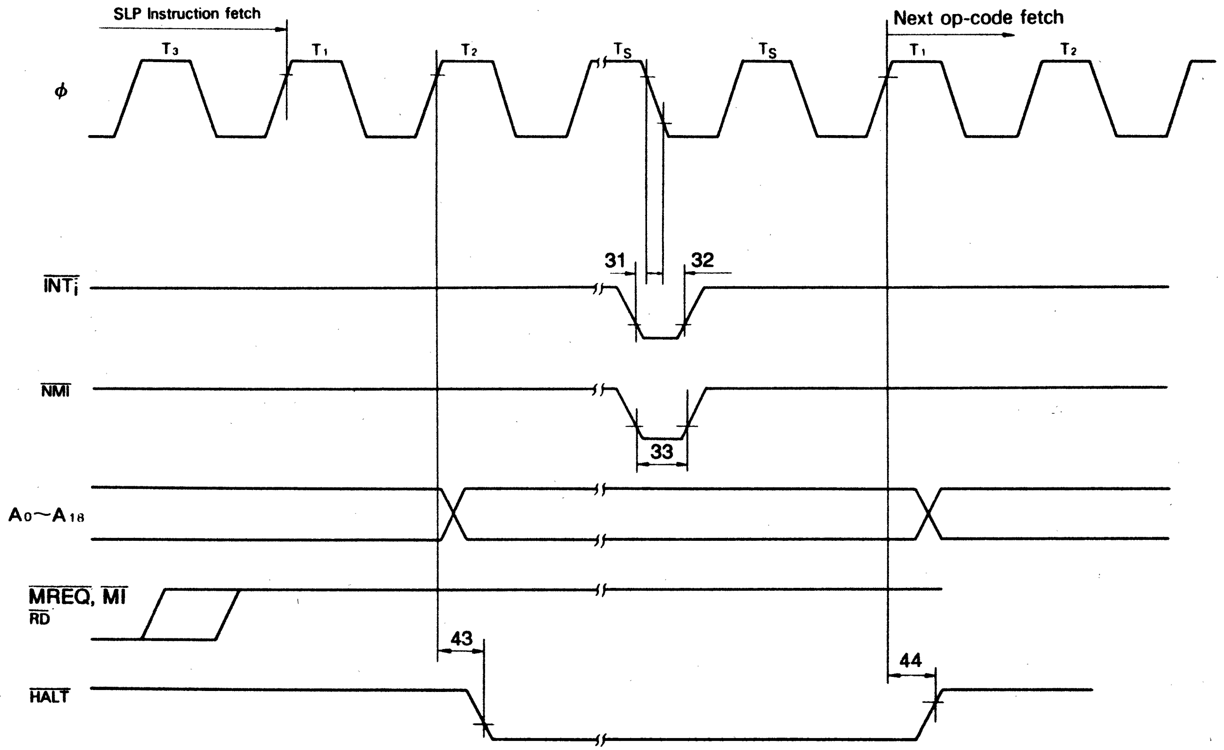
E Clock Timing (BUS RELEASE Mode / SLEEP Mode / SYSTEM STOP Mode)



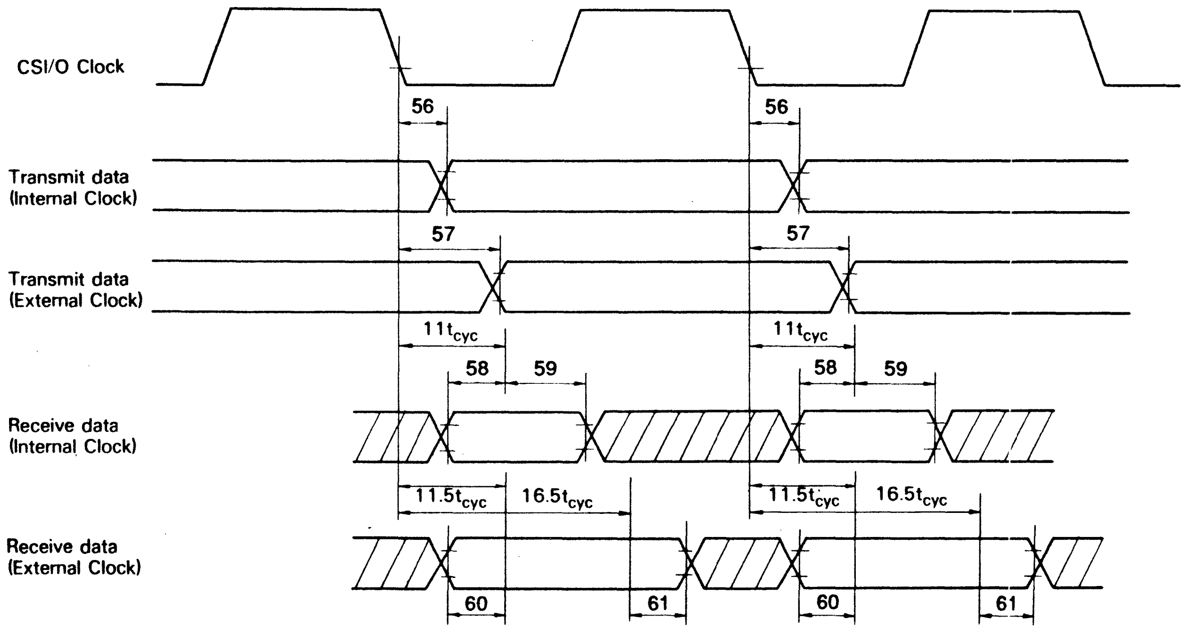
E Clock Timing (**Minimum timing example of P_{WEL} and P_{WEH}**)



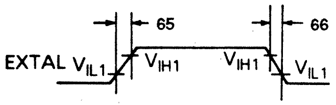
Timer Output Timing



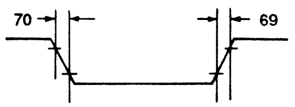
SLP Execution Cycle


CSI/O Clock
Transmit data (Internal Clock)
Transmit data (External Clock)
Receive data (Internal Clock)
Receive data (External Clock)
56
57
$11t_{cyc}$
58
59
$11.5t_{cyc}$
$16.5t_{cyc}$
60
61

CSI/O Receive/Transmit Timing



External Clock Rise Time and Fall Time



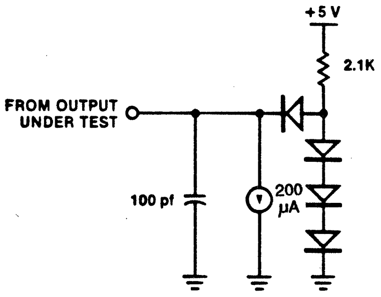
Input Rise Time and Fall Time (Except EXTAL, $\overline{\text{RESET}}$)

STANDARD TEST CONDITIONS:

The DC Characteristics and Capacitance sections above apply to the following standard test conditions, unless otherwise noted. All voltages are referenced to GND (0V). Positive current flows in to the referenced pin.

All AC parameters assume a load capaitance of 100 pF. Add 10 ns delay for each 50 pF increase in load up to a maximum of 200 pF for the data bus and 100 pF for the address and control lines. AC timing measurements are referenced to 1.5 volts (except for CLOCK, which is referenced to the 10% and 90% points).

The Ordering Information section lists temperature ranges and product numbers. Package drawings are in the Package Information section. Refer to the Literature List for additional documentation.



Zilog

PRELIMINARY
Product Specification

January 1989

Z280™ MPU Microprocessor Unit

FEATURES

- Designed in CMOS for low power operations.
- Enhanced Z80® CPU instruction set that maintains object-code compatibility with Z80 microprocessor.
- Three-stage pipelined, 16-bit CPU architecture with user and system modes.
- Direct coprocessor and multiprocessor interface support.
- On-chip paged Memory Management Unit (MMU) addresses up to 16 Mbytes.
- On-chip 256-byte instruction and data associative cache memory with burst load.
- High performance 16-bit Z-BUS® bus interface or 8-bit Z80 CPU compatible bus interface.
- Three on-chip 16-bit counter/timers.
- Four on-chip DMA channels.
- On-chip full duplex UART.
- Refresh controller for dynamic RAMs.
- **On-chip oscillator or direct input clock options.**
- **20 MHz oscillator clock frequency.**

GENERAL DESCRIPTION

Zilog's new Z280 microprocessor features a high-performance microprocessor designed to give the end-user a powerful and cost-effective solution to application requirements. The Z280 microprocessor unit (MPU) incorporates advanced architectural features that allow fast and efficient throughput and increased memory addressing while maintaining Z80 object-code compatibility. The Z280 microprocessor offers a continuing growth path for present Z80 based designs and serves as a high-performance microprocessor for new, advanced designs.

Central to the Z280 microprocessor is an enhanced version of the Z80 Central Processing Unit (CPU). To assure system integrity, the Z280 microprocessor can operate in either user or system mode, allowing protection of system resources from user tasks and programs. System mode operation is supported by the addition of the system Stack Pointer to the working register set. The IX and IY registers have been modified so that in addition to their regular function as index registers, each register can be accessed as a 16-bit general purpose register or as two byte registers. The R register, used for refresh by the Z80 CPU, is now available to the programmer as a data register in the Z280 microprocessor.

The Z80 CPU instruction set has been retained, meaning that the Z280 microprocessor is completely binary-code compatible with present Z80 code. The basic addressing modes of the Z80 microprocessor have been augmented with the addition of Indexed mode with full 16-bit displacement, Program Counter Relative with 16-bit displacement, Stack Pointer Relative with 16-bit displacement, and Base Index modes. The new addressing modes are incorporated into many of the old Z80 CPU instructions, resulting in greater flexibility and power. Some additions to the instruction set include 8-and 16-bit signed and unsigned multiply and divide, 8-and 16-bit sign extension, and a test and set instruction to support multiprocessing. The 16-bit instructions have been expanded to include 16-bit compare, memory increment, memory decrement, negate, add, and subtract, in addition to the previously mentioned multiply and divide.

A requirement of many of today's microprocessor-based system designs is to increase the memory address space beyond the 64K byte range of typical 8-bit microprocessors. The Z280 microprocessor has an on-chip Memory Management Unit (MMU) that enables addressing of up to 16M bytes of memory. In addition to enabling the address space to be expanded, the MMU performs other memory management functions previously handled by dedicated off-chip memory management devices.

I/O address space has been expanded by the addition of an I/O Page register used to select pages of I/O addresses. The 8-bit I/O Page register can select one of 256 possible pages of I/O addresses to be active at one time, allowing a total of 64K I/O addresses to be accessed.

There are 256 bytes of on-chip memory present on the Z280 MPU. This memory can be configured as a high-speed cache or as a fixed address local memory. When configured as a cache, the memory can be programmed to be instruction only, data only, or both data and instruction. The cache memory allows programs to run significantly faster by reducing the number of external bus accesses. Operation and update of the cache is performed automatically and is completely transparent to the user. When used as a local memory, the addresses are programmable, which permits selected storage of time-critical loops in local memory.

Many features that have traditionally been handled by external peripheral devices have been incorporated in the design of the Z280 microprocessor. The "on-chip peripherals" reduce system chip count and reduce interconnection on the external bus. The Z280 MPU contains an on-chip clock oscillator and a refresh controller that provides 10-bit refresh addresses for dynamic memories. Also present are additional on-chip peripherals to provide system design flexibility. To support high-bandwidth data transmission, four Direct Memory Access (DMA) channels are incorporated on-chip. Each DMA channel operates using full 24-bit source and destination addresses with a 16-bit count. The channels can be programmed to operate in single transaction, burst, or continuous mode. System event counting and timing requirements are met with the help of the three 16-bit counter/timers. The counter/timer functions can be externally controlled with gate and trigger inputs, and can be programmed as retriggerable or nonretriggerable. A full duplex UART, capable of handling a variety of data and character formats, is present to facilitate asynchronous serial communication.

The Z280 MPU also features programmable bus timing, allowing the user to tailor timing to the individual system. Upon reset the Z280 microprocessor can be programmed for system timing that is one-fourth, one-half, or equal to the speed of the MPU's internal Central Processing Unit (CPU), with one-half being the default. In addition to clock scaling, programmable wait states can be inserted during various bus transactions. Without the use of external hardware, one to three wait states can be inserted into memory, I/O, and interrupt acknowledge transactions. Furthermore, separate memory wait states can be specified for upper and lower memory areas, facilitating the use of different speeds of ROMs and RAMs in the same system.

An additional feature of the 16-bit bus interface is the ability to support "nibble-mode" dynamic RAMs. Using this feature (known as burst mode), the bus bandwidth of memory read transactions is essentially doubled. Burst mode transactions have the further benefit of allowing the cache to operate more efficiently by guaranteeing a high probability that the contents of the accessed memory will be present in the cache.

The Z280 MPU supports Zilog's Extended Processor Architecture (EPA) in a number of ways. It is capable of trapping Extended Processor Unit (EPU) instructions in order to perform software emulation of the EPU. With its 16-bit external bus interface, the Z280 MPU directly interfaces with an EPU and operates in a manner that is completely transparent to the user and the program.

Multiprocessor system architectures are also supported by the Z280 MPU. When operating in multiprocessor mode, the Z280 MPU's Local Address register is used to distinguish between local and global memory access. Global accesses are controlled through a global request and global acknowledge protocol.

The pin functions and the pin assignments of the Z280 MPU are illustrated in Figures 1 and 2. Figure 3 shows the block diagram.

Z280 CPU

User and System Modes of Operation

The Z280 CPU can operate in either user or system mode. In user mode, some instructions cannot be executed and some registers of the CPU are inaccessible. In general, this mode of operation is intended for use by application programs. In system mode, all of the instructions can be executed and all of the CPU registers can be accessed. This mode is intended for use with programs that perform operating system functions. This separation of CPU resources promotes the integrity of the system, since programs operating in user mode cannot access those aspects of the CPU that deal with system interface events.

To further support the dual user/system mode, there are two Stack Pointers—one for the user stack and another for the system stack. These two stacks facilitate the task switching involved when interrupts or traps occur. To ensure that the user stack is free of system information, the information saved on the occurrence of interrupts or traps is always pushed onto the system stack before the new program status is loaded.

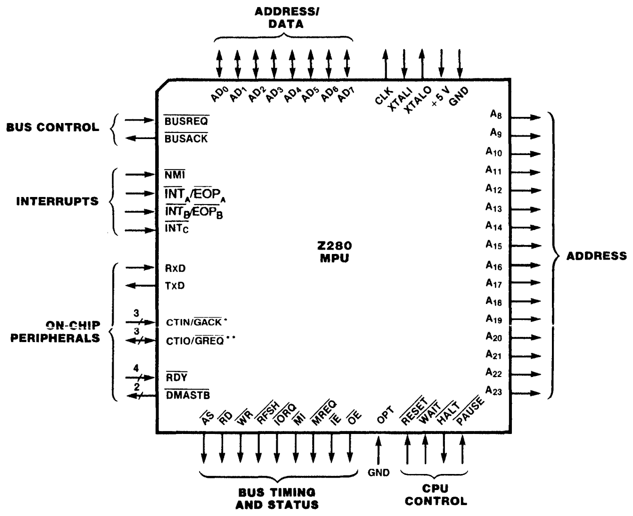


Figure 1a. Z280 Pin Functions, Z80 Bus Configuration (Input OPT tied to GND)

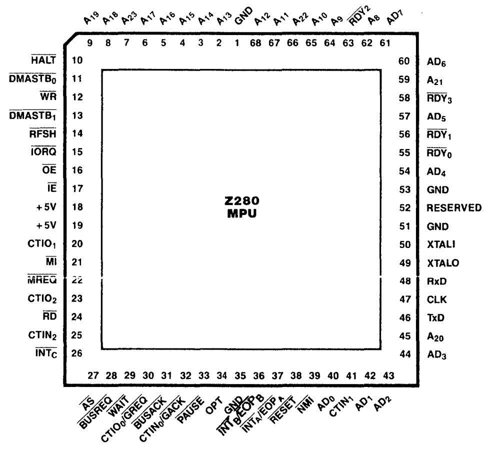


Figure 1b. Z280 Pin Assignments, Z80 Bus (Input OPT tied to GND)

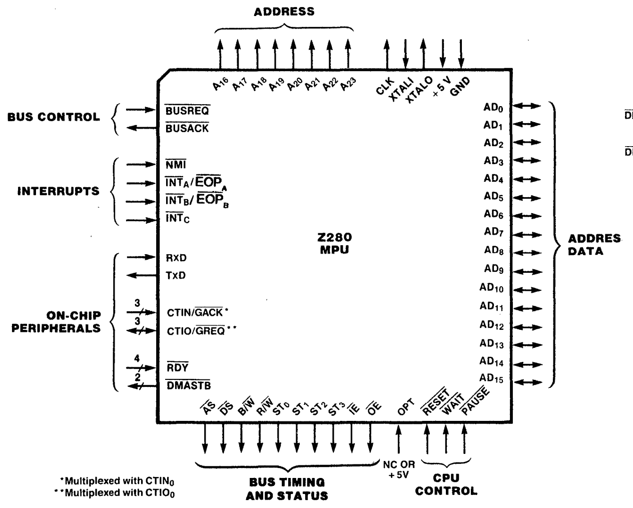


Figure 2a. Z280 Pin Functions, Z-BUS Configuration (Input OPT tied to +5V or not connected)

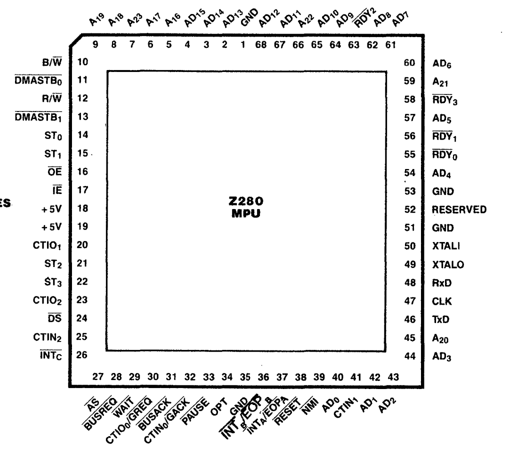
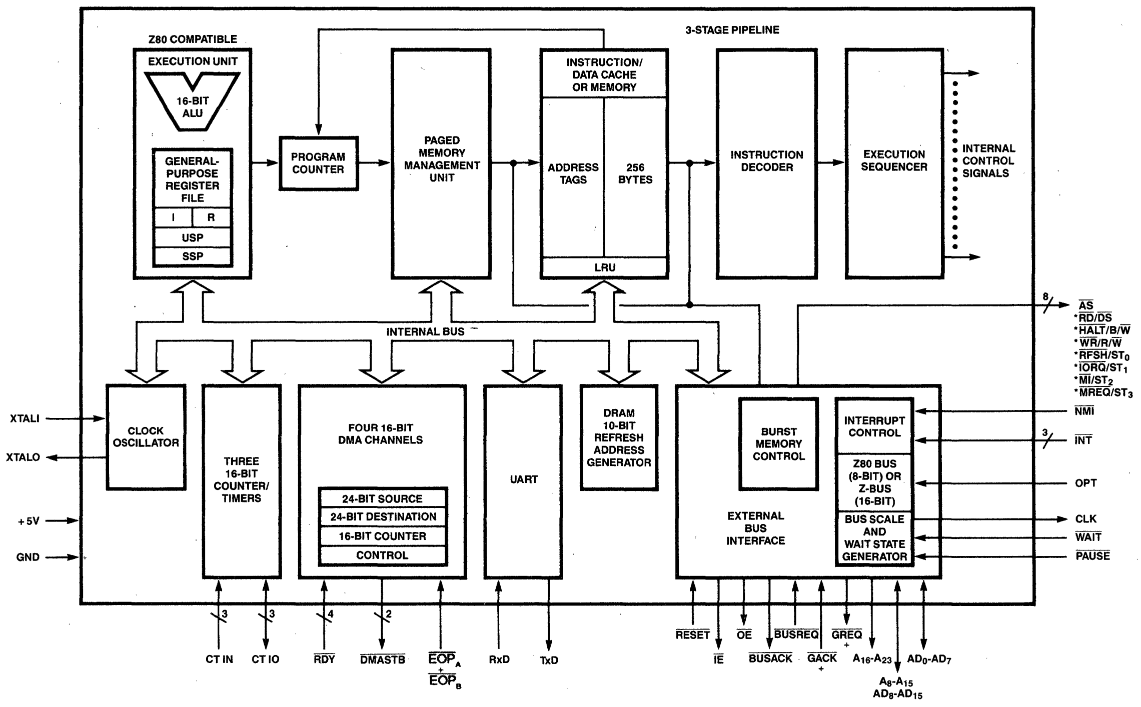


Figure 2b. Z280 Pin Assignments, Z-BUS (Input OPT tied to +5V or not connected)



* Signal definition depends on OPT.
+ $\overline{EOP}_A$ shares with $\overline{INT}_A$.
+ $\overline{EOP}_B$ shares with $\overline{INT}_B$.
+ $\overline{GACK}$ shares with CTIN$_0$.
+ $\overline{GREQ}$ shares with CTIO$_0$.

Figure 3. Z280 MPU Block Diagram.

Address Spaces

The Z280 CPU architecture supports four distinct address spaces corresponding to the different types of locations that can be accessed by the CPU. These four address spaces are:

- CPU register space
- CPU control and status register space
- Memory address space
- I/O address space

CPU Register Space. The CPU register space (Figure 4) consists of all of the registers in the CPU register file. The CPU registers are used for data and address manipulation. Access to these registers is specified in the instruction. The CPU registers are labeled A, F, B, C, D, E, H, L, A′, F′, B′, C′, D′, E′, H′, L′, IX, IY, SSP, USP, PC, I, and R.

CPU Control and Status Register Space. The CPU control register space consists of all of the control and status registers found in the CPU control register file (Figure 5). These registers govern the operation of the CPU and are accessible only by the privileged Load Control instruction. The registers in the CPU control file consist of the Bus Timing and Control register, Bus Timing and Initialization register, Local Address register, Cache Control register, Master Status register, Interrupt Status register, Interrupt/Trap Vector Table Pointer, I/O Page register, Trap Control register, and System Stack Limit register.

Memory Address Space. Two memory address spaces are supported by the Z280 CPU; one for user and one for system mode of operation. They are selected by the User/System Mode (U/S) bit in the Master Status register, which governs the selection of Page Descriptor registers during address translation.

Each address space can be viewed as a string of 64K bytes numbered consecutively in ascending order. The 8-bit byte is the basic addressable element in the memory address spaces. However, there are other addressable data elements: bits, 2-byte words, byte strings and multiple-byte EPU operands.

The address of a multiple-byte entity is the address of the byte with the lowest address. Multiple-byte entities can be stored beginning at either even or odd memory addresses.

I/O Address Space. I/O addresses are generated only by the I/O instructions (IN, OUT, and the I/O block move instructions). Logical I/O addresses are eight bits in length, augmented by the A register on lines A_8-A_{15} in Direct Address addressing mode and by the B register on lines A_8-A_{15} in Indirect Register addressing mode and for block I/O instructions. The 16-bit logical I/O address is always extended by appending the contents of the 8-bit page register to the augmented I/O address. Thus the complete address generated to address an I/O port consists of an I/O page number on A_{23}-A_{16}, the contents of the A or B register on A_8-A_{15}, and the 8-bit I/O address on A_7-A_0.

Unlike memory references, in which a 16-bit word store or fetch can generate two memory references, an I/O word store or fetch is always one I/O bus transaction, regardless of bus size or I/O port address. Note, however, that on-chip peripherals with word registers are accessed via word I/O instructions for those 16-bit registers, regardless of the external bus size (Table 1).

Data Types

The CPU can operate on bits, binary-coded decimal (BCD) digits (4 bits), bytes (8 bits), words (16 bits), byte strings, and word strings. Bits in registers or memory can be set, cleared, and tested. BCD digits, packed two to the byte, can be manipulated with the Decimal Adjust Accumulator instruction (in conjunction with binary addition and subtraction) and the Rotate Digit instructions. Bytes are operated on by 8-bit load, arithmetic, logical, and shift and rotate instructions. Words are operated on in a similar manner by the 16-bit load and 16-bit arithmetic instructions. Block move and search operations can manipulate byte strings up to 64K bytes long. Block I/O word instructions can manipulate word strings up to 32K words long. To support EPU operations, byte strings up to 16 bytes in length can be transferred by the CPU.

CPU Registers

The Z280 MPU contains 23 programmable registers (Figure 4) in the CPU register address space.

Primary and Working Register Set. The working register set is divided into the two 8-bit register files—the primary file and alternate (designated by ') file. Each file contains an 8-bit accumulator (A), a Flag register (F), and six general-purpose registers (B, C, D, E, H, and L). Only one file can be active at any given time. Upon reset, the primary register file is active. Exchange instructions allow the programmer to exchange the active file with the inactive file.

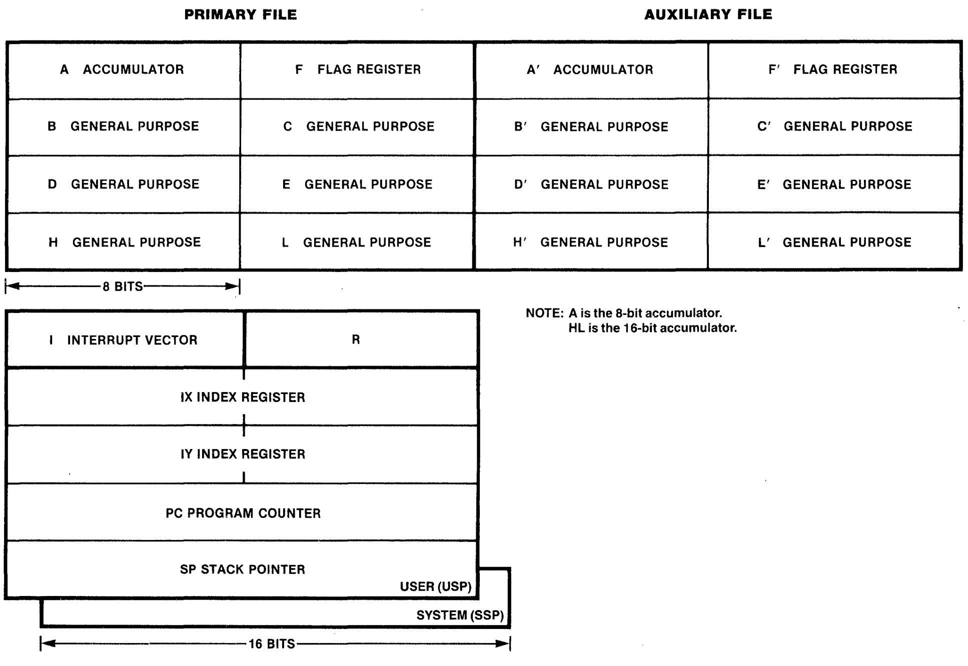


Figure 4. CPU Register Configuration

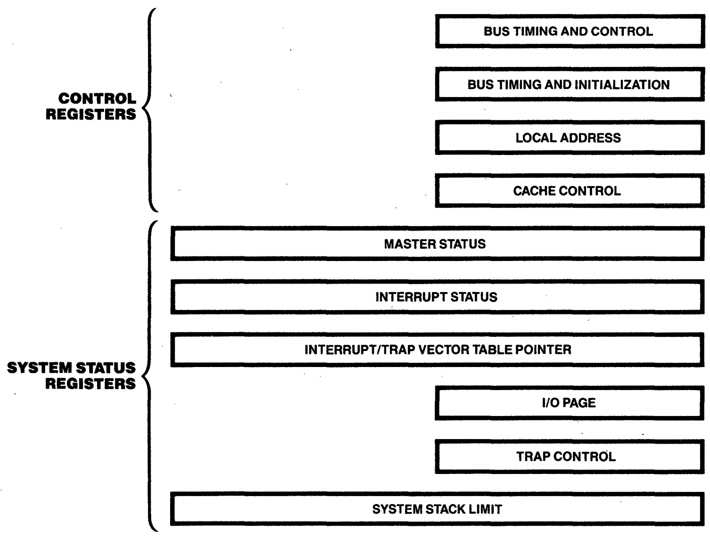


Figure 5. CPU Control and Status Registers

The accumulator is the destination register for 8-bit arithmetic and logical operations. The six general-purpose registers can be paired (BC, DE, and HL) to form three 16-bit general-purpose registers. The HL register pair serves as a 16-bit accumulator for 16-bit arithmetic operations.

CPU Flag Register. The Flag register contains six flags that are set or reset by various CPU operations. This register is illustrated in Figure 6.

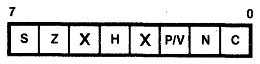


Figure 6. CPU Flag Register

The flags in this register are:

Carry (C). This flag is set when an add instruction generates a carry or a subtract instruction generates a borrow. Certain logical and rotate and shift instructions affect the Carry flag.

Add/Subtract (N). This flag is used by the Decimal Adjust Accumulator instruction to distinguish between add and subtract operations. The flag is set for subtract operations and cleared for add operations.

Parity/Overflow (P/V). During arithmetic operations this flag is set to indicate a two's complement overflow. During logical and rotate operations, this flag is set to indicate even parity of the result or cleared to indicate odd parity.

Half Carry (H). This flag is set if an 8-bit arithmetic operation generates a carry or borrow between bits 3 and 4, or if a 16-bit operation generates a carry or borrow between bits 11 and 12. This bit is used to correct the result of a packed BCD addition or subtract operation.

Zero (Z). This flag is set if the result of an arithmetic or logical operation is a zero.

Sign (S). This flag stores the state of the most significant bit of the accumulator. The Sign flag is also used to indicate the results of a test and set instruction.

Dedicated MPU Registers

Index Registers. The two Index registers, IX and IY, each hold a 16-bit base address that is used in the Indexed addressing mode. The Index registers can also function as general-purpose registers with the upper and lower bytes capable of being accessed individually. The high and low bytes of the IX register are called IXH and IXL. The high and low bytes of the IY register are called IYH and IYL.

Interrupt Register. The Interrupt register (I) is used in interrupt mode 2 to generate a 16-bit indirect logical address to an interrupt service routine. The Interrupt register supplies the upper eight bits of the indirect address and the interrupting peripheral supplies the lower eight bits.

Program Counter. The Program Counter (PC) is used to sequence through instructions in the currently executing program and to generate relative addresses. The Program Counter contains the 16-bit logical address of the current instruction being fetched from memory.

R Register. The R register can be used as a general-purpose 8-bit read/write register. The R register is not associated with the refresh address and its contents are changed only by the user.

NOTE: To be compatible with possible future enhancements, a user should write 0's into reserved register bits. A user should not rely on values read from reserved register bits. In figures and tables, unless otherwise noted, reserved bits are labeled with "X".

Table 1. On-Chip Peripheral I/O Port Addresses

Peripheral	Address (Hexadecimal)			
Refresh Rate Register	FFxxE8			
UART				
Configuration	FExx10			
Transmitter Control/Status	FExx12			
Receiver Control/Status	FExx14			
Receiver Data	FExx16			
Transmitter Data	FExx18			
MMU				
Master Control	FFxxF0			
Page Descriptor Register Pointer	FFxxF1			
Descriptor Select Port	FFxxF5			
Block Move Port	FFxxF4			
Invalidation I/O Port	FFxxF2			
Page Descriptor Registers*				
User PDR 0	00			
User PDR 1	01			
. .	. .			
User PDR 14	0E			
User PDR 15	0F			
System PDR 0	10			
System PDR 1	11			
. .	. .			
System PDR 14	1E			
System PDR 15	1F			
DMA				
Master Control	FFxx1F			
	DMA0	**DMA1**	**DMA2**	**DMA3**
Destination Address (bits 0-11)	FFxx00	FFxx08	FFxx10	FFxx18
Destination Address (bits 12-23)	FFxx01	FFxx09	FFxx11	FFxx19
Source Address (bits 0-11)	FFxx02	FFxx0A	FFxx12	FFxx1A
Source Address (bits 12-23)	FFxx03	FFxx0B	FFxx13	FFxx1B
Count	FFxx04	FFxx0C	FFxx14	FFxx1C
Transaction Descriptor	FFxx05	FFxx0D	FFxx15	FFxx1D
Counter/Timer	**C/T0**	**C/T1**	**C/T2**	
Configuration	FExxE0	FExxE8	FExxF8	
Command/Status	FExxE1	FExxE9	FExxF9	
Time Constant	FExxE2	FExxEA	FExxFA	
Count-Time	FExxE3	FExxEB	FExxFB	

*The Page Descriptor register address must be loaded into the Page Descriptor Register Pointer in order to access that Page Descriptor register.

Stack Pointers. Two hardware Stack Pointers, the User Stack Pointer (USP) and the System Stack Pointer (SSP), support the dual mode of operation of the microprocessor. The SSP is used for saving information when an interrupt or trap occurs and for supporting subroutine calls and returns in system mode. The USP is used for supporting subroutine calls and returns in user mode.

Status and Control Registers. There are ten status and control registers available to the programmer in the Z280 MPU. Table 2 shows the addresses occupied by the registers in the status and control register addressing space.

Table 2. Status and Control Register I/O Port Addresses

Control Register Name	Address (Hexadecimal)
Bus Timing and Control	Control 02
Bus Timing and Initialization	Control FF
Cache Control[1]	Control 12
Interrupt Status	Control 16
Interrupt/Trap Vector Table	Control 06
I/O Page Register	Control 08
Local Address Register[2]	Control 14
Master Status (MSR)	Control 00
Stack Limit	Control 04
Trap Control	Control 10

NOTES:
1. See section on on-chip memory for register description.
2. See section on multiprocessing mode of operation for register description.

Bus Timing and Control Register. This 8-bit register (Figure 7) governs the timing of transactions to high memory addresses and the daisy-chain timing for interrupt requests, as well as the functionality of requests on the various Z280 MPU interrupt request lines.

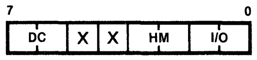


Figure 7. Bus Timing and Control Register

The fields in this register are:

I/O Wait Insertion (I/O). This 2-bit field specifies the number of additional wait states (in addition to the one automatically inserted for I/O) to be inserted by the CPU in both I/O transactions and vector response timing (00 = none, 01 = one, 10 = two, 11 = three).

High Memory Wait Insertion (HM). This 2-bit field specifies the number of automatic wait states (00 = none, 01 = one, 10 = two, 11 = three) for the CPU to insert in memory transactions when the MMU is enabled and there is a 1 in bit 15 of the selected Page Descriptor register.

Daisy Chain Timing (DC). This 2-bit field determines the number of additional automatic wait states the CPU inserts while the interrupt acknowledge daisy chain is settling (00 = none, 01 = one, 10 = two, 11 = three). A value of 01 in the DC field indicates that one additional cycle will be added to the four cycles that normally elapse between interrupt acknowledge, $\overline{AS}$ and $\overline{DS}$ (or $\overline{IORQ}$) assertions.

Bus Timing and Initialization Register. This 8-bit register (Figure 8) is used to specify the duration of control signals for the external interface bus when the MMU is disabled or when the MMU is enabled and there is a 0 in bit 15 of the selected Page Descriptor register. It also controls the relationship between internal processor clock rates and bus timing. It can be programmed by external hardware upon reset.

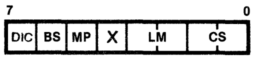


Figure 8. Bus Timing and Initialization Register

During reset this register is initialized to one of two settings, depending on the state of the $\overline{WAIT}$ input line on the rising edge of Reset: if the $\overline{WAIT}$ line is not asserted, the register is set to 00_H. If the $\overline{WAIT}$ line is asserted during reset, then this register is set to the contents of the AD lines.

The fields in this register are:

Clock Scaling (CS). This 2-bit field specifies the scaling of the CPU clock for all bus transactions (00 = one bus clock cycle is equal to two internal processor clock cycles, 01 = bus clock cycle is equal to the internal processor clock cycle, 10 = one bus clock cycle is equal to four internal processor clock cycles, 11 = reserved). This field cannot be modified by software.

Low Memory Wait Insertion (LM). This 2-bit field specifies the number of automatic wait states (00 = none, 01 = one, 10 = two, 11 = three) for the CPU to insert in memory transactions when the MMU is disabled or when the MMU is enabled and there is a 0 in bit 15 of the selected Page Descriptor register.

Multiprocessor Configuration Enable (MP). This 1-bit field enables the multiprocessor mode of operation (0 = disabled, 1 = enabled). (See the Multiprocessor Mode section.)

Bootstrap Mode Enable (BS). This 1-bit field enables the bootstrap mode of operation (0 = disabled, 1 = enabled). (See the UART section for details about bootstrap mode.)

***Direct Input Clock Option (DIC).* This bit when set (0=disabled, 1=enabled) selects the direct clock source option for the XTALI input. In this mode, the crystal oscillator and divide by 2 circuits are bypassed and XTALI input is used to directly generate the MPU internal clocks. The XTALI input must have TTL levels, 50% duty cycle, and 10MHz maximum frequency. When disabled, the input frequency is divided by 2 to generate the internal processor clock. A maximum crystal or input clock frequency of 20MHz is supported in this case.**

Interrupt Status Register. This 16-bit register (Figure 9) indicates which interrupt mode is in effect and which interrupt sources have interrupt requests pending. It also contains the bits that specify whether the interrupt inputs are to be vectored. Only the interrupt vector enable bits are writeable; all other bits are read-only.

15														0
I_C	I_B	I_A	I_{NMI}	X	X	IM	X	IP_6	IP_5	IP_4	IP_3	IP_2	IP_1	IP_0

Figure 9. Interrupt Status Register

The fields in this register are:

Interrupt Request Pending (IP). When bit IP_n is set to 1, an interrupt request from sources at level n is pending. (See the Interrupt and Trap Structure section.)

Interrupt Mode (IM). A value of n in this 2-bit field indicates that interrupt mode n is in effect. This field can be changed by executing the IM instruction.

Interrupt Vector Enable (I). These four bits indicate whether each of the four interrupt inputs are to be vectored. When I_n is set to 1, interrupts on the Interrupt n line are vectored when the CPU is in interrupt mode 3; when cleared to 0, all interrupts on this line use the same entry in the Interrupt/Trap Vector Table. These bits are ignored except in interrupt mode 3.

***Interrupt/Trap Vector Table Pointer.* This 16-bit register (Figure 10) contains the most significant 12 bits of the physical address at the beginning of the Interrupt/Trap Vector Table: the lower 12 bits of the physical address are assumed to be 0.**

15															0
A_{23}	A_{22}	A_{21}	A_{20}	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	X	X	X	X

Figure 10. Interrupt/Trap Vector Table Pointer

I/O Page Register. This 8-bit register (Figure 11) indicates the bits to be appended to the 16 bits that are output during the I/O address phase of I/O transactions.

7							0
A_{23}	A_{22}	A_{21}	A_{20}	A_{19}	A_{18}	A_{17}	A_{16}

Figure 11. I/O Page Register

Master Status Register. The Master Status register (Figure 12) is a 16-bit register containing status information about the currently executing program. This register is cleared to 0 during reset.

15															0
X	U/S	X	BH	X	X	SSP	SS	X	E_6	E_5	E_4	E_3	E_2	E_1	E_0

Figure 12. Master Status Register

The fields in this register are:

Interrupt Request Enable (E_n). There are seven Interrupt Enable bits, one for each type of maskable interrupt source (both external and internal). When bit E_n is set to 1, interrupt requests from sources at level n are accepted by the CPU; when this bit is cleared to 0, interrupt requests at level n are not accepted.

Single-Step (SS). While this bit is set to 1, the CPU is in single-stepping mode; while this bit is cleared to 0, automatic single-stepping is disabled. This bit is automatically cleared when a trap or interrupt is taken.

Single-Step Pending (SSP). While this bit is set to 1, the CPU generates a trap prior to executing an instruction. The SS bit is automatically copied into this field at the completion of each instruction. This bit is automatically cleared to 0 when a Single-Step, Page Fault, Privileged Instruction, Breakpoint-on-Halt or Division trap is taken so that the SSP bit in the saved Master Status register is cleared to 0.

Breakpoint-on-Halt Enable (BH). While this bit is set to 1, the CPU generates a Breakpoint trap whenever a HALT instruction is encountered; while this bit is cleared to 0, the HALT instruction is executed normally.

User/System Mode (U/S). While this bit is cleared to 0, the CPU is in the system mode of operation; while it is set to 1 the CPU is in the user mode of operation.

***System Stack Limit Register.* This 16-bit register (Figure 13) indicates when a System Stack Overflow Warning trap is to be generated. If enabled, by setting a control bit in the Trap Control register, pushes onto the system stack cause the 12 most significant bits in this register to be compared to the upper 12 bits of the system Stack Pointer and a trap is generated if they match.**

15															0
A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	X	X	X	X

Figure 13. System Stack Limit Register

Trap Control Register. This 8-bit register (Figure 14) enables the maskable traps. Upon reset this register is initialized to all 0s.

7							0
X	X	X	X	X	I	E	S

Figure 14. Trap Control Register

The bits in this register are:

System Stack Overflow Warning (S). While this bit is set to 1, the CPU generates a Stack Overflow Warning trap when the system stack enters the specified region of memory.

EPU Enable (E). While this bit is cleared to 0, the CPU generates a trap whenever an EPA instruction is encountered.

Inhibit User I/O (I). While this bit is set to 1, the CPU generates a Privileged Instruction trap when an I/O instruction is encountered in user mode.

Cache Control and Local Address Registers. See the On-Chip Memory section for information about the Cache Control register and the Multiprocessor Mode section for information about the Local Address register.

Interrupt and Trap Structure

The Z280 MPU provides a very flexible and powerful interrupt and trap structure. Interrupts are external asynchronous events requiring CPU attention and are generally triggered by peripherals needing service. Traps are synchronous events resulting from the execution of certain instructions.

Interrupts. Two types of interrupt, nonmaskable and maskable, are supported by the Z280 MPU. The nonmaskable interrupt ($\overline{\text{NMI}}$) cannot be disabled (masked) by software and is generally reserved for highest priority external events that require immediate attention. Maskable interrupts, however, can be selectively disabled by software. Both nonmaskable and maskable interrupts can be programmed to be vectored or nonvectored. Interrupts are always accepted between instructions and acknowledged after execution of the prior instruction is complete. The block move, search, and I/O instructions can be safely interrupted after any iteration and restarted after the interrupt is serviced.

Interrupt Sources. The Z280 MPU accepts nonmaskable interrupts on the $\overline{\text{NMI}}$ pin only. The Z280 MPU accepts maskable interrupts on the $\overline{\text{INT}}$ pins and from the on-chip counter/timers, DMA channels, and the UART receiver and transmitter.

Interrupt Lines A, B, and C can be selectively programmed to support vectored interrupts by setting the appropriate bits in the Interrupt Status register. The external interrupts can be programmed to be vectored or nonvectored in interrupt mode 3.

Interrupt Modes of Operation. The CPU has four modes of interrupt handling. The first three modes extend the Z80 interrupt modes to accommodate additional interrupt input lines in a compatible fashion. The fourth mode provides more flexibility in handling the interrupts. On-chip peripherals use the fourth mode regardless of which mode is selected for externally generated interrupt requests. The interrupt mode is selected by using the privileged instructions IM 0, IM 1, IM 2, or IM 3. On reset, the Z280 MPU is automatically set to interrupt mode 0. The current interrupt mode in effect can be read from the Interrupt Status register.

Mode 0. This mode is identical to the 8080 interrupt response mode. With this mode, the interrupting device on any of the maskable interrupt lines can place a call or restart instruction on the data bus and the CPU will execute it. As a result, the interrupting device, instead of the memory, provides the next instruction to be executed.

Mode 1. When this mode is selected, the CPU responds to a maskable external interrupt by executing a restart to the logical address 0038_H in the system program address space.

Mode 2. This mode is a vectored interrupt response mode. With a single 8-bit byte from the interrupting device, an indirect call can be made to any memory location. With this mode the system maintains a table of 16-bit starting addresses for every interrupt service routine. This table can be located anywhere in the system mode logical data address space on a 256-byte boundary. When an interrupt is accepted, a 16-bit pointer is formed to obtain the desired interrupt service routine starting address from the table. The upper eight bits of this pointer are formed from the contents of the I register. The lower eight bits of the pointer must be supplied by the interrupting device. The 16-bit pointer so formed is treated as a logical address in the system data address space, which can be translated by the MMU to a physical address.

Mode 3. This is the intended mode of operation for systems that take advantage of the enhancements of the Z280 microprocessor (such as single-step and user/system mode) since the Master Status register is automatically saved and another loaded for the interrupts. Also, vector tables can be used for the external interrupt sources to provide more interrupt vectors for the Z8000® family, Z80 family, and Z8500 Universal Peripherals. When an interrupt request (either maskable or nonmaskable) is accepted, the Master Status register, the address of the next instruction to be executed, and a 16-bit "reason code" are pushed onto the system stack. A new Master Status register and Program Counter are then fetched from the Interrupt/Trap Vector Table. The "reason code" for externally generated interrupts is the contents of the bus during the interrupt acknowledge sequence; for **8-bit data buses, the least significant byte of the reason code is all 1's. For interrupts generated by on-chip peripherals, the reason code identifies which peripheral generated the interrupt and is identical to the vector address in the Interrupt/Trap Vector Table. The Interrupt/Trap Table Pointer is used to reference the table.**

Traps. The Z280 CPU supports eight traps that are generated internally. The following traps can be disabled: the EPA trap, which allows software to emulate an EPU; the Stack Warning trap, which is taken at the end of an instruction causing the trap; the Breakpoint-on-Halt trap, which is taken when a HALT instruction is encountered; and the Single-Step trap, which is taken for each instruction. In addition, I/O instructions can be specified as privileged instructions. Traps cause the instruction to be terminated without altering CPU registers (except for the System Stack

Pointer, which is modified when the program status is pushed onto the system stack).

The saving of the program status on the system stack and the fetching of a new program status from the Interrupt/Trap Vector Table is the same in any interrupt mode of operation.

Traps can only occur if the trap generating features of the Z280 CPU (such as System Stack Overflow warning) have been explicitly enabled. Traps cannot occur on instructions of the Z80 instruction set unless explicitly enabled by the operating system using Z280 CPU extensions.

Extended Instruction. This trap occurs when the CPU encounters an extended instruction while the Extended Processing Architecture (EPA) bit in the Trap Control register is 0. Four trap vectors are used by the EPA trap—one for each type of EPA instruction. This greatly simplifies trap handlers that use I/O instructions to access an EPU or software to emulate an EPU.

Privileged Instruction. This trap occurs whenever an attempt is made to execute a privileged instruction while the CPU is in user mode (User/System Mode control bit in the Master Status register is 1).

System Call. This trap occurs whenever a System Call (SC) instruction is executed.

Access Violation. This trap occurs whenever the MMU's translation mode is enabled and an address to be translated is invalid or (for writes) is write-protected.

System Stack Overflow Warning. This trap occurs only while the Stack Overflow Warning bit in the Trap Control register is set to 1. For each system stack push operation, the most significant bits in the Stack Pointer register are compared with the contents of the Stack Limit register and a trap is signaled if they match. The Stack Overflow Warning bit is then automatically cleared in order to prevent repeated traps.

Division Exception. This trap occurs whenever the divisor is zero (divide-by-zero case) or the true quotient cannot be represented in the destination precision (overflow); the CPU flags are set to distinguish these two cases.

Single-Step. This trap occurs before executing an instruction if the Single-Step Pending control bit in the Master Status register is set to 1. Two control bits in the Master Status register are used for the Single-Step trap. The Single-Step bit (bit 8), on being set when previously clear, causes a trap to occur after the execution of the next instruction. While this bit is set to 1, if an instruction execution causes a trap, the Single-Step trap occurs after the execution of the trap-handling routine. The Single-Step Pending bit (bit 9), is used by the processor to ensure that only one Single-Step trap occurs for each instruction executed while the Single-Step bit is set to 1.

Breakpoint-on-Halt. This trap occurs whenever the Breakpoint-on-Halt control bit in the Master Status register is 1 and a HALT instruction is encountered.

Interrupt and Trap Disabling. Maskable interrupts can be enabled or disabled independently via software by setting or clearing the appropriate control bits in the Master Status register.

A 7 bit mask field in the Master Status register indicates which of the requested interrupts will be accepted. Interrupt requests are grouped as follows, with each group controlled by a separate Interrupt Enable control bit. The list is presented in order of decreasing priority, with sources within a group listed in order of descending priority.

- Maskable Interrupt A line (bit 0)
- Counter/Timer 0, DMA0 (bit 1)
- Maskable Interrupt B line (bit 2)
- Counter/Timer 1, UART receiver, DMA1 (bit 3)
- Maskable Interrupt C line (bit 4)
- UART Transmitter, DMA2 (bit 5)
- Counter/Timer 2, DMA3 (bit 6)

When a source of interrupts has been disabled, the CPU ignores any interrupt request from that source.

The System Stack Overflow Warning trap, Privileged Instruction trap (I/O instructions in user mode), or Extended Instruction trap can be enabled by setting control bits in the Trap Control register, and the Single-Step and Breakpoint-on-Halt trap can be enabled by setting control bits in the Master Status register; these are the only traps that can be disabled.

Interrupt/Trap Vector Table. The format of the Interrupt/Trap Vector Table consists of pairs of Master Status register and Program Counter words, one pair for each separate on-chip interrupt or trap source. For each external interrupt, there is a separate Master Status register word and Program Counter word (for use if the input is not vectored). If the external interrupt is vectored, a vector table consisting of one Program Counter word for each of the 128 possible vectors that can be returned for each input line is used instead of the dedicated Program Counter word; thus for vectored interrupts, there is only one Master Status register for each interrupt type.

The format of the Interrupt/Trap Vector Table is shown in Table 3.

Table 3. Interrupt/Trap Vector Table

Address (Hexadecimal)	Contents
00	Reserved
04	NMI Vector
08	Interrupt Line A Vector
0C	Interrupt Line B Vector
10	Interrupt Line C Vector
14	Counter/Timer 0 Vector
18	Counter/Timer 1 Vector
1C	Reserved
20	Counter/Timer 2 Vector
24	DMA0 Vector
28	DMA1 Vector
2C	DMA2 Vector
30	DMA3 Vector
34	UART Receiver Vector
38	UART Transmitter Vector
3C	Single-Step Trap Vector
40	Breakpoint-on-Halt Trap Vector
44	Division Exception Trap Vector
48	Stack Overflow Warning Trap Vector
4C	Page Fault Trap Vector
50	System Call Trap Vector
54	Privileged Instruction Trap Vector
58	EPU ← Memory Trap Vector
5C	Memory ← EPU Trap Vector
60	A ← EPU Trap Vector
64	EPU Internal Operation Trap Vector
68-6C	Reserved
70-16E	**128 Program Counter Values for NMI and Interrupt Line A Vectors (MSR from 04 and 08, respectively)**
170-26E	**128 Program Counter Values for Interrupt Line B Vectors(MSR from 0C)**
270-36E	**128 Program Counter Values for Interrupt Line C Vectors(MSR from 10)**

Addressing Modes

Addressing modes (Figure 15) are used by the CPU to calculate the effective address of an operand needed for execution of an instruction. Nine addressing modes are supported by the Z280 CPU. Of these nine, four are additions to the Z80 addressing modes (Indexed with 16-bit displacement, Stack Pointer Relative, Program Counter Relative, and Base Index) and the remaining five modes are either existing or extensions to the existing Z80 addressing modes.

Register. The operand is one of the 8-bit registers (A, B, C, D, E, H, L, IXH, IXL, IYH or IYL); or one of the 16-bit registers (BC, DE, HL, IX, IY, or SP), or one of the special byte registers (I or R).

Immediate. The operand is in the instruction itself and has no effective address.

Indirect Register. The contents of a register specify the effective address of an operand. The HL register is the register used for memory accesses. (For the Load To or Load From Accumulator instruction, BC and DE can also be used for indirection; for the JP instruction, IX and IY can also be used for indirection.) The C register is used for I/O and control register space accesses.

Direct Address. The effective address of the operand is the location whose address is contained in the instruction. Depending on the instruction, the specified operand is either in the I/O or data memory address space.

Indexed. The effective address of the operand is the location specified by adding the 16-bit address contained in the instruction to a two's complement "index" contained in the HL, IX, or IY register.

Short Index. The effective address of the operand is the location computed by adding the 8-bit two's complement signed displacement contained in the instruction to the contents of the IX or IY register. This addressing mode is equivalent to the Z80 CPU indexed mode.

Program Counter Relative. An 8- or 16-bit displacement contained in the instruction is added to the Program Counter to generate the effective address of the operand.

Stack Pointer Relative. The effective address of the operand is the location computed by adding a 16-bit two's complement displacement contained in the instruction to the contents of the Stack Pointer.

Base Index. The effective address of the operand is the location whose address is computed by adding the contents of HL, IX, or IY to the contents of another of these three registers.

EXTENDED PROCESSING ARCHITECTURE

Features

The Zilog Extended Processing Architecture (EPA) provides an extremely flexible and modular approach to expanding both the hardware and software capabilities of the Z280 CPU. Features of the EPA include:

- Allows Z280 CPU instruction set to be extended by external devices.
- Increases throughput of the system by using up to four specialized external processors in parallel with the CPU.
- Permits modular design of Z280 CPU-based systems.
- Provides easy management of multiple microprocessor configurations via "single instruction stream" communication.
- Direct interconnection between EPUs and Z280 MPU requires no additional external supporting logic.
- EPUs can be added as the system grows and as EPUs with specialized functions are developed.

General Description

The processing power of the Zilog Z-BUS Z280 microprocessor can be boosted beyond its intrinsic capability by the Extended Processing Architecture (EPA). The EPA allows the Z280 CPU to accommodate up to four Extended Processing Units (EPUs), which perform specialized functions in parallel with the CPU's main instruction execution stream.

The EPUs connect directly to the Z-BUS and continuously monitor the CPU instruction stream for an instruction intended for the EPU (template). When a template is detected, the appropriate EPU responds, obtaining or placing data or status information on the Z-BUS by using the Z280 CPU-generated control signals and performing its function as directed.

The CPU is responsible for instructing the EPU and delivering operands and data to it. The EPU recognizes templates intended for it and executes them, using data supplied with the template and/or data within its internal registers. There are three classes of EPU instructions:

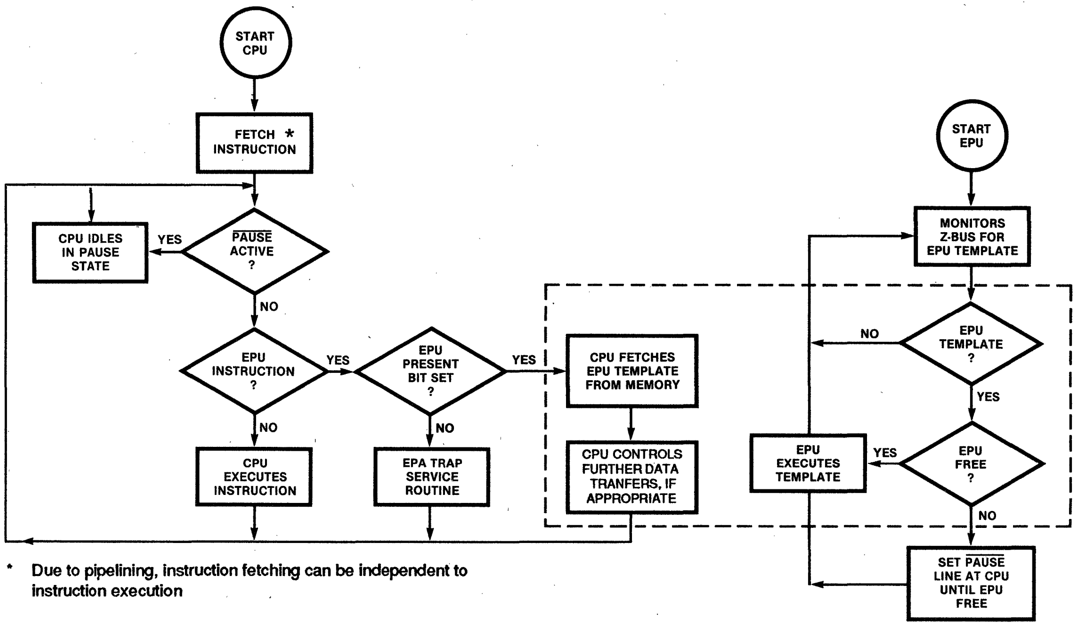
- Data transfers between main memory and EPU registers
- **Data transfers between CPU registers and EPU status registers**
- EPU internal operations

Six addressing modes can be utilized with transfers between EPU registers and the MPU and main memory:

- Indirect Register
- Direct Address
- Indexed
- Program Counter Relative
- Stack Pointer Relative
- Base Index

In addition to the hardware-implemented capabilities of the EPA, there is an extended instruction trap mechanism to permit software simulation of EPU functions. An EPU present bit in the Z280 MPU Trap Control register indicates **whether actual EPUs are present or not. If not, the CPU generates a trap when an extended instruction is detected, and a software "trap handler" can emulate the desired EPU function. Thus, the EPA software trap routine supports systems not containing an EPU.**

EPA and CPU instruction execution are shown in Figure 16. If an instruction has been fetched and decoded, the CPU determines whether or not it is an EPU instruction. If the instruction is an EPU instruction, the state of the EPU Enable bit in the Trap Control register is examined. If the EPU Enable bit is reset (E = 0), the CPU generates a trap and the EPU instruction can be simulated by an EPU instruction trap software routine. However, if the EPU Enable bit is set (E = 1), indicating that an EPU is present in the system, then the 4-byte EPU template is fetched from memory. The fetching of the EPU template is indicated by the status lines ST_0-ST_3. Each EPU continuously monitors the Z-BUS and the status lines for its own templates. After fetching the EPU template, the CPU, if necessary, transfers appropriate data between the EPU and memory or between the CPU and the EPU. These transactions are indicated by unique encodings of the status lines. If the EPU is free when the template and the data appear, the EPU template is executed. If the EPU is still processing a previous instruction, the $\overline{\text{PAUSE}}$ line can be activated to halt further execution of CPU instructions until EPU execution is complete. After the execution of the template is complete, the EPU deactivates the $\overline{\text{PAUSE}}$ line and CPU instruction execution continues.



* **Due to pipelining, instruction fetching can be independent to instruction execution**

Figure 16. EPA and Z280 MPU Instruction Execution.

MEMORY MANAGEMENT

Features

- On-chip dynamic address translation
- Permits addressing of 16M bytes of physical memory
- Separate translation facilities for user and system modes
- Permits instructions and data to reside in separate memory areas.
- Write protection for individual pages of memory
- Aborts CPU on access violation to support virtual memory

General Description

The Z280 microprocessor contains an on-chip Memory Management Unit (MMU), which translates logical addresses into physical addresses. This allows access to more than 64K bytes of physical memory and provides memory protection features typical of those found on large systems. With the MMU, the CPU can access up to 16M bytes of physical memory. The MMU features a sophisticated trapping mechanism that generates page faults on error conditions. Instructions that are aborted by a page fault can be restarted in a manner compatible with virtual memory system requirements. On reset, the MMU features are not enabled, thus permitting logical addresses to pass to the physical memory untranslated.

The physical address space is expanded by dividing the 64K byte logical address space (the space manipulated by the program) into pages. The pages are then mapped (translated) into the larger physical address space of the Z280 microprocessor. The mapping process makes the user software addresses independent of the physical memory, so the user is freed from specifying where information is actually stored in physical memory. The actual size of the page depends on whether the program/data separation mode is enabled—if it is enabled, each page is 8K bytes in length, and if it is not enabled, the page length is 4K bytes. With the page mapping technique, 16-bit logical addresses can be translated into 24-bit physical addresses. Address translation can occur both in system and in user mode, with separate translation facilities available to each mode. The MMU further allows instruction references to be separated from data references, which enables programs of up to 64K bytes in length to manipulate up to 64K bytes of data without operating system intervention.

MMU Architecture

The Z280 MMU consists of two sets of sixteen Page Descriptor registers (Figure 17) that are used to translate addresses, a 16-bit control register that governs the translation facilities, a Page Descriptor Register Pointer, an I/O write-only port that can be used to invalidate sets of page descriptors, and two I/O ports for accesses to the Page Descriptor registers. One set of Page Descriptor registers is dedicated to the system mode of operation and the other set is dedicated to the user mode of operation.

While an address is being translated, attributes associated with the logical page containing that location are checked. The correct logical page is determined by the CPU mode (user or system), address space (program/data), and the four most significant bits of the logical address. Pages can be write-protected to prevent them from being modified by the executing task and can also be marked as non-cacheable to prevent information from being copied into the cache for later reference. The latter capability is useful in multiprocessor systems, to ensure that the processor always accesses the most current version of information being shared among multiple devices. The MMU also maintains a bit for each page that indicates if the page has been modified.

Each Page Descriptor register contains a Valid bit, which indicates that the descriptor contains valid information. Any attempt by the MMU to translate an address using an invalid descriptor generates a page fault. Valid bits for groups of Page Descriptor registers can be reset by writing to an MMU control port.

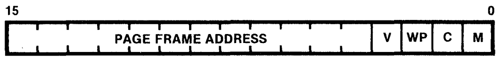


Figure 17. Page Descriptor Register

For each mode of CPU operation, the MMU can be configured to separate instruction fetches from data fetches, and thus separate the program address space from the data address space. When the program/data separation mode is in effect, the sixteen Page Descriptor registers for the current CPU mode of operation (user or system) are partitioned into two sets, one for instruction fetches and one for data fetches. An instruction fetch or data access using the Program Counter Relative addressing mode is translated by the MMU registers associated with the program address space; data accesses using other addressing modes and accesses to the Interrupt Vector Table in interrupt mode 2 use the MMU registers associated with the data address space. In this mode of MMU operation, the page size is 8192 bytes. There are two control bits in the MMU Master Control register that independently specify whether the user and system modes of MPU operation have separate program and data address spaces.

Each 16-bit Page Descriptor register consists of a 4-bit attribute field and a 12-bit page frame address field. The attribute field consists of the least significant bits of the descriptor and contains four control and status bits, listed below.

Modified (M). This bit is automatically set whenever a write is successfully performed to a logical address in this page; it can be cleared to 0 only by a software routine that loads the Page Descriptor register. If the Valid bit is 0, the contents of this bit are undefined.

Cacheable (C). While this bit is set to 1, information fetched from this page can be placed in the cache. While this bit is cleared to 0, the cache control mechanism is inhibited from retaining a copy of the information.

Write-Protect (WP). While this bit is set to 1, CPU writes to logical addresses in this page cause a page fault to be generated and prevent a write operation from occurring. While this bit is cleared to 0, all valid accesses are permitted.

Valid (V). While this bit is set to 1, the descriptor contains valid information. While this bit is cleared to 0, all CPU accesses to logical addresses in this page cause a page fault to be generated.

MMU Control Registers and I/O Ports

MMU operation is controlled by one control register and four dedicated I/O ports. The MMU Master Control register (Figure 18) determines the program/data address space separation in effect in both user and system modes and whether logical addresses generated in user and system mode will be translated by the MMU. Page Descriptor registers are accessed indirectly through the register address contained in the Page Descriptor Register Pointer. The Descriptor Select Port is used to access the Page Descriptor register that is pointed to by the Page Descriptor Register Pointer. After this access the Page Descriptor Register Pointer is left unchanged. The Block Move I/O Port is used to move blocks of words between the Page Descriptor registers and memory; reads or writes to this I/O port access data pointed to by the Page Descriptor Register Pointer, then increment the pointer by one. The Invalidation I/O Port is used to invalidate blocks of Page Descriptor registers; writes to this port cause the Valid bits in selected blocks of Page Descriptor registers to be cleared to 0, which indicates that the descriptors no longer contain valid information.

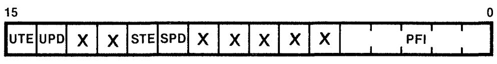


Figure 18. MMU Master Control Register

MMU Master Control Register. The MMU Master Control register (I/O address location FFxxF0) controls the operation of the MMU. This register contains four control bits; all other bits in this register must be cleared to 0. The four control bits of the MMU Master Control register are described below.

Page Fault Identifier (PFI). This 5-bit field latches information that indicates which Page Descriptor register was being accessed when the access violation was detected.

System Mode Program/Data Separation Enable (SPD). While this bit is set to 1, instruction fetches and data accesses via the PC Relative addressing mode use the system mode Page Descriptor registers 8-15, and data references that do not use the PC Relative addressing mode use the system mode Page Descriptor registers 0-7. While this bit is cleared to 0, system mode Page Descriptor registers 0-15 are used to translate instruction and data references.

System Mode Translate Enable (STE). While this bit is set to 1, logical addresses generated in the system mode of operation are translated. While this bit is cleared to 0, addresses are passed through the MMU extended with zeros in the most significant bits and no attribute checking or modified bit setting is performed.

User Mode Program/Data Space Separation Enable (UPD). While this bit is set to 1, instruction fetches and data accesses via the PC Relative addressing mode use the user mode Page Descriptor registers 8-15, and data references that do not use the PC Relative addressing mode use the user mode Page Descriptor registers 0-7. While this bit is cleared to 0, user mode Page Descriptor registers 0-15 are used to translate instruction and data references.

User Mode Translated Enable (UTE). While this bit is set to 1, logical addresses generated in the user mode of operation are translated. While this bit is cleared to 0, addresses are passed through the MMU extended with zeros in the most significant bits and no attribute checking or modified bit setting is performed.

Page Descriptor Register Pointer. Moves of data into and out of the MMU Page Descriptor registers use the Page Descriptor Register Pointer, which is at I/O address location FFxxF1. This 8-bit register contains the address of one of the Page Descriptor registers. When a word I/O instruction accesses I/O address FFxxF5 (Descriptor Select Port), this register is used to access a Page Descriptor register. When a word I/O instruction accesses I/O address FFxxF4 (Block Move I/O Port), this register is also used to access a Page Descriptor register, but after the access, this register is automatically incremented by one.

Descriptor Select Port. Moves of one word of data into and out of a Page Descriptor register are accomplished by writing and reading words to or from this dedicated I/O port at location FFxxF5. Any word I/O instruction can be used to access a Page Descriptor register via this port, provided that the Page Descriptor Register Pointer is properly initialized.

Block Move I/O Port. Block moves of data into and out of the Page Descriptor registers are accomplished by writing and reading words to or from this dedicated I/O port at location FFxxF4. Any word I/O instruction can be used to access Page Descriptor registers via this port, provided that the Page Descriptor Register Pointer is properly initialized.

Invalidation I/O Port. Valid bits can be cleared (i.e., the Page Descriptor registers invalidated) by writing to this dedicated 8-bit port (Table 4), which is at I/O address location FFxxF2. Individual Valid bits can subsequently be set by software writing to the Page Descriptor registers. Reading this I/O port returns unpredictable data.

Table 4. Invalidation Port Table

Encoding	Registers Invalid
01_H	System Page Descriptor Registers 0-7
02_H	System Page Descriptor Registers 8-15
03_H	System Page Descriptor Registers 0-15
04_H	User Page Descriptor Registers 0-7
08_H	User Page Descriptor Registers 8-15
$0C_H$	User Page Descriptor Registers 0-15

Translation Mechanism

Address Translation. Address translation is illustrated in Figure 19. While the Program/Data Space Separation bit is cleared to 0, the 16-bit logical address is divided into two fields, a 4-bit index field used to select one of 16 Page Descriptor registers and a 12-bit offset field that forms the lower 12 bits of the physical address. The physical address is composed of the 12-bit page frame address (bits 4-15) supplied by the selected Page Descriptor register and the 12-bit offset supplied by the logical address.

While the Program/Data Space Separation bit is set to 1, the logical address is divided into a 3-bit index field and a 13-bit offset field. The Page Descriptor register consists of an 11-bit Page Frame Address field (bits 5-15, with bit 4 = 0). The physical address is a result of concatenating the page frame address and the logical offset. The Page Descriptor register is chosen by a 4-bit index field, which consists of a Program/Data Address bit from the CPU and the three Index bits from the logical address.

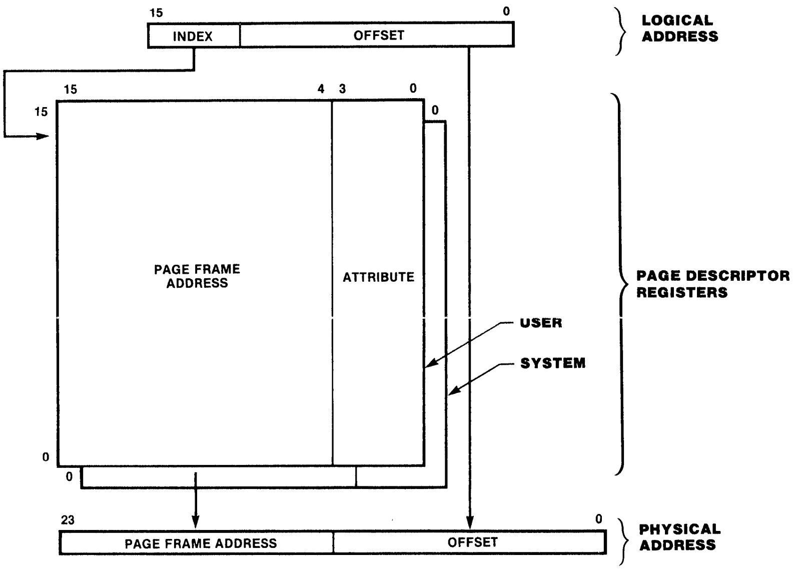


Figure 19. Address Translation

ON-CHIP MEMORY

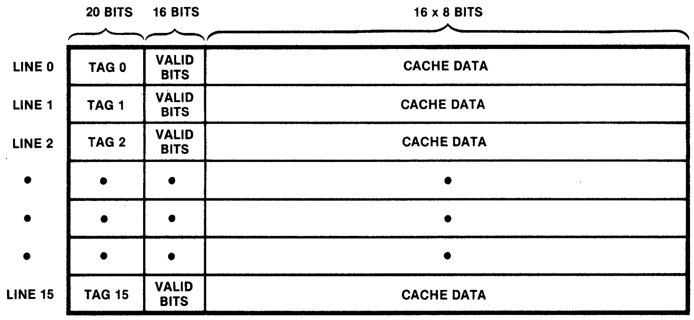
Features

- 256-byte local memory
- Configurable as high-speed associative cache
- Programmable to cache instructions, data, or both
- Permits faster execution by minimizing external bus accesses
- Operation is transparent to user
- Configurable as local RAM with user-definable addresses

The Z280 MPU has 256 bytes of on-chip memory, which can be dedicated to memory locations programmed by the system or used as a cache for instructions or data. Its mode of use (dedicated memory or cache) is programmable; on reset it is automatically enabled for use as a cache for instructions only.

On-Chip Memory Architecture

The on-chip memory is organized as 16 lines of 16 bytes each. Each line can hold a copy of 16 consecutive bytes in physical memory locations whose 20 most significant bits of physical address are identical. Each byte in the cache has an associated Valid bit that indicates whether the cache holds a valid copy of the memory contents at the associated physical memory location. Figure 20 illustrates the cache organization.



Tag n = the 20 Address bits associated with line n
Valid bits = 16 bits that indicate which bytes in the cache line contain valid data
Cache data = 16 bytes

Figure 20. Cache Organization

The on-chip memory has two modes of operation. If the Memory/Cache bit in the Cache Control register is set to 1, then the 256 bytes of on-chip memory are treated as physical memory locations. Memory accesses to addresses covered by the on-chip memory do not generate bus transactions on the external bus and hence the accesses are faster. In this mode, the Valid bits are ignored.

If the Memory/Cache bit is cleared to 0, then the 256 bytes of on-chip memory are treated as a cache memory. The lines are allocated using a least-recently used (LRU) algorithm. When a cache "miss" on a read occurs (and the MMU does not assert cache inhibit), the line in the cache that has been least recently accessed is selected to hold the newly read data. All bytes in the selected line are marked invalid except for the bytes containing the newly accessed data. On a cache miss, one or two bytes, depending on the bus size, are fetched from main memory. Except for burst mode instruction fetches, the cache does not pre-fetch beyond the currently-requested address. A cache miss on a data write does not cause a line to be allocated to the memory location accessed.

The cache can hold both instructions and data. Two control bits in the Cache Control register can be separately set to enable the cache to hold instructions and to hold data. If the cache contains data, writes to data at locations contained in the cache also cause external bus transactions to update the appropriate memory location.

Both the CPU and the on-chip DMAs access the cache. For the CPU, if the MMU is enabled, the access can be either cacheable or non-cacheable, depending on the value of the Cacheable bit in the Page Descriptor register used to translate the logical address. If the MMU is not enabled, all memory transactions are considered to be cacheable. Two bits in the Cache Control register, the Cache Instructions Disable bit and the Cache Data Disable bit, further determine the operation of the cache for various situations. These bits enable the cache for instructions and for data.

When the on-chip memory is used as fixed memory locations, neither the Cache Instruction Disable or Cache Data Disable bits are used, and no distinction is made as to whether the CPU is accessing data or instructions.

In general, when devices such as on-chip DMAs transfer data to the memory, the cache data is modified if the write is to a valid location in the cache but the LRU mechanism is unaffected. Also, for the EPU to memory transfer, if the cache contains valid locations that are updated by an EPU transaction, the on-chip cache is also updated.

Cache Control Register. The operation of the on-chip memory is controlled by an 8-bit Cache Control register (Figure 21) that is accessed using a load control instruction. This register contains five control bits.

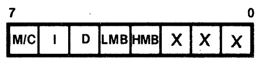


Figure 21. Cache Control Register

The bits in this register are:

High Memory Burst Capability (HMB). This 1-bit field specifies whether a memory burst transaction occurs when the MMU is enabled and there is a 1 in bit 15 of the selected Page Descriptor register (0 = burst mode not supported, 1 = burst mode supported).

Low Memory Burst Capability (LMB). This 1-bit field specifies whether a memory burst transaction occurs when the MMU is disabled or when the MMU is enabled and there is a 0 in bit 15 of the selected Page Descriptor register (0 = burst mode not supported, 1 = burst mode supported).

Cache Data Disable (D). While this bit is cleared to 0, data fetches are copied into the cache if the M/C bit = 0 (cache mode). If M/C = 1, the state of this bit is ignored.

Cache Instructions Disable (I). While this bit is cleared to 0, instruction fetches are copied into the cache when the M/C bit = 0 (cache mode). When M/C = 1, the state of this bit is ignored.

Memory/Cache (M/C). While this bit is set to 1, the on-chip memory is to be accessed as physical memory; while it is cleared to 0, the memory is accessed associatively as a cache.

If the on-chip memory is to be used as fixed memory locations, the user can programmably select the ranges of memory addresses for which the on-chip memory responds.

CLOCK OSCILLATOR

The Z280 MPU has an on-chip clock oscillator/generator that can be connected to a fundamental, parallel-resonant crystal or any suitable clock source. The bus timing clock generated from the on-chip oscillator is output for use by the rest of the system.

REFRESH

The Z280 MPU has an internal mechanism for refreshing dynamic memory. This mechanism can be activated by setting the Refresh Enable bit in the Refresh Rate register to 1. Memory refresh is performed periodically at a rate specified by the Refresh Rate register. Refresh transactions are identical to memory transactions except that different status signals are used and no data is transferred. They can be inserted immediately after the last clock cycle of any bus transaction, including an internal operation.

The refresh transaction is generated as soon as possible after the refresh period has elapsed (generally after the last clock cycle of the current bus transaction). If the MPU receives an interrupt request, the refresh operation is performed first. When the Z280 MPU does not have control of the bus or is in the Wait state, internal circuitry records the number of refresh periods that have elapsed and refresh cycles cannot be generated. When the MPU regains control of the bus or the $\overline{\text{WAIT}}$ input signal is deactivated and the bus transaction completes, the refresh mechanism immediately issues the skipped refresh cycles. The internal circuitry can record up to 256 such skipped refresh operations.

A 10-bit refresh address is generated for each refresh operation with the refresh address being incremented by two between refreshes for 16-bit data bus and by one for 8-bit data bus.

On reset, the Refresh Rate register contains 88_H, refresh is enabled, the rate is 32 processor clock cycles, and the refresh address is not affected.

The Refresh mechanism is controlled by an 8-bit control register, described below.

Refresh Rate Register

This 8-bit register (Figure 22) enables the refresh mechanism and specifies the frequency of refresh transactions.

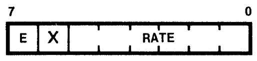


Figure 22. Refresh Rate Register

The fields in this register are:

Refresh (Rate). This field indicates in processor clock cycles the rate at which refresh transactions are to be generated; a value of n in this field indicates a refresh period of 4n, with Rate = 0 indicating 256 clock cycles.

Refresh Enable (E). When this 1-bit field is set to 1, the refresh mechanism is enabled.

UART

The Z280 UART transmits and receives serial data using any common asynchronous data-communication protocol.

Transmission and reception can be performed independently with five, six, seven, or eight bits per character, plus optional even or odd parity. The transmitter can supply one or two stop bits and can provide a break output at any time. Reception is protected from spikes by a "transient spike-rejection" mechanism that checks the signal one-half a bit time after a Low level is detected on the receiver data input; if the Low does not persist—as in the case of a transient—the character assembly process is not started. Framing errors and overruns are detected and buffered with the partial character on which they occur. Furthermore, a built-in checking process avoids interpreting a framing error as a new start bit: a framing error results in the addition of one-half a bit time to the point at which the search for the next start bit is begun.

The UART uses the same clock frequency for both the transmitter and the receiver. The input for the UART clocking circuitry is derived from counter/timer 1, either from its external input line for an external clock or from the counter/timer output for a bit rate generated from the internal processor clock. The UART input clock is further scaled by 1, 16, 32, or 64 for clocking the transmitter and receiver.

Two of the DMA channels can be used independently to move characters between memory and the transmitter or receiver without CPU intervention. Both the transmitter and receiver can interrupt the CPU for processor assistance.

The UART uses two external pins, Transmit and Receive. Data that is to be transmitted is placed serially on the Transmit pin and data that is to be received is read in from the Receive pin.

Asynchronous Transmission

The Transmitter Data Output line is held High (marking) when the transmitter has no data to send. Under program control, the Send Break command can be issued to hold the Data Output line Low (spacing) until the command is cleared.

The UART automatically adds the start bit, the programmed parity bit (odd, even, or no parity), and the programmed number of stop bits to the data character to be transmitted. When the character is five, six, or seven bits, the unused most significant bits in the Transmitter Data register are automatically ignored by the UART.

Serial data is shifted from the transmitter at a rate equal to 1, 1/16th, 1/32nd or 1/64th of the clock rate supplied to the transmitter clock input. Serial data is shifted out on the falling edge of the clock input.

Asynchronous Reception

An asynchronous receive operation begins when the Receive Enable bit in the Receiver Control/Status register is set to 1. A Low (spacing) condition on the Receive input line indicates a start bit. If this Low persists for at least one-half of a bit time, the start bit is assumed to be valid and the data input is then sampled at mid-bit time until the entire character is assembled. This method of detecting a start bit improves error rejection when noise spikes exist on an otherwise marking line. If the × 1 clock mode is selected, bit synchronization must be accomplished externally; received data is sampled on the rising edge of the clock.

Received characters are read from the Receive Data register. If parity is enabled, the parity bit is assembled as part of the character and is not removed from the assembled character for character lengths other than 8 bits. If the resulting character is still less than 8 bits, 1s are appended in the unused high-order bit positions.

Since the receiver is buffered by one 8-bit register in addition to the receiver shift register, the CPU has adequate time to service an interrupt and to accept the data character assembled by the UART. The receiver also has a buffer that stores error flags for each data character in the receive buffer. These error flags are loaded at the same time as the data character.

After a character is received, it is checked for the following error conditions:

- Parity Error: when the parity bit of the character does not match the programmed parity.
- Framing Error: if the character is assembled without any stop bits (i.e., a Low level is detected for a stop bit).
- Receiver Overrun Error: if the CPU fails to read a data character when more than one character has been received.

Since the Parity Error and Receiver Overrun Error flags are latched, the error status that is read reflects an error in the current character in the Receiver Data register plus any Parity or Overrun Errors detected since the last write to the Receiver Control/Status register. To keep correspondence between the state of the error buffers and the contents of the receiver data buffers, the Receiver Control/Status register must be read before the data.

Polled Operation

In a polled environment, the Receive Character Available bit in the Receiver Control/Status register must be monitored so the CPU can know when to read a character. This bit is automatically cleared when the Receiver Data register is read. To prevent overwriting data in polled operations, the transmitter buffer status must be checked before writing into the transmitter. The Transmit Buffer Empty bit in the Transmitter Control/Status register is set to 1 whenever the transmit buffer is empty.

UART Control and Status Registers

The UART operation is controlled by three control and status registers. The UART configuration register specifies the character size, parity, clock source, scaling, and loop-back enable. Both the transmitter and the receiver have their own control/status register.

UART Configuration Register. This 8-bit register (Figure 23) contains control information for both the transmitter and receiver.

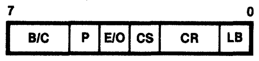


Figure 23. UART Configuration Register

The control fields for this register are:

Loopback Enable (LB). The UART is capable of local loopback. In this mode the internal transmit data line is tied to the internal receiver line and the external receiver input is ignored. If this bit is set to 1, loop back mode is enabled.

Clock Rate (CR). These two bits specify the multiplier between the clock and data rates (00 = data rate × 1, 01 = data rate × 16, 10 = data rate × 32, 11 = data rate × 64). The same rate is used for both the receiver and transmitter. If the × 1 clock rate is selected, bit synchronization must be accomplished externally.

Clock Select (CS). This bit specifies the clock input for the UART. If the bit is set to 1, the counter/timer 1 output pulse is used for bit-rate generation; if the bit is cleared to 0, the input line to counter/timer 1 provides the clock from an external source.

Parity Even/Odd (E/O). If parity is specified, this bit determines whether it is sent and checked as even or odd (1 = even).

Parity (P). If this bit is set to 1, an additional bit position (in addition to those specified in the bits/character control field) is added to transmitted data and is expected in received data. In the Receiver, the parity bit received is transferred to the CPU as a part of the character, unless eight bits/character is selected.

Bits/Character (B/C). Together, these two bits determine the number of bits to form a character. If these bits are changed during the time that a character is being assembled, the results are unpredictable (00 = 5 bits/character, 01 = 6 bits/character, 10 = 7 bits/character, 11 = 8 bits/character).

Transmitter Control/Status Register. This 8-bit register (Figure 24) specifies the operation of the transmitter.

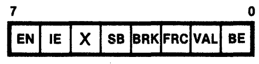


Figure 24. Transmitter Control/Status Register

The control bits for this register are:

Transmitter Buffer Empty (BE). This bit is automatically set to 1 whenever the transmitter buffer becomes empty and cleared to 0 when a character is loaded into the transmit buffer. This bit is in the set condition after a reset. This bit is controlled by the UART control circuitry; it can be read by an I/O read but cannot be set to 1 or cleared to 0 by an I/O write.

Value (VAL). This bit determines the value of the bits transmitted while the FRC bit is 1 and dummy characters are loaded into the transmitter buffer. When this bit is 1, a mark character (all 1s) is sent; when this bit is 0, a break character (all 0s) is sent.

Force Character (FRC). When this bit is set to 1, any character loaded into the transmitter buffer causes the transmitter output to be held High or Low (as indicated by the VAL bit) for the length of time required to transmit a character. This allows a program to generate a marking signal or a break of multiple-character duration simply by setting this bit to 1, setting the VAL bit to 1 or 0, and loading the appropriate number of dummy characters into the transmitter buffer.

Send Break (BRK). When set to 1, this bit immediately forces the transmitter output to the spacing condition, regardless of any data being transmitted. When this bit is cleared to 0, the transmitter returns to marking.

Stop Bits (SB). This bit determines the number of stop bits added to each asynchronous character sent. The receiver always checks for one stop bit. If this bit is set to 1, two stop bits are automatically appended to the character sent; if this bit is cleared to 0, only one stop bit is appended.

Transmitter Interrupt Enable (IE). When this bit is set to 1, interrupt requests are generated whenever the transmitter buffer becomes empty; when this bit is cleared to 0, no requests are made.

Transmitter Enable (EN). While this bit is cleared to 0, data is not transmitted and the transmitter output is held marking. Data characters in the process of being transmitted are completely sent if this bit is cleared to 0 after transmission has started.

Receiver Control/Status Register. This 8-bit register (Figure 25) specifies the operation of the receiver. The control bits are described below.

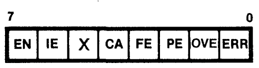


Figure 25. Receiver Control/Status Register

Receiver Error (ERR). This bit is the logical OR of the PE, OVE, and FE bits.

Framing Error (FE). This bit is automatically set to 1 for the received character in which the framing error occurred. Detection of a framing error adds an additional one-half of a bit time to the character time so the framing error is not interpreted as a new start bit.

Parity Error (PE). When parity is enabled, this bit is automatically set to 1 for those characters whose parity does not match the programmed sense (even/odd). This bit is latched, so once an error occurs, it remains set until it is cleared by software.

Receiver Overrun Error (OVE). This bit is automatically set to 1 to indicate that more than two characters have been received without a read from the CPU (or DMA). Only the most recently received character is flagged with this error, but when this character is read, the error condition is latched until cleared by software.

Receiver Character Available (CA). This bit is automatically set to 1 when at least one character is available in the receive buffer; it is automatically cleared to 0 when the Receiver Data register is read. This bit is controlled by the UART control circuitry; it can be read by an I/O read but cannot be set or cleared by an I/O write.

Receiver Interrupt Enable (IE). While this bit is set to 1, interrupt requests are generated whenever the receiver detects an error or the receiver has a character available.

Receiver Enable (EN). When this bit is set to 1, receiver operations begin. This bit should be set only after the parameters in the UART Configuration register are set.

UART Bootstrapping Option

The Z280 CPU supports an automatic initialization of memory via the UART after a reset operation. This system bootstrapping capability permits ROMless system configurations: the memory can be initialized by a serial link before the Z280 CPU fetches information from memory after the reset.

On the rising edge of Reset, the AD lines are sensed if $\overline{\text{WAIT}}$ is asserted; if AD_6 is being driven High, the Z280 CPU automatically enters a Halt state. The UART is also automatically initialized to receive 8-bit character data with odd parity at a ×16 clock rate. An external clock source is assumed. A minimum of 15 processor clock cycles must elapse before the transmission can begin.

During the bootstrapping operation, DMA Channel 0 is used to transfer received characters into the memory. This channel is initialized as follows:

***Transaction Descriptor register*-- IE, EPS, and TC cleared, ST- byte transfer, BRP-continuous, TYPE-flowthrough, DAD-Auto-increment memory address**

DMA Master Control register—DOR and EOP set

Count register—0100 (256 bytes to be transferred)

Destination Address register—000000 (starting memory address = 0)

Source Address register—undefined (not used when DMAO is linked to UART

Characters received are placed in memory starting at physical memory location zero. If an error occurs, the Z280 CPU drives the Transmitter Output line Low. External circuitry monitoring this line can use this signal to cause the transmitting device to begin the initialization procedure again, starting with a reset and AD_6 asserted on the rising edge of Reset.

After 256 bytes of data have been transferred, the Z280 CPU automatically begins execution by fetching the first instruction from memory location 0.

DMA CHANNELS

The Z280 MPU has four on-chip Direct Memory Access (DMA) channels to provide high bandwidth data transmission capabilities. There are two types of DMA channels; two support flyby transactions and the other two do not. The two types of DMA channels otherwise have identical capabilities, although they have different priorities with respect to interrupt requests and bus requests.

Each DMA channel is a powerful and versatile device for controlling and processing transfers of data. Its basic function of managing CPU-independent transfers between two ports is augmented by an array of features requiring little or no external logic in systems using an 8- or 16-bit data bus.

Transfers can be performed between any two ports (source and destination), including memory-to-I/O, I/O-to-memory, memory-to-memory, and I/O-to-I/O. Except for flyby, two port addresses are automatically generated for each transaction and can be either fixed or incrementing/decrementing.

During a transfer, a DMA channel assumes control of the system address and data bus. Data is read from one addressable port and written to the other addressable port, byte-by-byte or word-by-word. The ports can be programmed to be either system main memory or peripheral I/O devices.

For both flyby and flowthrough DMA transactions, if the destination is a memory location that corresponds to an entry in the on-chip memory (either cache or fixed memory location), the on-chip memory is updated to reflect the new contents of the memory location.

Except in flyby mode, two 24-bit addresses are generated by the DMA for every transfer operation, one address for the source port and another for the destination port. Two readable address counters (three bytes each) keep the current address of each port.

The DMA devices use the same memory and I/O timing as the CPU for bus transactions, as indicated by the appropriate bus timing register.

Modes of Transfer Operation

Each DMA can be programmed to operate in one of three transfer modes:

- *Single Transaction.* Data operations are performed one byte or word at a time.
- *Burst.* Data operations continue until a port's Ready line to the DMA goes inactive.
- *Continuous.* Data operations continue until either the end of the programmed block of data is reached or an end of process has been signaled before the system bus is released.

In all modes, once a byte or word of data is read by the DMA channel, the operation is completed in an orderly fashion, regardless of the state of other signals (including a port's Ready line).

Pin Descriptions

Each DMA channel has a Ready input line. In addition, two DMA channels have a flyby output line to support high speed data transfers between I/O devices and memory.

The flyby output is asserted by the DMA channel to signal a peripheral device associated with the DMA channel that it should participate in the data transmission during the current flyby bus transaction.

If Ready is active, the DMA channel requests control of the external system bus to perform the DMA transaction. When the external system bus is available for DMA transfers, the DMA channel with a request pending and the highest priority assumes bus mastership. The priority of DMA channels from highest to lowest is: DMA0, DMA1, DMA2, and DMA3. A DMA channel in burst mode relinquishes bus mastership to a higher priority DMA channel only when its Ready line is deasserted (or EOP is signaled or terminal count is reached). A DMA channel in continuous mode relinquishes bus mastership only when EOP is signaled or terminal count is reached.

Priority of On-Chip DMA Channels and External Bus Requesters

The on-chip DMA channels are arranged in a daisy chain with the external Bus Request input line being the "next lower bus requester" on this chain. The on-chip DMAs behave as if they were external bus requestors with respect to acquiring the bus, relinquishing the bus, and priority access to the bus.

End-of-Process

If the end-of-process (EOP) capability is enabled, transfers by DMA channels can be prematurely terminated by a Low on Interrupt A line or Interrupt B line during the transfer. This capability is programmed by control bits in the DMA Master Control register. EOP occurs regardless of the

setting of the Interrupt A Enable bit in the Master Status register. When an EOP is signaled, the EOP Signaled (EPS) bit in the Transaction Descriptor register of the active DMA channel is set to 1 and the Enable bit is cleared to 0. If interrupt requests are enabled (IE = 1 in the Transaction Descriptor register), an interrupt request is generated by the channel that was active when the EOP was signaled. After an EOP has been signaled, the DMA relinquishes the bus within 16 cycles of the last DMA bus transaction.

If the End-Of-Process signal on Interrupt A or B line is still asserted when the CPU is bus master, the signal is interpreted as an interrupt request; thus, both the DMA channel and the external EOP generating device can request interrupts simultaneously. Separate mask bits in the Master Status register enable the CPU to accept interrupts from these two sources.

On a flowthrough transaction, if the EOP signal is received while the information is being read into the Z280 MPU, the transfer is aborted and the data is not written out from the Z280 MPU.

DMA Linking

The DMA devices can be linked together to achieve DMA transfers to non-contiguous memory locations (linked operation). Bits in the DMA Master Control register allow DMA3 to be linked to DMA1 and DMA2 to be linked to DMA0. If the appropriate bit is set to 1 in the DMA Master Control register, the master DMA (0 or 1) signals its linked DMA each time its transfer is complete (count = 0). This acts as an internal ready input to the linked DMA that reloads the master DMA control registers.

Words are loaded into the master DMA control registers in the following order: Destination Address register (two words), Source Address register (two words), Count (one word), Transfer Descriptor register (one word). After six words have been transferred, the master DMA deasserts its internal ready line and begins the transfer of the next block **of data. The master DMA can be programmed to interrupt the CPU on "count equals 0" when the last block transfer is completed by the master DMA (to notify software that the entire sequence of transfers is completed).**

When programming linked DMAs, the last word to be programmed must be the master DMA's Transaction Descriptor register. Also, the linked DMA must be programmed before the master DMA's status register is programmed.

DMA Master Control Register. This 16-bit register (Figure 26) specifies the general configuration of the four on-chip DMA channels: the linking of the DMA channels, the software ready enables, and EOP enable.

X	X	X	X	EOPCSB		EOPCSA		EOPB	SRI	SRO	EOPA	D3L	D2L	D1T	DOR
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 26. DMA Master Control Register

The fields in this register are:

DMA0 to Receiver Link (D0R). When this bit is set to 1, DMA channel 0 is linked to the UART receiver.

DMA1 to Transmitter Link (D1T). When this bit is set to 1, DMA channel 1 is linked to the UART transmitter.

DMA2 Link (D2L). When this bit is set to 1, DMA channel 2 is linked to DMA channel 0.

DMA3 Link (D3L). When this bit is set to 1, DMA channel 3 is linked to DMA channel 1.

End-of-Process (EOP$_A$). **When this bit is set to 1, the INT$_A$ line is used as an end-of-process signal for the DMA channel defined by the EOPCSA field.**

End-of-Process (EOP$_B$). **When this bit is set to 1, the INT$_B$ input acts as an EOP input for the DMA channel defined by the EOPCSB field.**

Software Ready for DMA0 (SR0). When this bit is set to 1, DMA channel 0 requests service if enabled.

Software Ready for DMA1 (SR1). When this bit is set to 1, DMA channel 1 requests service if enabled.

End-of-Process Channel Select A (EOPCSA). **This field defines the DMA channel that has $\overline{\text{INT}}_A$ as its EOP input. This field has no effect if EOP$_A$ bit (bit 4) is cleared to zero**

00	DMA Channel 0
01	DMA Channel 1
02	DMA Channel 2
03	DMA Channel 3

End-of-Process Channel Select B (EOPCSB). **This field defines the DMA channel that has $\overline{\text{INT}}_B$ as its EOP input. This field has no effect if EOP$_B$ bit (bit 7) is cleared to zero.**

00	DMA Channel 0
01	DMA Channel 1
02	DMA Channel 2
03	DMA Channel 3

Note that while the EOP$_A$ and EOP$_B$ bits are active, $\overline{\text{INT}}_A$ and $\overline{\text{INT}}_B$ can still serve as interrupt inputs.

DMA Channel Control Registers

Transaction Descriptor Registers. These four 16-bit registers, one for each channel (Figure 27), describe the type of DMA transfer to be performed and contain control and status information.

15								0
EN	SAD	IE	ST	BRP	TYPE	TC	DAD	EPS

Figure 27. Transaction Descriptor Register

The fields in this register are:

End-of-Process Signaled (EPS). **This bit is set to 1 automatically when the channel is active and an end-of-process is signaled for this channel as programmed on the Interrupt A or Interrupt B input lines, thus prematurely terminating the transfer.**

Destination Address Descriptor (DAD). The setting of this 3-bit field indicates the type of location (memory or I/O) and how the address is to be manipulated (incremented, decremented or left unchanged), as shown in Table 5.

Table 5. SAD and DAD Encodings

Encoding	Address Modification Operation
000	Auto-increment memory location
001	Auto-decrement memory location
010	Memory address unmodified by transaction
011	Reserved
100	Auto-increment (by 1) I/O location
101	Auto-decrement (by 1) I/O location
110	I/O address unmodified by transaction
111	Reserved

Transfer Complete (TC). This bit is set to 1 automatically when the count register has reached zero.

Transaction Type (Type). This 2-bit field specifies flyby or flowthrough type of operation (00 = flowthrough, 01 = reserved, 10 = flyby write, 11 = flyby read). In flowthrough mode of operation, two bus transactions occur for each DMA operation—a read from the source followed by a write to the destination. In a flyby operation, only one bus transaction occurs for each DMA operation. In flyby write to memory, the flyby output pin is pulsed instead of an I/O transaction being performed and the contents of the Destination Address register are output to specify the memory location. In flyby read from memory, the flyby output pin is pulsed instead of an I/O transaction being performed and the contents of the Source Address register are output to specify the memory location. Only two DMAs have flyby capability.

Bus Request Protocol (BRP). The setting of these two bits indicates the mode of DMA operation (Table 6).

Table 6. Bus Request Protocol (BRP)

Encoding	DMA
0 0	Single Transaction
0 1	Burst
1 0	Continuous
1 1	Reserved

Size of Transfer (ST). This 2-bit field specifies the size of the entity to be transferred by the DMA channel (Table 7). For word transfers to or from memory locations, the memory address must be even (least significant bit is 0). Long word (32-bit) transfers are supported only in flyby mode, with the cache disabled.

Table 7. Size of Transaction (ST)

Encoding		Size of	Number to Increment/
ST1	ST0	Transfer	Decrement By
0	0	Byte	1
0	1	16-bit word	2
1	0	32-bit longword	4
1	1	Reserved	

Interrupt Enable (IE). When this bit is set to 1, the DMA generates an interrupt request at end of count or end of process. When this bit is 0, no interrupt request is generated.

Source Address Descriptor (SAD). The setting of this 3-bit field indicates the type of location (memory or I/O) and how the address is to be manipulated (incremented, decremented or left unchanged), as shown in Table 5.

DMA Enable (EN). While this bit is 1, the DMA transfer is enabled.

Count Register. This 16-bit register is programmed to contain the number of DMA transfers to be performed. When the contents of the count register reach zero, further requests on the RDY input line are ignored. The DMA channel can be programmed to generate an interrupt when the count register reaches zero.

Source Address Register and Destination Address Register. These 24-bit registers contain the 24-bit physical addresses to be used during the DMA transaction. They are not translated by the MMU. In flyby mode, only one of these registers is used to supply the address for the bus transaction as indicated in the Mode field in the Transfer Descriptor register. The format for these registers is shown in Figure 28.

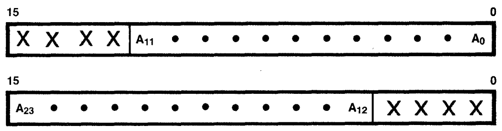


Figure 28. Source and Destination Address Registers Format

Flyby Transaction Timing

The Transaction Type field in the Transaction Descriptor register indicates whether the transaction is a read or a write. For flyby read transactions, the Source Address Descriptor indicates the transaction is a read from memory; for write flyby transactions the Destination Address Descriptor indicates the transaction is a write to memory. Additional wait states can be automatically inserted if programmed in the appropriate timing register. See Figures 29 and 30 for timing diagrams.

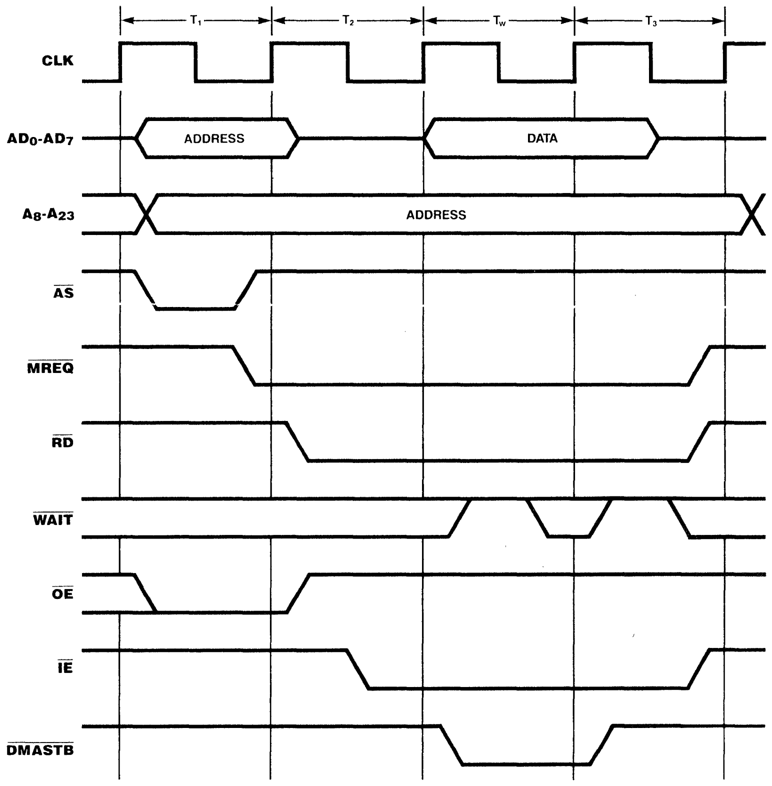


Figure 29a. On-Chip DMA Channel Flyby Memory Read Transaction, Z80 Bus

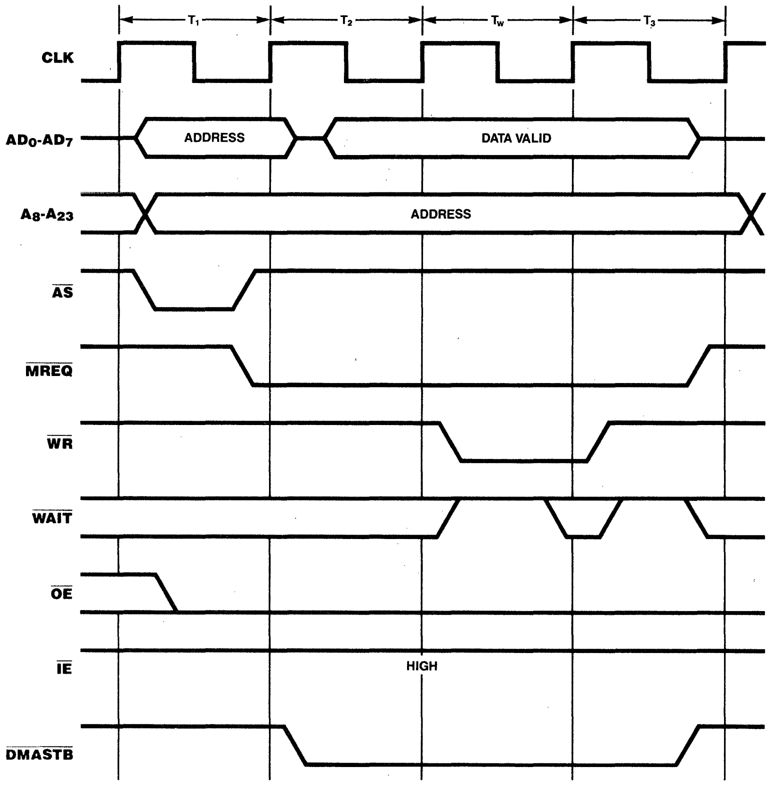


Figure 29b. On-Chip DMA Channel Flyby Memory Write Transaction, Z80 Bus

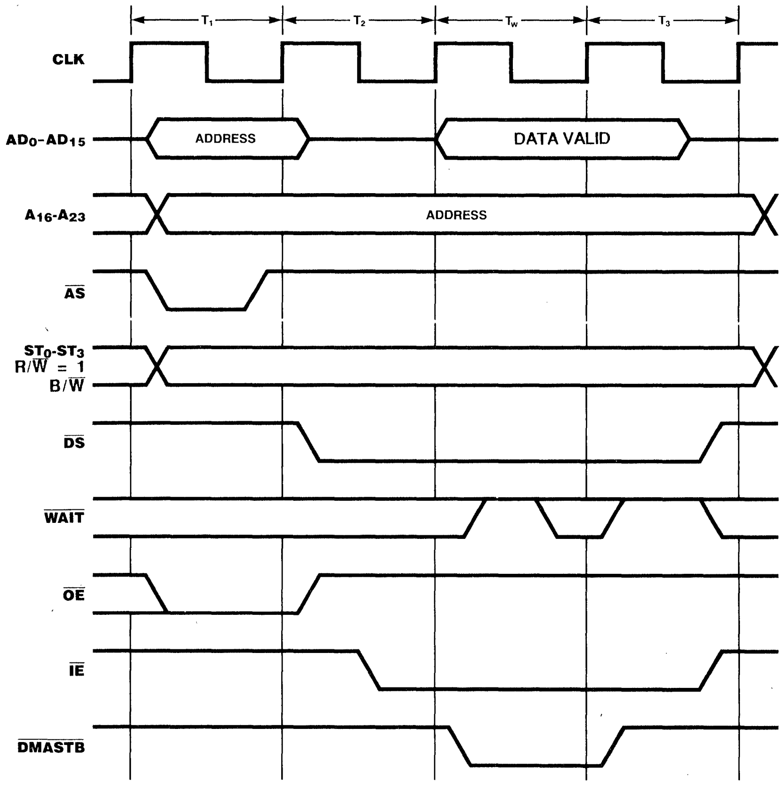


Figure 30a. On-Chip DMA Channel Flyby Memory Read Transaction, Z-BUS

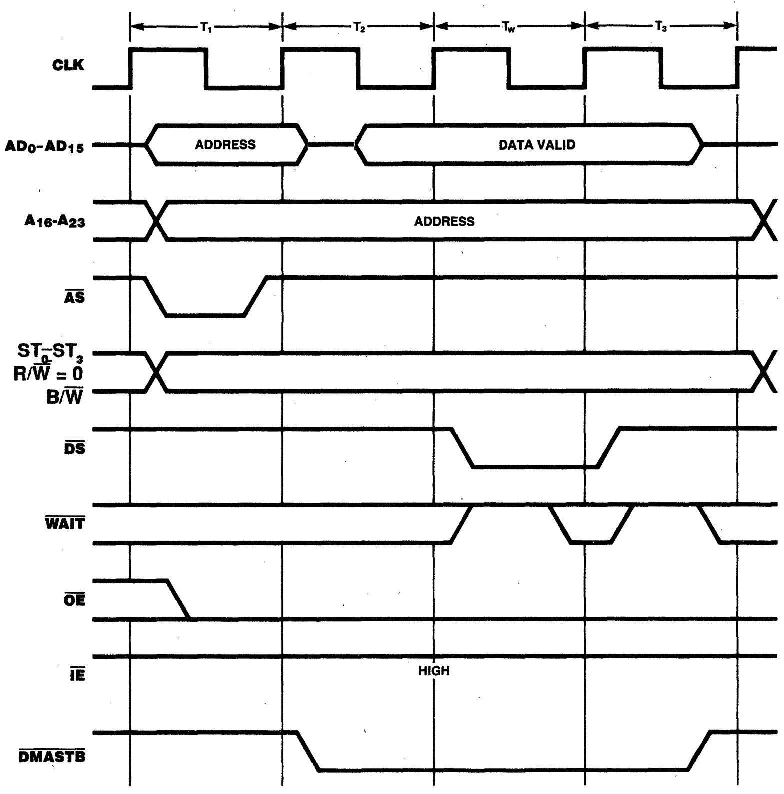


Figure 30b. On-Chip DMA Channel Flyby Memory Write Transaction, Z-BUS

COUNTER/TIMERS

The Z280 MPU's three counter/timers can be programmed by system software for a broad range of counting and timing applications. The three independently programmable channels satisfy common microcomputer system requirements for event counting, interrupt and interval timing, and general clock generation.

Programming the counter/timers is straightforward: each channel is programmed with four bytes. Once started, the channel counts down, and optionally reloads its time constant automatically and resumes counting. Software timing loops are completely eliminated. Interrupt processing is simplified because each channel uses a unique vector from the Interrupt/Trap Vector Table.

Each channel is individually programmed with three registers: a configuration byte, a control byte, and a time-constant word. The configuration byte selects the operating mode (counter or timer), enables or disables the channel interrupt, and selects certain other operating parameters. In the timing mode, the CPU processor clock is divided by four for input to the counter/timers. The time-constant word contains a value from 0 to 65,535.

During operation, the individual counter channel counts down from the present time-constant value. In counter mode operation, the counter decrements on each of the input pulses until the count/time output condition is met. Each decrement is synchronized by the scaled internal processor clock. For counts greater than 65,536, two of the counters can be programmably cascaded. When the count/time output condition is reached, the downcounter is automatically reset with the time constant value, if so programmed.

The timer mode determines time intervals without additional logic or software timing loops. Time intervals are generated by dividing the internal processor clock by four and decrementing a presettable downcounter. Thus, the time interval is an integral multiple of the processor clock period, the prescaler value four, and the time constant that is preset in the downcounter. A timer is triggered by setting the software trigger control bit in the Control/Status register or by an external input.

All three channels can generate an external output when the count/time output condition is met. The output is high when the internal presettable downcounter contains all zeros.

Each channel can be programmed to generate an Interrupt Request, which occurs only if the channel has its Interrupt Enable control bit set to 1 by software programming. When the Z280 CPU accepts the interrupt request it automatically vectors through the Interrupt Vector Table.

The three channels of the Z280 MPU are fully prioritized and fit into three different slots in the Z280 internal peripheral daisy-chain interrupt structure. Channel 0 has the highest priority and Channel 2 has the lowest. The channels have separate interrupt enables and the CPU's Master Status register has individual control bits that selectively inhibit interrupts from each channel.

Modes of Operation

The counter/timer channels have two basic modes of operation: as counters or as timers. As counters they monitor external input lines and record Low to High transitions on these lines. In the timer mode, the processor clock, scaled by four, is used instead of the external input line. The duration of this counting or timing can be either continuous from initial enabling (trigger operation) or only during intervals specified by signals on an input line (gate and gate/trigger operation). The count can be automatically restarted by programming the Retrigger Enable control bit in the counter/timer's Configuration register.

Each of the three counter/timers has a software gate and trigger facility that extends the hardware capabilities of the counter/timers.

Counting Operation. While the appropriate enabling conditions are met, the counter/timer monitors its input line for Low-to-High transitions. When such a transition occurs, the Count/Time register is decremented by 1.

Timing Operation. While the appropriate enabling conditions are met, the counter/timer monitors the internal processor clock scaled by four for Low-to-High transitions. When such a transition occurs the Count/Time register is decremented by 1.

Gate Operation. A counter/timer can be programmed to count or time only when a gating condition is met. While the counter/timer is enabled and the external gate capability is selected, an external input line is monitored; only while this line is High are the counting or timing operations performed. The software gate facility filters the state of the input line; while the software gate bit in the Command and Status register is cleared to 0, the gating condition is not met regardless of the signals on the gating line. The gate facility is illustrated in Figure 31.

Trigger Operation. A counter/timer can be programmed to count or time only after a triggering condition occurs. While the counter/timer is enabled and the external trigger capability is programmed, an external input line is monitored; only after this line makes a Low-to-High transition is a counting or timing operation performed. The software trigger facility causes the triggering condition to be met regardless of the activity of this line. The trigger operation is illustrated in Figure 32.

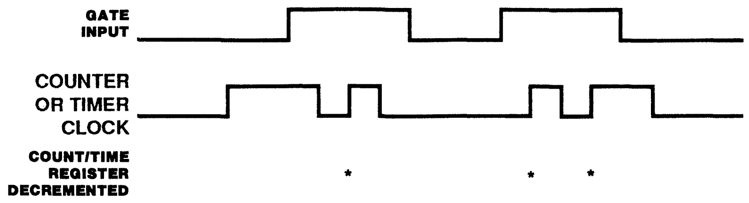


Figure 31. Gate Facility

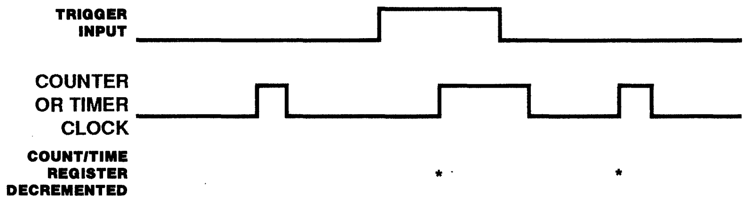


Figure 32. Trigger Operation

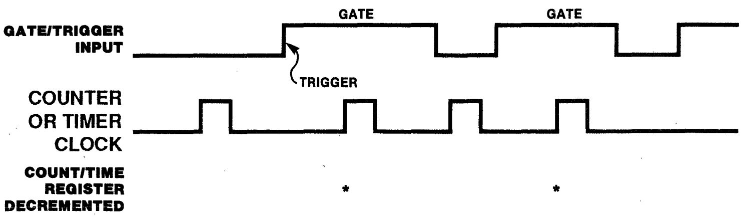


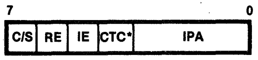
Figure 33. Gate/Trigger Operation

Gate/Trigger Operation. One input line can be used for both the gating and the triggering functions. A Low-to-High transition on this line acts as a trigger and subsequent High signals on this line function as gate signals. If non-retriggerable mode is programmed, subsequent Low-to-High transactions do not cause a trigger. Gate/Trigger Operation is shown in Figure 33.

The software gate and trigger mechanism can also be used in this mode of operation. A software gate before a trigger (hardware or software) has no effect on the counter/timer. After a hardware or software trigger, the software gate must be set to 1 for the Count/Time register to be decremented. A software trigger after a hardware or software trigger has no effect unless the Retrigger Enable control bit is set to 1.

Counter/Timer Control and Status Registers

Each counter/timer has two 8-bit control registers and two 16-bit count registers. The Configuration register and Command/Status register determine the counter/timers' operation, the Counter/Timer Command/Status register provides information about the current operation, the Time Constant register contains the initialization value for the counter/timer, and the Count/Time register contains the current value of the count in progress.



* Only the CTC bit in Counter/Timer 0 is used.

Figure 34. Counter/Timer Configuration Register

Counter/Timer Configuration Register. This 8-bit register (Figure 34) specifies the counter/timer's mode of operation: the pin configuration, whether an interrupt request is generated, and whether the countdown sequence is automatically restarted when the count reaches zero or when a trigger occurs.

The fields in this register are:

Input Pin Assignments (IPA). This 4-bit field specifies the functionality of the input lines associated with the counter/timer and whether the counter/timer monitors an external input (counting operation) or uses the scaled internal processor clock (timing operation). The four bits in this field can be associated with enabling output generation (EO), selecting the external signal or internal clock (C/T), enabling the gating facility (G), and enabling the triggering facility (T). The selected options determine the functions associated with each input line associated with the counter/timer, as illustrated in Table 8.

Table 8. Input Pin Functionality

IPA Field				Pin Functionality		
E0	C/T	G	T	Counter/Timer I/O	Counter/Timer Input	Notes
0	0	0	0	Unused	Unused	Timer
0	0	0	1	Unused	Trigger	Timer
0	0	1	0	Gate	Unused	Timer
0	0	1	1	Gate	Trigger	Timer
0	1	0	0	Unused	Input	Counter
0	1	0	1	Trigger	Input	Counter
0	1	1	0	Gate	Input	Counter
0	1	1	1	Gate/Trigger	Input	Counter
1	0	0	0	Output	Unused	Timer
1	0	0	1	Output	Trigger	Timer
1	0	1	0	Output	Gate	Timer
1	0	1	1	Output	Gate/Trigger	Timer
1	1	0	0	Output	Input	Counter
1	1	0	1	Unused	Unused	Reserved
1	1	1	0	Unused	Unused	Reserved
1	1	1	1	Unused	Unused	Reserved

Counter/Timer Cascade (CTC). When this bit is set to 1, counter/timers 0 and 1 form a 32-bit counter. When used as a 32-bit counter/timer, the fields in the Configuration register and Command/Status register for Counter/Timer 0 are ignored with the exception of the IE, CTC, EO, CIP, CC, and COR fields. The CTC bits in the Counter/Timer Configuration registers of counter/timers 1 and 2 are never used.

Interrupt Enable (IE). While this bit is set to 1, the counter/timer generates an interrupt request when the count/time output condition is met. While this bit is 0, no interrupt request is generated.

Retrigger Enable (RE). While this bit is set to 1, the time constant value is automatically loaded into the Count/Time register when a trigger input is received while the counter/timer is counting down. While this bit is 0, no reloading occurs.

Continuous/Single Cycle (C/S). While this bit is set to 1, the countdown sequence is automatically restarted when the count reaches zero by loading the time constant value into the Count/Time register. While this bit is 0, no reloading occurs.

Counter/Timer Command/Status Register. This 8-bit register (Figure 35) provides software control over the operation of the counter/timer and reflects the current status of the counter/timer's operation. Control bits in this register enable the counter/timer's operation and provide software gate and trigger capabilities. Status bits indicate whether a count is in progress, the count/time output condition has been reached, or the condition has been reached a second time.

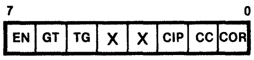


Figure 35. Counter/Timer Command/Status Register

The fields of this register are:

Count Overrun (COR). When this bit is set to 1, the count/time output condition has been reached and the CC bit is set to 1, thus indicating a count overrun condition. While this bit is cleared to 0, the count/time output condition has not been reached with the CC bit set since the time the CC bit was cleared by software. This bit can be read or written (set or cleared) by software I/O instructions.

Count/Time Output Condition has been Met (CC). When this bit is set to 1, the Count/Time register has been decremented to zero by the counter/timer control circuitry in single cycle mode or the Count/Time register has been reloaded in continuous mode. When this bit is cleared to 0, the count has not reached the count/time output condition since the bit was cleared by software. This bit can be read or written (set or cleared) by software I/O instructions.

Count In Progress (CIP). While this bit is set to 1, the counter/timer is operating and the Count/Time register is non-zero; while this bit is cleared to 0, the counter/timer is not operating. This bit is controlled by the counter/timer control circuitry; it can be read by an I/O read but cannot be set or cleared by an I/O write instruction.

Software Trigger (TG). When this bit is set to 1 (and the trigger operation of the counter/timer is enabled), if the Enable bit is also set to 1, the trigger operation is enabled on the rising edge of the first processor clock period following the setting of this bit from a previously cleared value. That is, if a hardware trigger has not already occurred, the contents of the Time Constant register are loaded into the Count/Time register and the countdown sequence begins. If a hardware trigger has already occurred, then if Retrigger Enable is set to 1, the counter/timer is retriggered; otherwise, setting this bit has no effect. Writing a 1 in this field when the previous value was 1 has no effect on the operation of the counter/timer. When this bit is cleared to 0, this bit has no effect on the operation of the counter/timer.

Software Gate (GT). When this bit is set to 1 (and the gate operation of the counter/timer is enabled), if the Enable bit is also set to 1, operation begins on the rising edge of the first processor clock period following the setting of this bit from a previously cleared value. Writing a 1 in this field when the previous value was 1 has no effect on the operation of the counter/timer. When this bit is cleared to 0, the countdown sequence is halted.

Enable (EN). While this bit is set to 1, the counter/timer is enabled; operation begins on the rising edge of the first processor clock period following the setting of this bit from a previously cleared value. Reset clears this bit. While this bit is cleared to 0, the value in the Time Constant register is constantly transferred to the Count/Time register. If the Time Constant register is all zeros, the output of the counter/timer is one. Thus, when the counter/timer is not enabled, the counter/timer output in conjunction with the Time Constant register can be used as an I/O port. Writing a 1 in this field when the previous value was 1 has no effect on the operation of the counter/timer. While this bit is 0, the counter/timer performs no operation during the next (and subsequent) processor clock periods.

Time Constant Register. This 16-bit register holds the value that is automatically loaded into the Count/Time register when the counter/timer is enabled or in the continuous or retrigger mode when the count reaches zero or the trigger is asserted, respectively. This register can be read or written by I/O instructions.

Count/Time Register. This 16-bit register holds the current value of the count or timing in progress. It is automatically loaded from the Time Constant register, and can be read by software using the I/O read instructions.

Pin Descriptions

The counter/timers have two external input lines associated with them. The I/O lines transfer signals between the counter/timers and external devices. The input lines receive signals from external devices for the counter/timers. The interpretations of the signals on these lines is determined by the Input Pin Assignment field in the Configuration register.

MULTIPROCESSOR MODE OF OPERATION

Features

- Allows global memory areas for shared resources
- Global memory addresses are user-specified
- Separate requests for local and global buses
- Requesting mechanism is transparent to user
- Easily interfaces to external arbiters

The Z280 supports various multiprocessor configurations, wherein it is the default bus master of the local bus, and it goes through a defined protocol to access the global bus. To invoke the multiprocessor mode, the Local Address Register contents should be defined, and the MP bit of the Bus Timing and Initialization Register set.

Pin Functionality When the Z280 is in the multiprocessor mode, Counter/Timer 0's IO pin is used as the Global Request ($\overline{\text{GREQ}}$) output, and Counter/Timer 0's Input pin is used as the Global Acknowledge ($\overline{\text{GACK}}$) input.

Local Address Register. Before an external memory bus transaction is to proceed, the Z280 distinguishes whether a bus transaction uses the local or global bus by comparing the four most significant bit of the physical address (address bits 20 through 23) with a 4-bit Base field in the Local Address register (Figure 36). A mask field in this register specifies which bits are to be compared. If all corresponding address bits match the Base field bits (for those bit positions specified by the mask field), then bus transaction can proceed on the local bus without requesting the global bus; if there is a mismatch in at least one specifies bit position, then the global bus is requested and the bus transaction does not proceed until the global bus acknowledge signal is asserted.

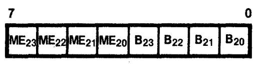


Figure 36. Local Address Register

The bits in the Local Address register are:

Base (B_n). When B_n is 1, address bit A_n must be 1 for a local bus transaction to be performed (unless Match Enable bit ME_n is 0); when bit B_n is 0, address bit A_n must be 0 for a local bus transaction to be performed.

Match Enable (ME_n). When ME_n is 1, address bit A_n is compared to base bit B_n to determine if the address requires the use of the global bus. When ME_n is 0, then any values for A_n and B_n will produce a match. If each ME_n is 0, then all bus transactions are performed on the local bus.

CPU Accesses on the Global Bus

The Z280 is the default local bus master, whether it is in the multi-processor mode or not. It relinquishes the local bus by following a protocol controlled by the $\overline{\text{BUSREQ}}$ input and $\overline{\text{BUSACK}}$ output pins. When $\overline{\text{BUSREQ}}$ is asserted, it is synchronized internally by the CPU. When the CPU is ready to relinquish the local bus, it places all its bus control outputs, including $\overline{\text{GREQ}}$, in 3-state, and then drives $\overline{\text{BUSACK}}$ active. After reset, the CPU acknowledges a request for the local bus before performing any transactions.

In multi-processor mode, the CPU determines if the next external memory transaction should access the global bus. If such is the case, and if the CPU currently is the local bus master, it puts the global address on the address outputs, and the status signals are also made valid, at the beginning of a bus clock cycle. $\overline{\text{GREQ}}$ is asserted in the second half of the same bus clock cycle. The CPU then samples $\overline{\text{BUSREQ}}$ and $\overline{\text{GACK}}$ continuously. Both inputs are synchronized internally by the CPU. The CPU will proceed with the global transaction after it samples that $\overline{\text{GACK}}$ is asserted, with the absence of $\overline{\text{BUSREQ}}$. Once the CPU controls the global bus, it can perform multiple global transactions. It relinquishes the global bus when the next transaction should not be global, when $\overline{\text{BUSREQ}}$ becomes active, or when $\overline{\text{GACK}}$ is de-asserted. A global test and set instruction is atomic (global read is followed by global write), and a global memory burst transaction completes its entire sequence of data transfers.

DMA Accesses on the Global Bus

Each on-chip DMA channel can access the global bus to perform data transfers. The address generated during each DMA-initiated memory transfer is compared with the contents of the Local Address register to determine whether the global bus should be requested. The protocol is identical to the global memory transactions initiated by the CPU.

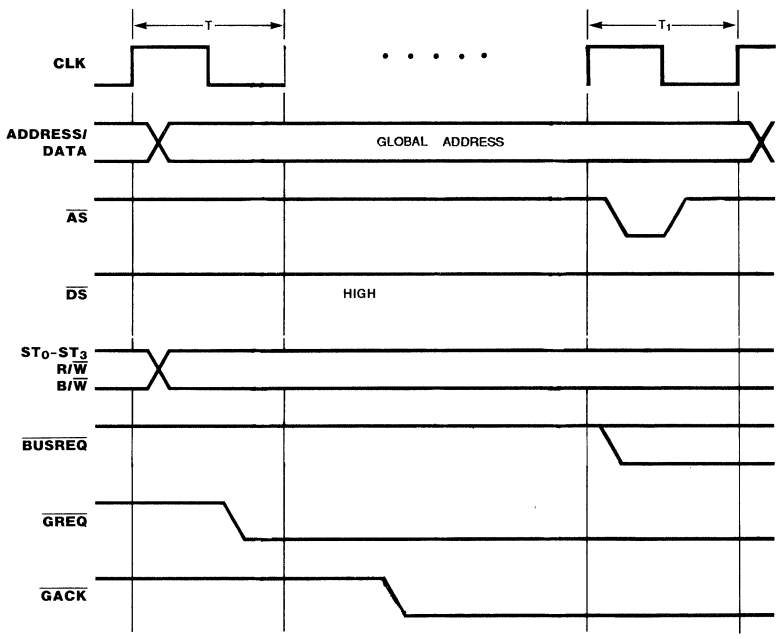


Figure 37. Multiprocessor Mode Timing, Z-Bus Example

EXTERNAL INTERFACE

The two different external interfaces for the Z280 MPU are the 8-bit Z80 Bus and the 16-bit Z-BUS.

Z80 Bus External Interface

Features

- 8-bit data bus
- Multiplexed address/data lines
- Supports Z80 Family peripherals

Pin Descriptions

A_8-A_{23}. *Address* (output, active High, 3-state). These address lines carry I/O addresses and memory addresses during bus transactions.

AD_0-AD_7. *Address/Data* (bidirectional, active High, 3-state). These eight multiplexed Data and Address lines carry I/O addresses, memory addresses, and data during bus transactions.

$\overline{AS}$. *Address Strobe* (output, active Low, 3-state). The rising edge of $\overline{AS}$ indicates the beginning of a transaction and shows that the address is valid.

$\overline{BUSACK}$. *Bus Acknowledge* (output, active Low). A Low on this line indicates that the CPU has relinquished control of the bus in response to a bus request.

$\overline{BUSREQ}$. *Bus Request* (input, active Low). A Low on this line indicates that an external bus requester has obtained or is trying to obtain control of the bus.

CLK. *Clock Output* (output). The frequency of the processor timing clock is derived from the oscillator input (external oscillator) or crystal frequency (internal oscillator). The processor clock is further divided by one, two, or four (as programmed) and then output on this line.

CTIN. *Counter/Timer Input* (input, active High). These lines receive signals from external devices for the counter/timers.

CTIO. *Counter/Timer I/O* (bidirectional, active High, 3-state). These I/O lines transfer signals between the counter/timers and external devices.

$\overline{DMASTB}$. *DMA Flyby Strobe* (output, active Low). These lines select peripheral devices for flyby transfers.

$\overline{EOP}_A$, $\overline{EOP}_B$. *End of Process* (input, active Low). An external source can terminate a DMA operation in progress by driving $\overline{EOP}_A$ or $\overline{EOP}_B$ Low. $\overline{EOP}$ always applies to the corresponding programmed channel; if no channel is active, $\overline{EOP}$ is ignored.

$\overline{GACK}$. *Global Acknowledge* (input, active Low). A Low on this line indicates the CPU has been granted control of a global bus.

$\overline{\text{GREQ}}$. *Global Request* (output, active Low, 3-state). A Low on this line indicates the CPU has obtained or is trying to obtain control of a global bus.

GND. *Ground.* Ground reference.

$\overline{\text{HALT}}$. *Halt* (output, active Low, 3-state). This signal indicates that the CPU is in the Halt state and is awaiting an interrupt before operation can resume.

$\overline{\text{IE}}$. *Input Enable* (output, active Low, 3-state). A Low on this line indicates that the direction of transfer on the Address/Data lines is toward the MPU.

$\overline{\text{INT}}$. *Maskable Interrupts* (input, active Low). A Low on these lines requests an interrupt.

$\overline{\text{IORQ}}$. *Input/Output Request* (output, active Low, 3-state). This signal indicates that AD_0-AD_7 and A_{16}-A_{23} of the address bus hold a valid I/O address for an I/O read or write operation. An $\overline{\text{IORQ}}$ signal is also generated with an $\overline{\text{M1}}$ signal when an interrupt is being acknowledged, to indicate that an interrupt response vector can be placed on the data bus.

$\overline{\text{M1}}$. *Machine Cycle One* (output, active Low, 3-state). This signal indicates that the current transaction is the opcode fetch cycle of a RETI instruction execution. $\overline{\text{M1}}$ also occurs with $\overline{\text{IORQ}}$ to indicate an interrupt acknowledge cycle.

$\overline{\text{MREQ}}$. *Memory Request* (output, active Low, 3-state). This signal indicates that the address bus holds a valid address for a memory read or write operation.

$\overline{\text{NMI}}$. *Nonmaskable Interrupt* (input, falling-edge activated). A High-to-Low transition on this line requests a nonmaskable interrupt.

$\overline{\text{OE}}$. *Output Enable* (output, active Low, 3-state). A Low on this line indicates that the direction of transfer on the Address/Data lines is away from the MPU.

OPT. *Bus Option* (input). This signal establishes the bus option during reset.

OPT	Bus Interface
0	Z80 Bus, 8-bit
1	Z-BUS, 16-bit

$\overline{\text{PAUSE}}$. *MPU Pause* (input, active Low). While this line is Low the MPU refrains from transferring data to or from an Extended Processing Unit in the system or from beginning the execution of an instruction.

$\overline{\text{RD}}$. *Read* (output, active Low, 3-state). This signal indicates that the CPU or DMA peripheral is reading data from memory or an I/O device.

$\overline{\text{RDY}}$. *DMA Ready* (input, active Low). These lines are monitored by the DMAs to determine when a peripheral device associated with a DMA port is ready for a read or write operation. When a DMA port is enabled to operate, its Ready line indirectly controls DMA activity; the manner in which DMA activity is controlled by the line varies with the operating mode (single-transaction, burst, or continuous).

$\overline{\text{RESET}}$. *Reset* (input, active Low). A Low on this line resets the CPU and on-chip peripherals.

$\overline{\text{RFSH}}$. *Refresh* (output, active Low, 3-state). This signal indicates that the lower ten bits of the Address bus contain a refresh address for dynamic memories and the current $\overline{\text{MREQ}}$ signal should be used to perform a refresh to all dynamic memories.

RxD. *UART Receive* (input, active High). This line receives serial data at standard TTL levels.

TxD. *UART Transmit* (output, active High). This line transmits serial data at standard TTL levels.

$\overline{\text{WAIT}}$. *Wait* (input, active Low). A Low on this line indicates that the responding device needs more time to complete a transaction.

$\overline{\text{WR}}$. *Write* (output, active Low, 3-state). This signal indicates that the bus holds valid data to be stored at the addressed memory or I/O location.

XTALI. *Clock/Crystal Input* (time-base input). Connects a parallel-resonant crystal or an external single-phase clock to the on-chip oscillator.

XTALO. *Crystal Output* (time-base output). Connects a parallel-resonant crystal to the on-chip oscillator.

+5V. *Power Supply Voltage.* (+5 nominal).

Bus Operations

Two kinds of operations can occur on the system bus: transactions and requests. At any given time, one device (either the CPU or a bus requester) has control of the bus and is known as the bus master. A transaction is initiated by the bus master and is responded to by some other device on the bus. Only one transaction can proceed at a time; seven kinds of transactions can occur:

DMA Flyby. This transaction is used by the DMA peripheral to transfer data between an external peripheral and memory.

Halt. This transaction is used to indicate that the CPU is entering the Halt state.

Interrupt Acknowledge. This transaction is used by the CPU to acknowledge an interrupt and to transfer additional information from the interrupting device.

I/O. This transaction is used by the CPU or DMA peripheral to transfer data to or from an external peripheral.

Memory. This transaction is used by the CPU or DMA peripheral to transfer data to or from a memory location.

Refresh. This type of transaction performed by the refresh peripheral does not transfer data; it refreshes dynamic memory.

RETI. This transaction is generated only by the CPU and is used in conjunction with the Z8400 peripheral's interrupt logic.

Only the bus master can initiate transactions. A request, however, can be initiated by a component that does not have control of the bus. Two types of these requests can occur:

Bus. This request is used by external devices to request control of the system bus to initiate transactions.

Interrupt. This request is used to request the attention of the CPU.

When an interrupt or bus request is made, it is answered by the CPU according to its type. For an interrupt request, the CPU initiates an interrupt acknowledge transaction and for bus requests, the CPU enters bus disconnect state, relinquishes the bus, and activates an Acknowledge signal.

Finally, the Z280 MPU itself may not be the system bus master. See the Multiprocessor Mode section for a discussion of this capability.

Transactions

Information transfers (both instructions and data) to and from the Z280 MPU are accomplished through the use of transactions. All transactions start when $\overline{\text{AS}}$ is driven Low and then raised High. This signal can be used to latch Z280 MPU addresses to de-multiplex the Z280 Address/Data lines required by Z80 Family peripherals. Coincident with $\overline{\text{AS}}$ assertion, the Output Enable line is also asserted.

If the transaction requires an address, it is valid on the rising edge of $\overline{\text{AS}}$. No address is required for Interrupt Acknowledge transactions.

The Read and Write lines are used to time the actual data transfer. (Refresh transactions do not transfer any data and thus do not activate $\overline{\text{RD}}$.) For write operations, a Low on $\overline{\text{WR}}$ indicates that valid data from the bus master is on the AD lines. The Output Enable line is also activated with $\overline{\text{WR}}$. For read operations, the bus master makes the AD lines 3-state before driving $\overline{\text{RD}}$ Low so that the addressed device can put its data on the bus. The bus master samples this data on the falling clock edge just before raising $\overline{\text{RD}}$ High. The Input Enable line is also activated with $\overline{\text{RD}}$.

Wait Cycle. The $\overline{\text{WAIT}}$ line is sampled on the falling clock edge when data is to be sampled (i.e., when $\overline{\text{RD}}$ or $\overline{\text{WR}}$ rises). If the $\overline{\text{WAIT}}$ line is Low, another cycle is added to the transaction before data is sampled ($\overline{\text{RD}}$ or $\overline{\text{WR}}$ rises). In this added cycle and all subsequent cycles added due to $\overline{\text{WAIT}}$ being Low, the $\overline{\text{WAIT}}$ line is sampled on the falling edge and, if it is Low, another cycle is added to the transaction. In this way, the transaction can be extended by external devices to an arbitrary length to accommodate (for example) slow memories or I/O devices that are not yet ready for data transfer.

The $\overline{\text{WAIT}}$ input is synchronous and thus must meet the specified setup and hold times in order for the Z280 MPU to function correctly. This requires asynchronously generated $\overline{\text{WAIT}}$ signals to be synchronized to the CLK output before they are input into the Z280 MPU. Automatic wait states can also be generated by programming the Bus Timing and Control register and the Bus Timing and Initialization register; these are inserted in the transaction before the external $\overline{\text{WAIT}}$ signal is sampled.

Memory Transactions. Memory transactions move instructions or data to or from memory when the Z280 MPU makes a memory access. Thus, they are generated during program execution to fetch instructions from memory and to fetch and store memory data. They are also generated to store old program status and fetch new program status during interrupt and trap handling, and are used by DMA peripherals to transfer information. A memory transaction is three bus cycles long unless extended with wait states (Figures 38 and 39).

RETI Transactions. These transactions (Figure 40) are similar to two memory read transactions except that $\overline{\text{M1}}$ is asserted throughout each read transaction, falling early in the first bus cycle, and that $\overline{\text{MREQ}}$, $\overline{\text{M1}}$, $\overline{\text{RD}}$ and $\overline{\text{IE}}$ are deasserted on the rising edge of the clock following the third cycle. Each of the read transactions is followed by a minimum of three bus cycles of inactivity. These transactions are invoked when an RETI instruction is encountered in the instruction stream; they are used during the re-fetching of the instruction from memory so that interrupt logic within Z80 peripherals that monitor the bus for this instruction will function correctly.

Note: Refresh cycles and DMA transfers may occur between RETI bus cycles.

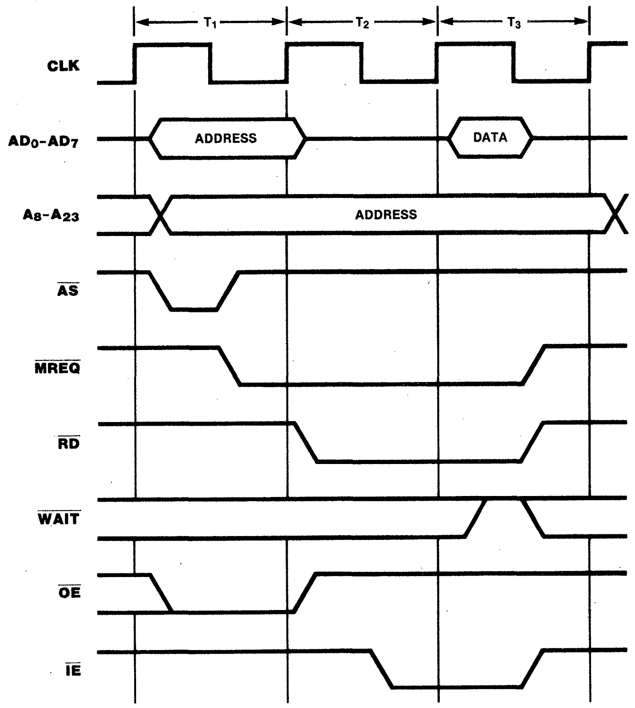


Figure 38. Memory Read Timing

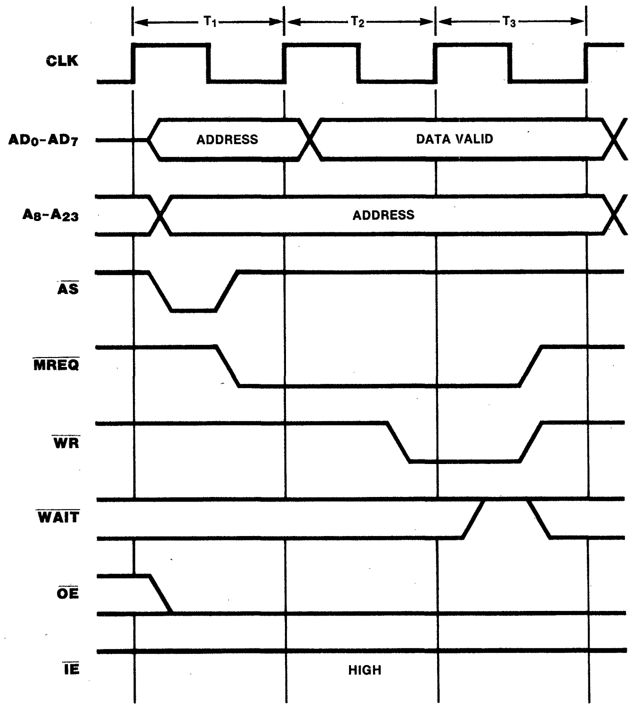


Figure 39. Memory Write Timing

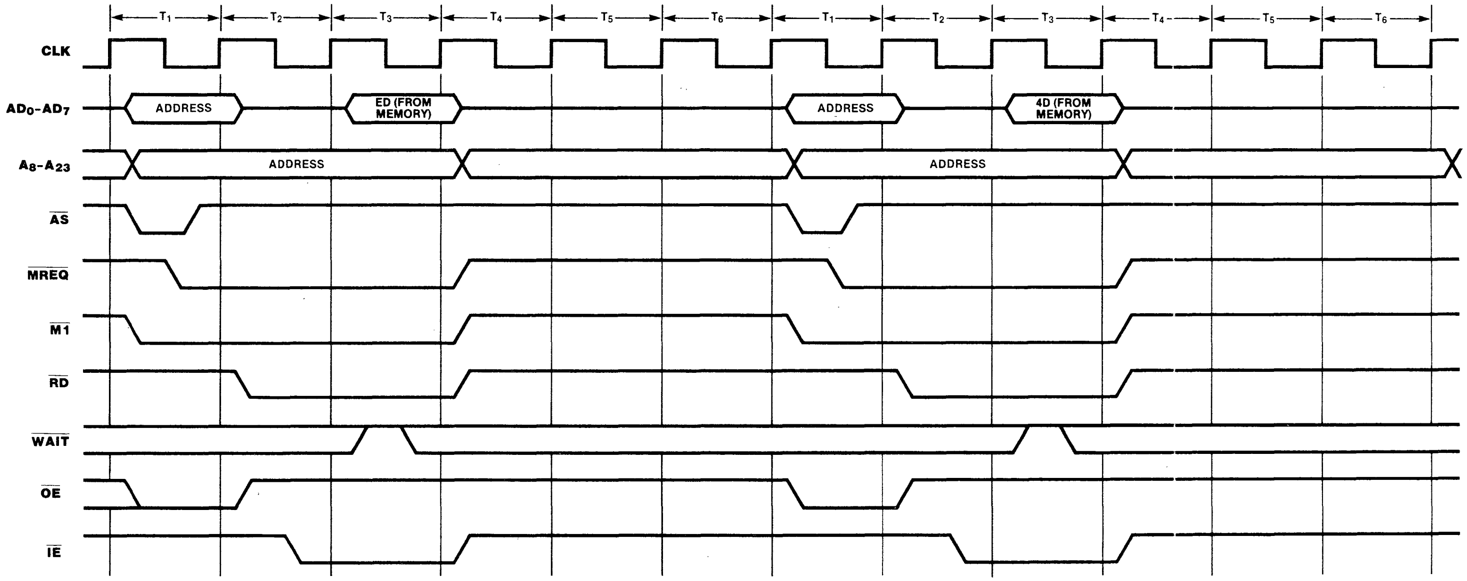


Figure 40. RETI Read Timing

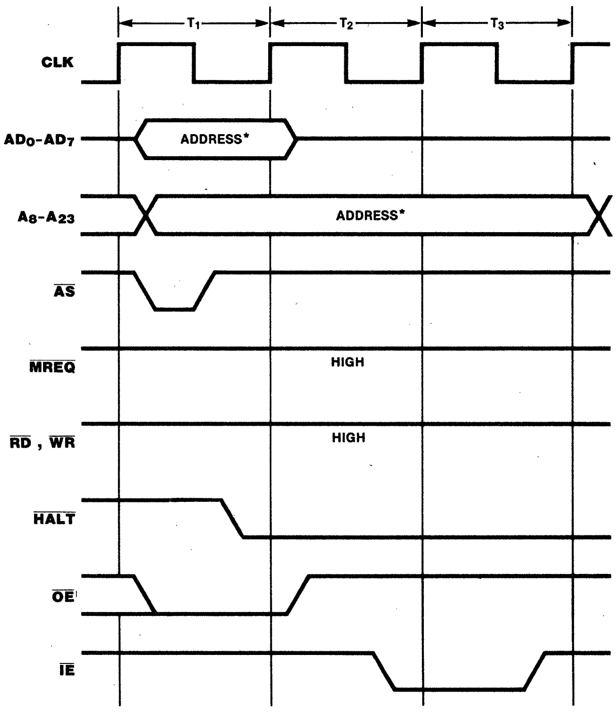
Halt Transactions. The Halt bus transaction does not transfer data (Figure 41). It looks like a memory transaction, except that $\overline{RD}$ and $\overline{WR}$ remain High and no data is transferred. The $\overline{WAIT}$ line is not sampled during the Halt transaction.

Halt transactions are identical to memory read transactions except that $\overline{HALT}$ is asserted throughout the transaction, falling during the second half of the first bus cycle, and remains asserted until an interrupt is acknowledged. This transaction is invoked when a Halt instruction is encountered in the instruction stream or a fatal sequence of traps occurs. Although the Halt transaction is three cycles, the $\overline{HALT}$ line remains asserted until an Interrupt request is acknowledged or a Reset is received. Refresh (to maintain a minimum frequency of bus transactions) or DMA transfers may occur while $\overline{HALT}$ is asserted; also, the bus can be granted. The address put out during the address phase of this cycle is the address of the Halt instruction.

I/O Transactions. I/O transactions move data to (Figure 42) or from (Figure 43) peripherals and are generated during the execution of I/O instructions.

I/O transactions are four clock cycles long at a minimum, and may be lengthened by the addition of wait cycles. The extra clock cycle allows for slower peripheral operation.

The $\overline{IORQ}$ line indicates that an I/O transaction is taking place. The I/O address is found on AD_0-AD_7 and A_8-A_{23} when $\overline{AS}$ rises.



* Address of HALT instruction.

Figure 41. Halt Timing

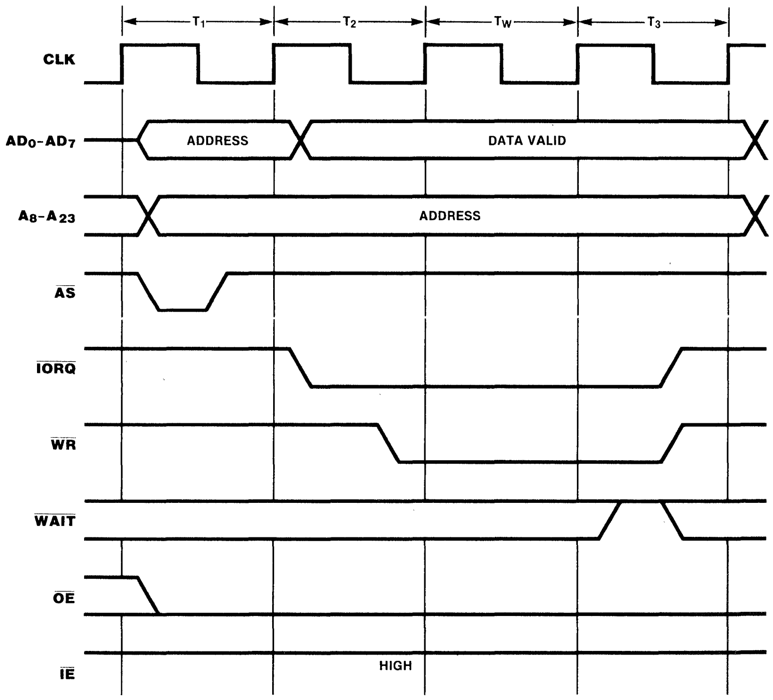


Figure 42. I/O Write Timing

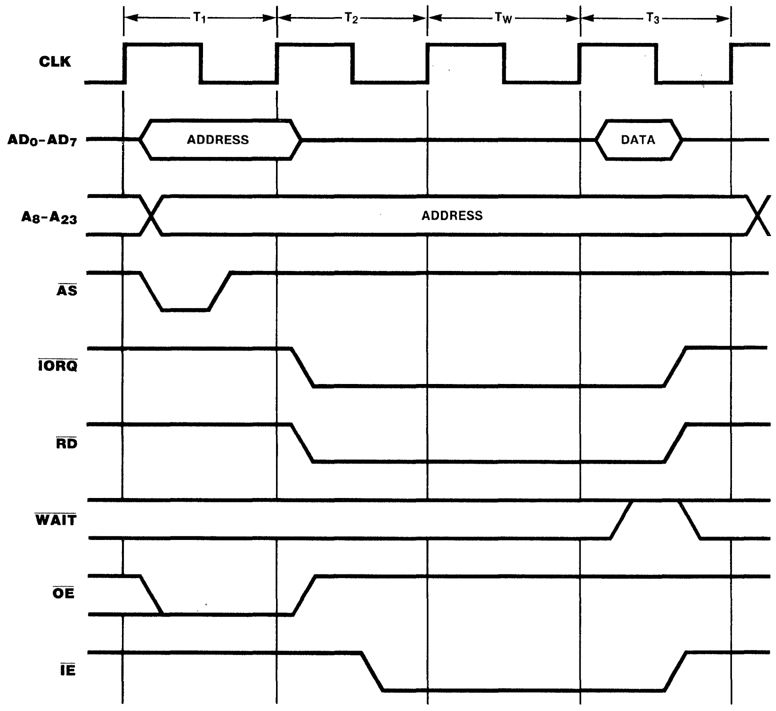


Figure 43. I/O Read Timing

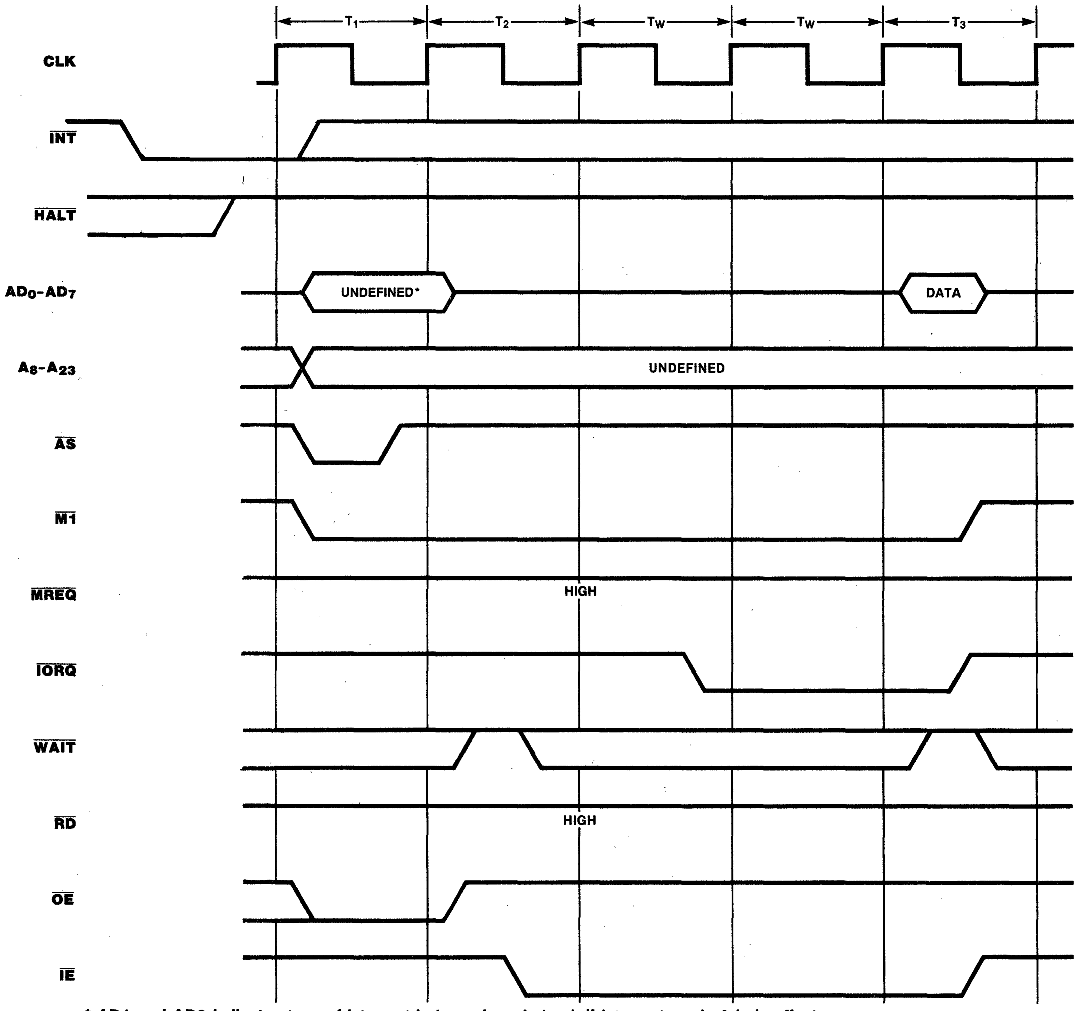
Interrupt Acknowledge Transactions. These transactions (Figure 44) acknowledge an interrupt and read information from the device that generated the interrupt. The transactions are generated automatically by the hardware when an external interrupt request is detected.

The Interrupt Acknowledge transactions are five cycles long at a minimum and have two automatic Wait cycles. The Wait cycles are used to give the interrupt priority daisy chain (or other priority resolution device) time to settle before the identifier is read. Additional automatic Wait states can be generated by programming the Bus Timing and Control register.

The Interrupt Acknowledge transaction is indicated by an $\overline{M1}$ assertion without $\overline{MREQ}$ during the first cycle. During this transaction the $\overline{IORQ}$ signal becomes active during the third cycle to indicate that the interrupting device can place an 8-bit vector on the bus. It is captured from the AD lines on the falling clock edge just before $\overline{IORQ}$ is raised High.

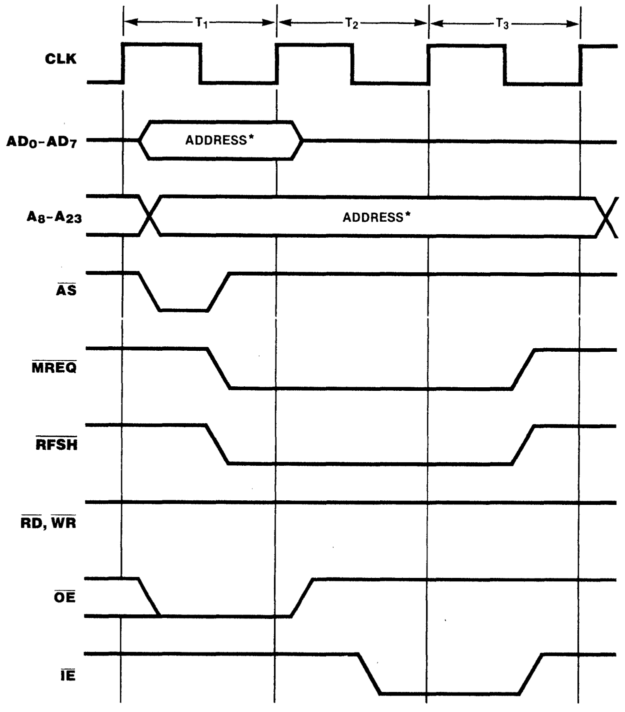
There are two places where the $\overline{WAIT}$ line is sampled and, thus, where a Wait cycle can be inserted by external circuitry. The first serves to delay the falling edge of $\overline{IORQ}$ to allow the daisy chain a longer time to settle, and the second serves to delay the point at which the vector is read.

Refresh Transactions. A memory refresh transaction (Figure 45) is generated by the Z280 refresh mechanism and can occur immediately after the final clock cycle of any other transaction. The memory refresh counter's 10-bit address is output on AD_0-AD_7 and A_8-A_9 during the normal time for addresses. The $\overline{RFSH}$ line is activated with $\overline{MREQ}$. This transaction can be used to generate refreshes for dynamic RAMs.



* AD1 and AD2 indicates type of interrupt being acknowledged, if interrupt mode 3 is in effect.

Figure 44. Maskable Interrupt Acknowledge Sequence



*10 least significant bits are Refresh address, the rest are undefined.

Figure 45. Refresh Timing

Requests

There are three kinds of request signals that the Z280 MPU supports. These are:

- Interrupt requests, which another device initiates and the CPU accepts and acknowledges.
- Bus requests, which an external potential bus master initiates and the Z280 MPU accepts and acknowledges.
- Global bus requests, which the CPU or on-chip DMA initiates to acquire a global System bus.

When a request is made, it is answered according to its type: for interrupt requests, an Interrupt Acknowledge transaction is initiated; for bus requests, an Acknowledge signal is sent; for global bus requests, an Acknowledge signal is received.

Interrupt Requests. The Z280 CPU supports two types of interrupt, maskable and nonmaskable ($\overline{NMI}$). The Interrupt Request line of a device that is capable of generating an interrupt can be tied to the $\overline{NMI}$ or maskable interrupt request lines. Several devices can be connected to one pin with the devices arranged in a priority daisy chain. However, all Z80 family peripherals should be on the same line (or no nesting of interrupts among different lines). The CPU uses different protocols for handling requests on the $\overline{NMI}$ pin than the protocol used for maskable interrupt pins. The sequence of events shown below should be followed:

Any High-to-Low transition on the $\overline{NMI}$ input is asynchronously edge-detected, and the internal $\overline{NMI}$ latch is set. At the beginning of the last clock cycle in the last internal machine cycle of any instruction, the interrupt inputs are sampled along with the state of the internal $\overline{NMI}$ latch.

If a maskable interrupt is requested and the Master Status register indicates that requests on that line are to be accepted, the next possible bus transaction is the Interrupt Acknowledge transaction, which results in information from the highest-priority interrupting device being read off the AD lines. This data is used to initiate the interrupt service routine. For a nonmaskable interrupt request, the hexadecimal constant 0066 is used to initiate the interrupt service routine, except in mode 3.

Bus Requests. To generate transactions on the bus, a potential bus master (such as the DMA Controller) must gain control of the bus by making a bus request. A bus request is initiated by pulling $\overline{BUSREQ}$ Low. Several bus requesters may be wired-OR to the $\overline{BUSREQ}$ pin; priorities are resolved externally to the CPU, usually by a priority daisy chain.

The asynchronous $\overline{\text{BUSREQ}}$ signal generates an internal $\overline{\text{BUSREQ}}$, which is synchronous. If the external $\overline{\text{BUSREQ}}$ is Low at the beginning of any machine cycle, the internal $\overline{\text{BUSREQ}}$ causes the Bus Acknowledge line ($\overline{\text{BUSACK}}$) to be asserted after the current machine cycle is completed. (Exceptions are the TSET instruction where the read-modify-write cycle is atomic and DMA transfer in burst or continuous mode.) The CPU then enters Bus Disconnect state and gives up control of the bus. All MPU Output pins, except $\overline{\text{BUSACK}}$, are 3-stated.

The CPU regains control of the bus after $\overline{\text{BUSREQ}}$ rises. Any device desiring control of the bus must wait at least two bus cycles after $\overline{\text{BUSREQ}}$ has risen before pulling it down again.

The on-chip DMA channels have higher priority than external devices requesting the bus via $\overline{\text{BUSREQ}}$.

Z-BUS External Interface

Features

- 16-bit data bus
- Multiplexed address/data lines
- Supports high-speed burst mode transfers
- Provides EPA interface

Pin Descriptions

A_{16}-A_{23}. *Address* (output, active High, 3-state). These address lines carry I/O addresses and memory addresses during bus transactions.

AD_0-AD_{15}. *Address/Data* (bidirectional, active High, 3-state). These 16 multiplexed address and data lines carry I/O addresses, memory addresses, and data during bus transactions.

$\overline{\text{AS}}$. *Address Strobe* (output, active Low, 3-state). The rising edge of Address Strobe indicates the beginning of a transaction and shows that the address, status, R/$\overline{\text{W}}$, and B/$\overline{\text{W}}$ signals are valid.

$\overline{\text{BUSACK}}$. *Bus Acknowledge* (output, active Low). A Low on this line indicates that the CPU has relinquished control of the bus in response to a bus request.

$\overline{\text{BUSREQ}}$. *Bus Request* (input, active Low). A Low on this line indicates that an external bus requester has obtained or is trying to obtain control of the bus.

B/$\overline{\text{W}}$. *Byte/Word* (output, Low = Word, 3-state). This signal indicates whether a byte or a word of data is to be transmitted during a transaction.

CLK. *Clock Output* (output). The frequency of the processor timing clock is derived from the oscillator input (external oscillator) or crystal frequency (internal oscillator) by dividing the crystal or external oscillator input by two. The processor clock is further divided by one, two, or four (as programmed), and then output on this line.

CTIN. *Counter/Timer Input* (input, active High). These lines receive signals from external devices for the counter/timers.

CTIO. *Counter/Timer I/O* (bidirectional, active High, 3-state). These I/O lines transfer signals between the counter/timers and external devices.

$\overline{\text{DMASTB}}$. *DMA Flyby Strobe* (output, active Low). These lines select peripheral devices for DMA flyby transfers.

$\overline{\text{DS}}$. *Data Strobe* (output, active Low, 3-state). This signal provides timing for data movement to or from the bus master.

$\overline{\text{EOP}}$. *End of Process* (input, active Low). An external source can terminate a DMA operation in progress by driving $\overline{\text{EOP}}$ Low. $\overline{\text{EOP}}$ always applies to the active channel; if no channel is active, $\overline{\text{EOP}}$ is ignored.

$\overline{\text{GACK}}$. *Global Acknowledge* (input, active Low). A Low on this line indicates the CPU has been granted control of a global bus.

$\overline{\text{GREQ}}$. *Global Request* (output, active Low, 3-state). A Low on this line indicates the CPU has obtained or is trying to obtain control of a global bus.

$\overline{\text{IE}}$. *Input Enable* (output, active Low, 3-state). A Low on this line indicates that the direction of transfer on the Address/Data lines is toward the CPU.

$\overline{\text{INT}}$. *Maskable Interrupts* (input, active Low). A Low on these lines requests an interrupt.

$\overline{\text{NMI}}$. *Nonmaskable Interrupt* (input, falling-edge activated). A High-to Low transition on this line requests a nonmaskable interrupt.

$\overline{\text{OE}}$. *Output Enable* (output, active Low, 3-state). A Low on this line indicates that the direction of transfer on the Address/Data lines is away from the MPU.

OPT. *Bus Option* (input). This signal establishes the bus option during reset as follows:

OPT	Bus Interface
0	Z80-Bus, 8-bit
1	Z-BUS, 16-bit

$\overline{\text{PAUSE}}$. *CPU Pause* (input, active Low). While this line is Low the CPU refrains from transferring data to or from an Extended Processing Unit in the system or from beginning the execution of an instruction.

$\overline{\text{RDY}}$. *DMA Ready* (input, active Low). These lines are monitored by the DMA channels to determine when a peripheral device associated with a DMA channel is ready for a read or write operation. When a DMA channel is enabled to operate, its Ready line indirectly controls DMA activity; the manner in which DMA activity is controlled by the line varies with the operating mode (single-transaction, burst, or continuous).

$\overline{\text{RESET}}$. *Reset* (input, active Low). A Low on this line resets the CPU and on-chip peripherals.

R/$\overline{\text{W}}$. *Read/Write* (output, Low = Write, 3-state). This signal determines the direction of data transfer for memory, I/O, or EPU transfer transactions.

RxD. *UART Receive* (input, active High). This line receives serial data at standard TTL levels.

ST_0-ST_3. *Status* (output, active High, 3-state). These four lines indicate the type of transaction occurring on the bus and give additional information about the transaction.

TxD. *UART Transmit* (output, active High). This line transmits serial data at standard TTL levels.

$\overline{\text{WAIT}}$. *Wait* (input, active Low). A Low on this line indicates that the responding device needs more time to complete a transaction.

XTALI. *Clock/Crystal Input* (time-base input). Connects a parallel-resonant crystal or an external single-phase clock to the on-chip clock oscillator.

XTALO. *Crystal Output* (time-base output). Connects a parallel-resonant crystal to the on-chip clock oscillator.

+5V. *Power Supply Voltage.* (+5 nominal).

GND. *Ground.* Ground reference.

Bus Operations

Two kinds of operations can occur on the system bus: transactions and requests. At any given time, one device (either the CPU or a bus requester) has control of the bus and is known as the bus master. A transaction is initiated by the bus master and is responded to by some other device on the bus. Only one transaction can proceed at a time; eight kinds of transactions can occur:

Burst Memory. These transactions are used to transfer four words of instructions from the memory to the CPU.

DMA Flyby. This transaction is used by the DMA peripheral to transfer data between an external peripheral and memory.

EPU Transfer. This transaction is used to transfer data between the CPU and an EPU.

Halt. This transaction is used to indicate that the CPU is entering the Halt state.

Interrupt Acknowledge. This transaction is used by the CPU to acknowledge an external interrupt request and to transfer additional information from the interrupting device.

I/O. This transaction is used by the bus master to transfer data to or from an external peripheral.

Memory. This transaction is used by the bus master to transfer data to or from a memory location.

Refresh. These transactions by the refresh mechanism do not transfer data; they refresh dynamic memory.

Only the bus master can initiate transactions. A request, however, can be initiated by a device that does not have control of the bus. Two types of requests can occur:

Bus. This request is used to request control of the bus to initiate transactions.

Interrupt. This request is used to request servicing by the CPU.

When an interrupt or bus request is made, it is answered according to its type: for an externally generated interrupt request, an Interrupt Acknowledge transaction is initiated by the CPU; for bus requests, the MPU enters Bus Disconnect state, relinquishes the bus, and activates an acknowledge signal.

Transactions

Data transfers to and from the Z280 MPU are accomplished through the use of transactions.

All transactions start with Address Strobe ($\overline{\text{AS}}$) being driven Low and then raised High by the Z280 MPU. On the rising edge of $\overline{\text{AS}}$, the Status lines ST_0-ST_3 are valid; these lines indicate the type of transaction being initiated (Table 9); seven types of transactions are discussed in the sections that follow. Associated with the status lines are two other lines that become valid at this time: R/$\overline{\text{W}}$, and B/$\overline{\text{W}}$.

Table 9. Status Code Table

Status Lines 3••0	Type of Transaction
0000	Reserved
0001	Refresh
0010	I/O transaction
0011	Halt
0100	Interrupt acknowledge line A
0101	$\overline{\text{NMI}}$ acknowledge
0110	Interrupt acknowledge line B
0111	Interrupt acknowledge line C
1000	Transfer between CPU and memory, cacheable
1001	Transfer between CPU and memory, non-cacheable
1010	Data transfer between EPU and memory
1011	Reserved
1100	EPU Instruction fetch, template, subsequent words
1101	EPU Instruction fetch, template, first word
1110	Data transfer between EPU and CPU
1111	Test and Set (data transfers)

If the transaction requires an address, it is valid on the rising edge of $\overline{\text{AS}}$. No address is required for EPU-CPU transfer transactions; the contents of the A and AD lines while $\overline{\text{AS}}$ is asserted are undefined. If an address is generated, the $\overline{\text{OE}}$ signal is also activated.

The Z-BUS MPUs use Data Strobe ($\overline{DS}$) to time the actual data transfer. (Note that Refresh and Halt transactions do not transfer any data and thus do not activate $\overline{DS}$.) For write operations (R/$\overline{W}$ = Low), a Low on $\overline{DS}$ indicates that valid data from the bus master is on the AD lines. The Output Enable continues to be asserted until $\overline{DS}$ is deasserted. For read operations (R/$\overline{W}$ = High), the bus master makes AD lines 3-state, deasserts $\overline{OE}$, and asserts $\overline{IE}$ after driving $\overline{DS}$ Low so that the addressed device can put its data on the bus. The bus master samples this data on the falling clock edge just before raising $\overline{DS}$ and $\overline{IE}$ High.

Wait Cycle. The $\overline{WAIT}$ line is sampled on the falling clock edge when data is sampled by the Z280 MPU (Read) or the falling clock edge before $\overline{DS}$ rises (Read or Write). If $\overline{WAIT}$ is Low, another cycle is added to the transaction before data is sampled or $\overline{DS}$ rises. In this added cycle, and all subsequent cycles added when $\overline{WAIT}$ is Low, $\overline{WAIT}$ is again sampled on the falling clock edge and, if it is Low, another cycle is added to the transaction. In this way, the transaction can be extended to an arbitrary length by external circuitry to accommodate (for example) slow memories or I/O devices that are not yet ready for data transfer. Automatic insertions of wait states by the CPU or on-chip DMA channels can be programmed by setting fields in the Bus Timing and Control register and Bus Timing and Initialization register to indicate the number to be inserted.

Memory Transactions. Memory transactions move data to or from memory when a bus master makes a memory access. Thus, they are generated during program execution to fetch instructions from memory and to fetch and store memory data. They are also generated to store old program status and fetch new program status during interrupt and trap handling and after reset.

A memory transaction is three bus cycles long unless extended when $\overline{WAIT}$ is asserted.

Bytes transferred to or from odd memory locations (address bit 0 = 1) are always transmitted on lines AD$_0$-AD$_7$ (bit 0 on AD$_0$). Bytes transferred to or from even memory locations (address bit 0 = 0) are always transmitted on lines AD$_8$-AD$_{15}$ (bit 0 on AD$_8$). For byte reads (B/$\overline{W}$ High, R/$\overline{W}$ High), the CPU or on-chip DMA channel uses only the byte whose address it put out on the bus. For byte writes (B/$\overline{W}$ High, R/$\overline{W}$ Low), the memory should store only the byte whose address was output. During byte memory writes, the CPU (or on-chip DMA channel in non-Flyby transactions) places the same byte on both halves of the bus, and the proper byte must be selected by testing A$_0$. For word transfers (B/$\overline{W}$ = Low), all 16 bits are captured by the CPU or DMA channel (Read: R/$\overline{W}$ = High) or stored by the memory (Write: R/$\overline{W}$ = Low). For these transactions (either memory or I/O) the bytes of data appear swapped on the bus with the most significant byte on AD$_7$-AD$_0$ and the least significant byte on AD$_{15}$-AD$_8$. A word is aligned if the address is even; otherwise it is unaligned.

Memory transaction timings are shown in Figures 46-50.

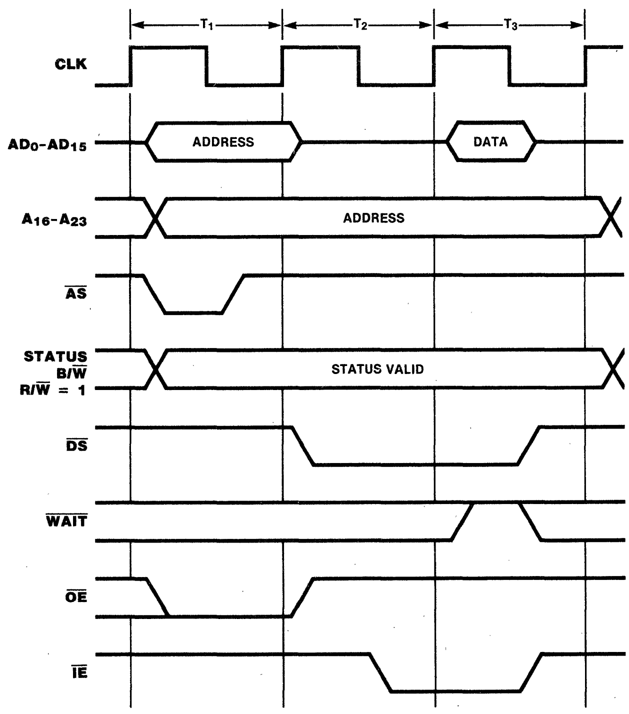


Figure 46. Memory Read Timing

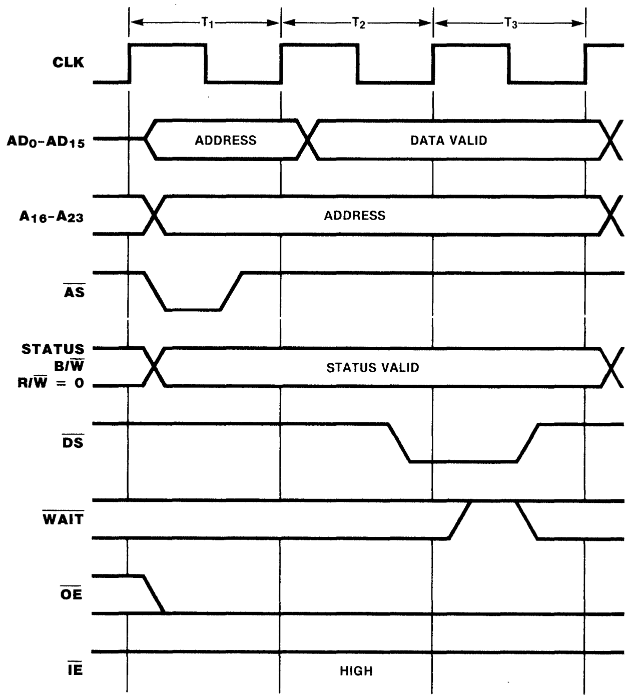


Figure 47. Memory Write Timing

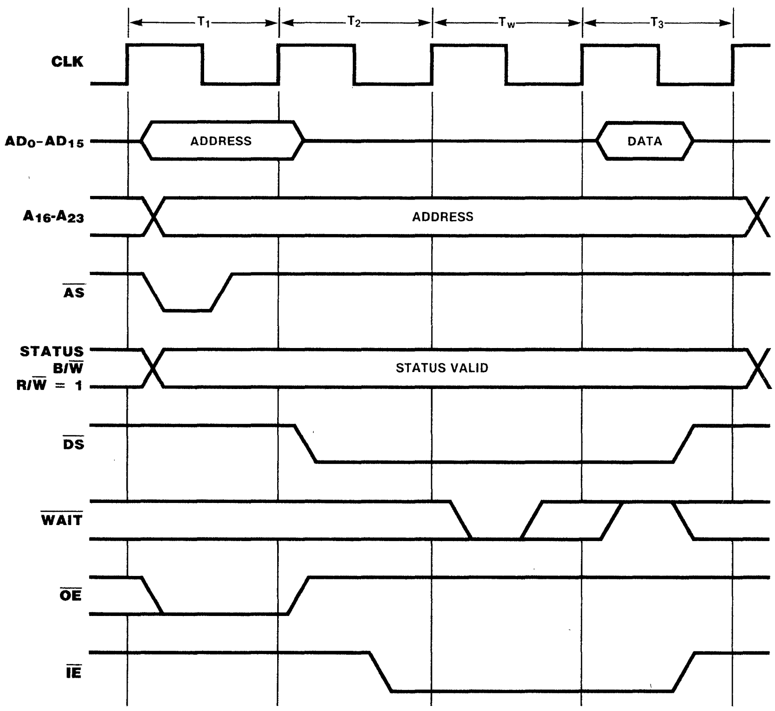


Figure 48. Memory Read Timing with External Wait Cycle

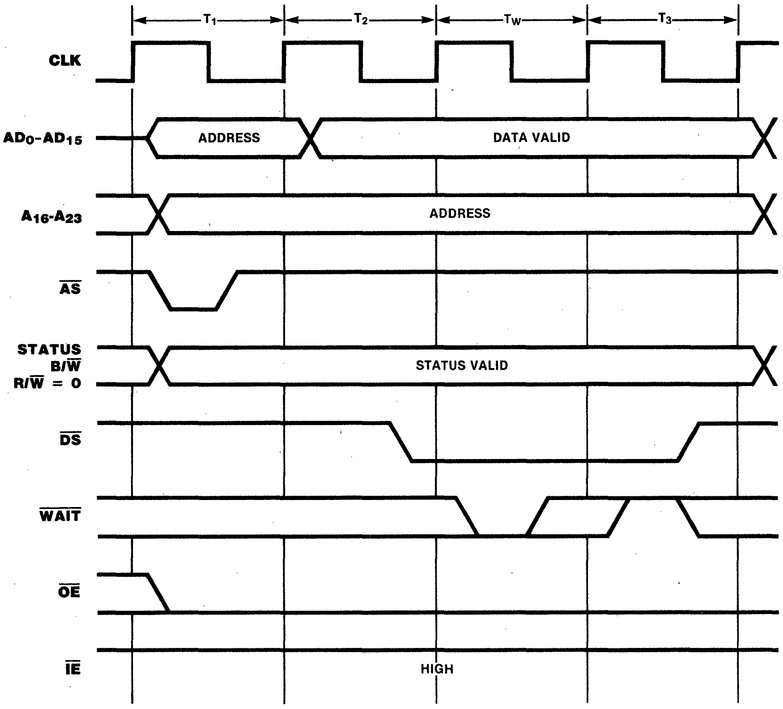


Figure 49. Memory Write Timing with External Wait Cycle

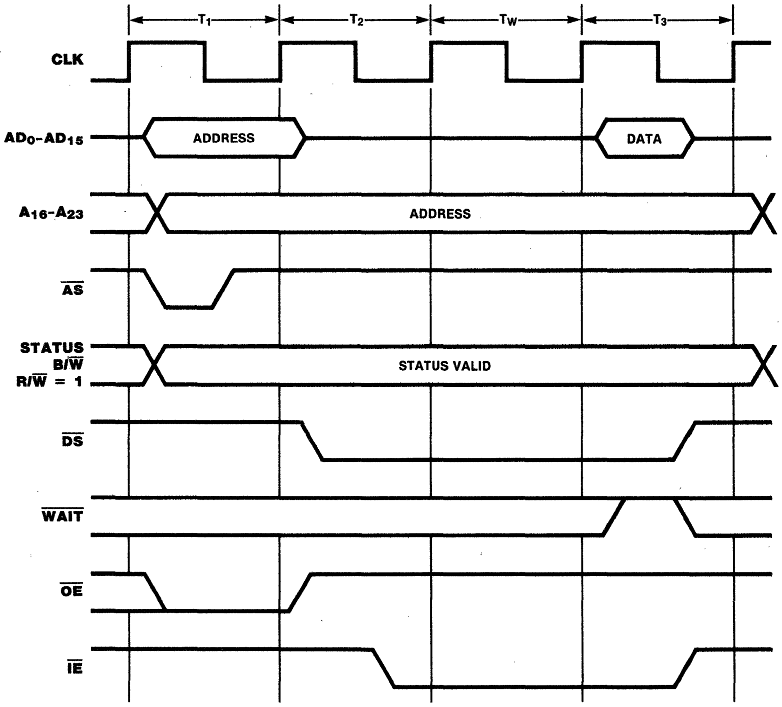


Figure 50. Memory Read Timing with Internal Wait Cycle

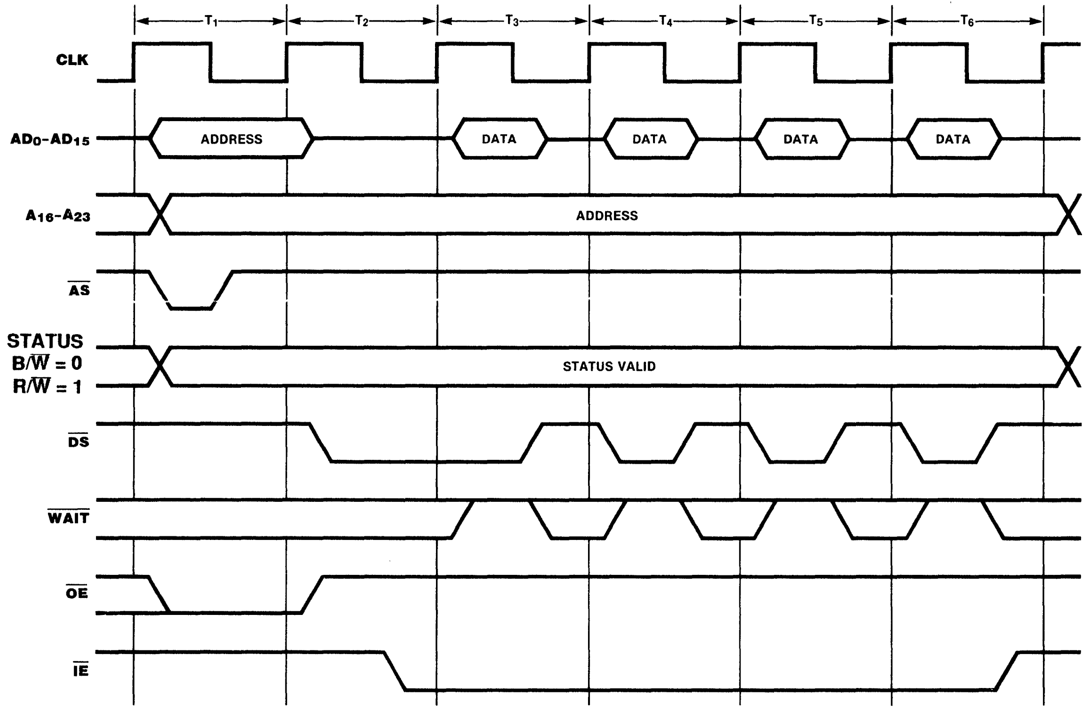


Figure 51. Burst Memory Read Timing

Burst Memory Transactions. Burst memory transactions use multiple Data Strobes associated with a single Address Strobe. The CPU uses burst transactions to read four consecutive words in four data transactions. The address of the first word read during a burst transaction has zeros in the three least significant bits. Control bits in the Cache Control register indicate whether or not portions of the memory system can support burst transactions.

The CPU uses burst mode reads only for fetching instructions. If an instruction is to be fetched from a location within a half of physical memory that supports burst transactions, the CPU reads the eight bytes that contain the first byte of the instruction. (EPA template fetches do not use the burst transaction.)

Timing for the first data transfer during a burst transaction is identical to that for a single memory read, including the automatic insertion of wait states, except there are four T_3 states. Subsequent data transfers do not include automatic wait states. On the first data transfer, if $\overline{\text{WAIT}}$ is sampled active then it is sampled again every bus clock cycle until it is inactive, at which time the data is read from the bus. Burst memory read timing is shown in Figure 51.

Note: Burst Transactions can occur only in Z-BUS mode.

Halt Transactions. Halt transactions do not transfer data. They look like a memory transaction, except that $\overline{\text{DS}}$ remains High and no data is transferred.

A Halt transaction (Figure 52) is generated when the CPU executes a HALT instruction or when a fatal sequence of traps and bus errors occurs. The address placed on the AD lines is the location of the Halt instruction or the instruction that initiated the fatal sequence of traps and errors. The Status lines indicate a Halt transaction (0011).

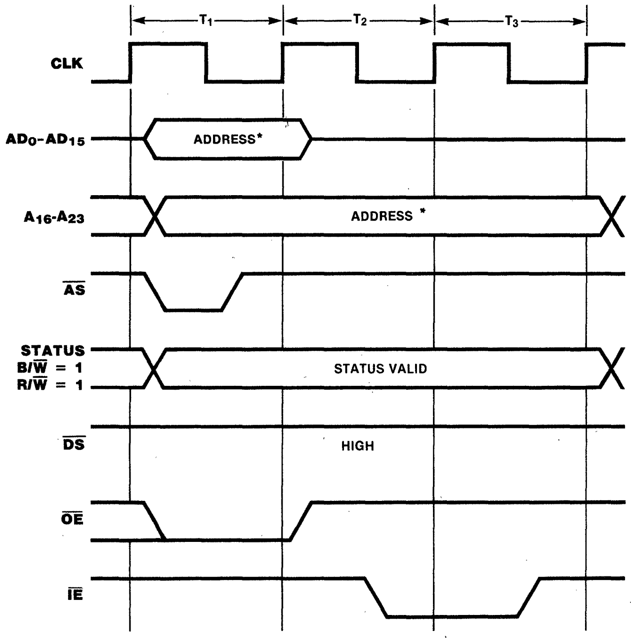
$\overline{\text{WAIT}}$ is not sampled during the Halt transaction.

I/O Transactions. I/O transactions (Figures 53 and 54) move data to or from peripherals and are generated during the execution of I/O instructions. I/O transactions to on-chip peripheral devices (I/O pages FE_H and FF_H) do not generate external bus transactions.

I/O transactions are four bus cycles long at a minimum, and they can be lengthened by the addition of wait cycles either automatically generated as indicated in the Bus Timing and Control register or generated by an external device. The extra clock cycles allow for slower peripheral operation.

The status lines indicate that the access is an I/O transaction (0010). The I/O address is found on AD_0-AD_{15} and A_{16}-A_{23}.

Byte data (B/$\overline{\text{W}}$ = High) is transmitted on AD_0-AD_7. This allows peripheral devices to attach to only eight of the AD lines. Word data (B/$\overline{\text{W}}$ = Low) is transmitted with the most significant byte on AD_0-AD_7 and the least significant byte on AD_8-AD_{15}.



*Address of Halt Instruction.

Figure 52. Halt Timing

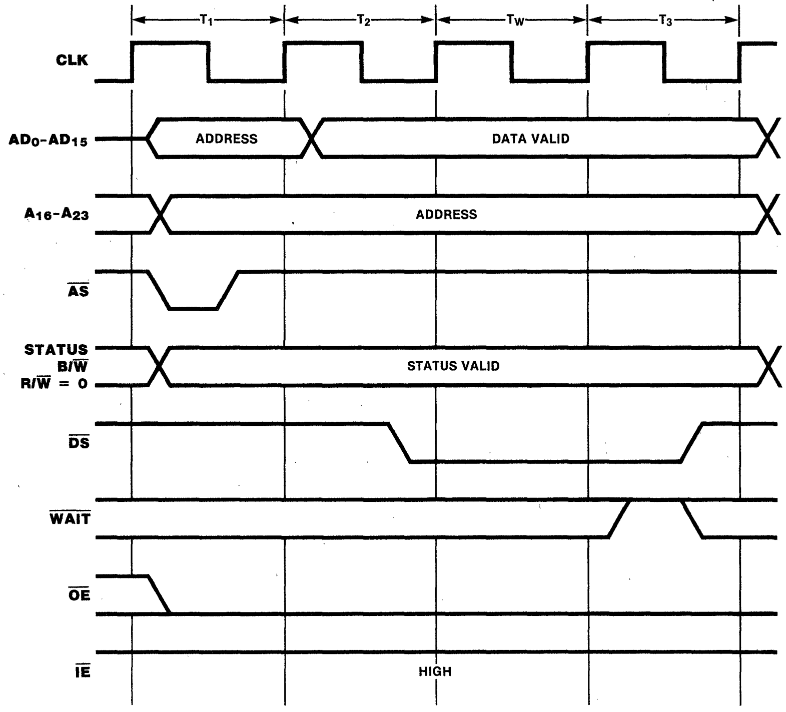


Figure 53. I/O Write Timing

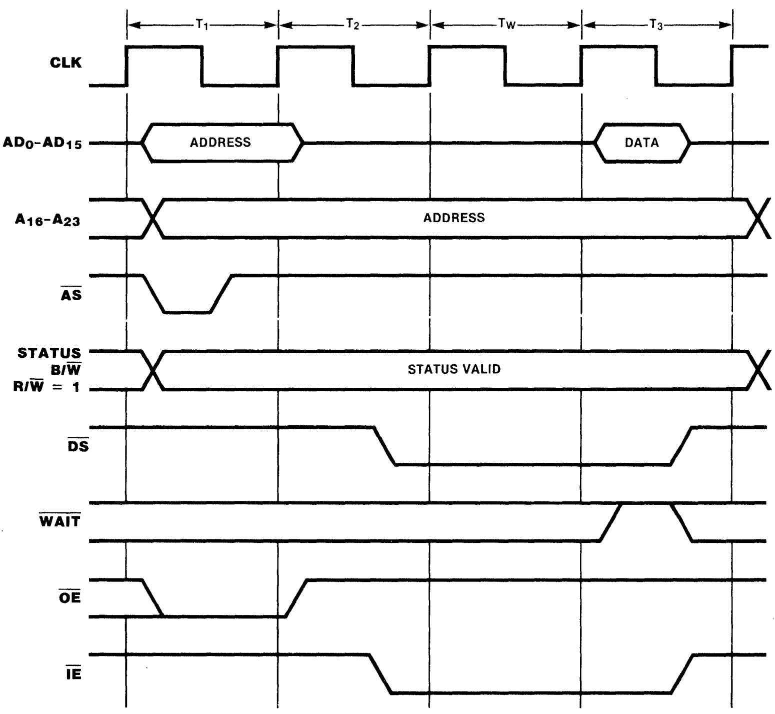


Figure 54. I/O Read Timing

Interrupt Acknowledge Transactions. These transactions (Figure 55) acknowledge an interrupt and read an identifier from the device that generated the interrupt. Interrupt Acknowledge transactions are generated automatically by the hardware when an external interrupt is detected.

These transactions are five cycles long at a minimum, with at least two automatic Wait cycles, although others can be added by programming the Bus Timing and Control register. The Wait cycles are used to give the interrupt priority daisy chain (or other priority resolution device) time to settle before the identifier is read.

The only item of data transferred is the identifier that is captured from the AD lines on the falling clock edge just before $\overline{DS}$ is raised High. The length of time that $\overline{DS}$ is asserted is identical with I/O timing programmed in the Bus Timing and Control register.

There are two places where $\overline{WAIT}$ is sampled and thus a Wait cycle can be inserted by external devices. The first place serves to delay the falling edge of $\overline{DS}$ to allow the daisy chain a longer time to settle, and the second place serves to delay the point at which data is read.

Refresh Transactions. A memory Refresh transaction (Figure 56) is generated by the refresh mechanism and can come immediately after the final clock cycle of any other transaction. The memory refresh counter's 10-bit address is output on the low order 10 bits of the bus during the first cycle of the transaction. The contents of the rest of the bus are undefined. The Status lines indicate Refresh (0001). This transaction can be used to generate refreshes for dynamic RAMs. Refreshes may occur while the CPU is in the Halt or Fatal state.

CPU-Extended Processing Unit Interaction

The Z280 CPU with a Z-BUS interface and $\overline{PAUSE}$ input line and one or more Extended Processing Units (EPUs) work together like a single CPU component, with the CPU providing address, status, and timing signals and the EPU supplying and capturing data. The EPU monitors the status and timing signals output by the CPU so that it knows when to participate in a memory transaction; for EPU to memory transfers, the CPU puts its AD lines in 3-state while $\overline{DS}$ is Low, so that the EPU can use them.

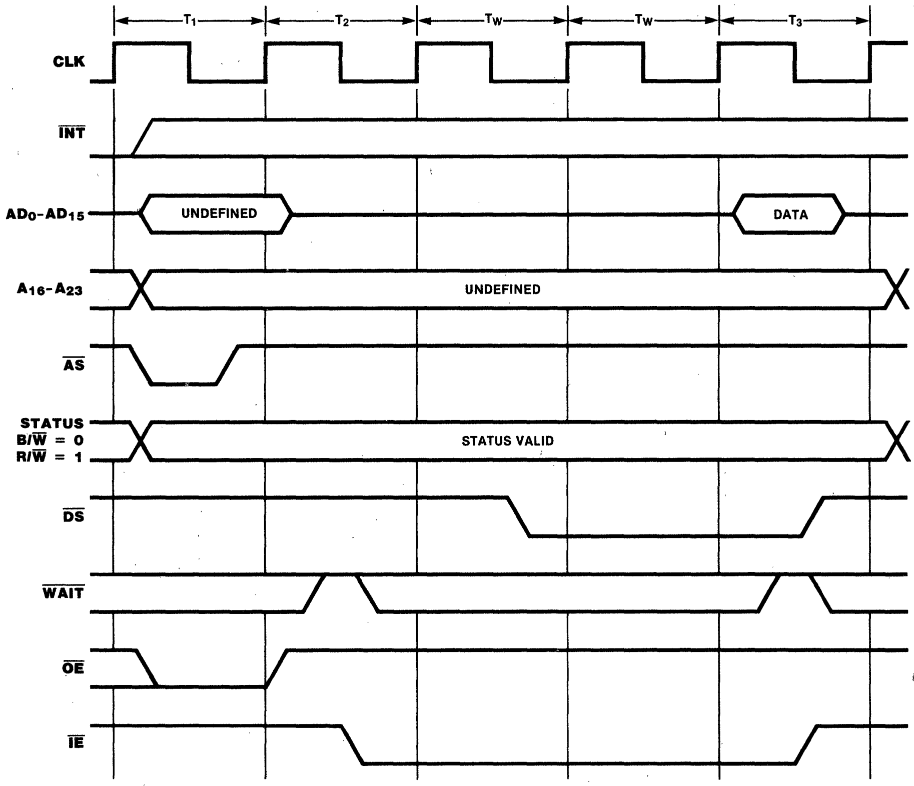
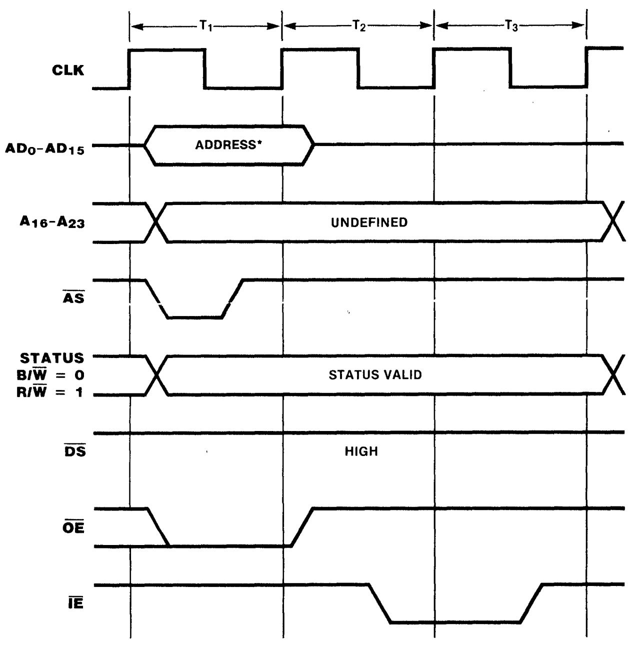


Figure 55. Interrupt Acknowledge Timing



*10 least-significant bits are Refresh address.

Figure 56. Memory Refresh Timing

In order to know which transaction it is to participate in, the EPU must track the following sequence of events:

- When the CPU fetches the first word of an EPA instruction template from memory (ST_3-ST_0 = 1101), the EPU must also capture the instruction returned by the memory. Within the template is an ID field that indicates whether or not the EPU is to execute the instruction.
- The next non-refresh transaction by the CPU is the fetch of the second word of the instruction (ST_3-ST_0 = 1100). The EPU must also capture this word. If the template is not aligned, a third fetch is made (ST_3-ST_0 = 1100).
- If the instruction involves a read or write to memory, then transfers of data between memory and the EPU (ST_3-ST_0 = 1010) are the next non-refresh transactions performed by the CPU. The EPU must supply the data (Write: $R/\overline{W}$ = Low) or capture the data (Read: $R/\overline{W}$ = High) for each transaction, just as if it were part of the CPU. In both cases, the CPU 3-states its AD lines while data is being transferred ($\overline{DS}$ Low).
- If the instruction involves a transfer from the EPU to the Z280 MPU, the next non-refresh transaction is the CPU transferring data between the EPU and CPU (ST_3-ST_0 = 1110).

In order to follow this sequence, an EPU has to monitor the status lines to verify that the transaction it is monitoring on the bus was generated by the CPU. In a multiple EPU system, there is no indication on the bus as to which EPU is cooperating with the CPU at any given time. This must be determined by the EPUs from the templates they capture.

When an EPU begins to execute an extended instruction, the CPU can continue fetching and executing instructions. If the EPU wishes to halt the CPU from executing another instruction or bus transaction, the EPU must activate the $\overline{PAUSE}$ line to stop the CPU until the EPU is ready for subsequent MPU activity. This mechanism is used to synchronize MPU-EPU activity.

EPU Transfer Transactions. These transactions (Figures 57-59) allow the CPU to transfer data to or from an EPU or to read or write an EPU's status registers. They are generated during the execution of the EPU instructions.

EPU-to-Memory transfers are five cycles unless extended by $\overline{\text{WAIT}}$. Memory-to-EPU transfers are three cycles unless extended by $\overline{\text{WAIT}}$.

EPU-CPU transfer transactions have the same form as I/O transactions and thus are four clock cycles long, unless extended by $\overline{\text{WAIT}}$. Although $\overline{\text{AS}}$ is asserted, no address is generated and the contents of the bus are undefined; only one status code is used (1110).

In a multiple EPU system, the EPU that is to participate in a transaction is selected implicitly by the ID code in the EPU template, rather than by an address. The Read/Write line (R/$\overline{\text{W}}$ = High) indicates the direction of the data transfer into the CPU.

Requests

The Z280 MPU supports three types of request signals. These are:

- Interrupt requests, which another device initiates and the CPU accepts and acknowledges.
- Bus requests, which an external potential bus master initiates and the CPU accepts and acknowledges.
- Global bus requests, which the CPU or on-chip DMA initiates to acquire a global system bus.

When a request is made, it is answered according to its type: for interrupt requests, an Interrupt Acknowledge transaction is initiated by the CPU; for bus requests, an acknowledge signal is sent; for global bus request, an acknowledge signal is received.

Interrupt Requests. The Z280 MPU supports two types of external interrupts, maskable and nonmaskable ($\overline{\text{NMI}}$). The Interrupt Request line of a device that is capable of generating an interrupt may be tied to any of the interrupt pins. Several devices can be connected to one pin, with the devices arranged in a priority daisy chain. The CPU uses the same protocol for handling requests on these pins. The sequence of events is given below:

Any High-to-Low transition on the $\overline{\text{NMI}}$ input is asynchronously edge-detected, and the internal $\overline{\text{NMI}}$ latch is set. At the beginning of the last processor clock cycle of any instruction, the interrupt inputs are sampled along with the state of the internal $\overline{\text{NMI}}$ latch.

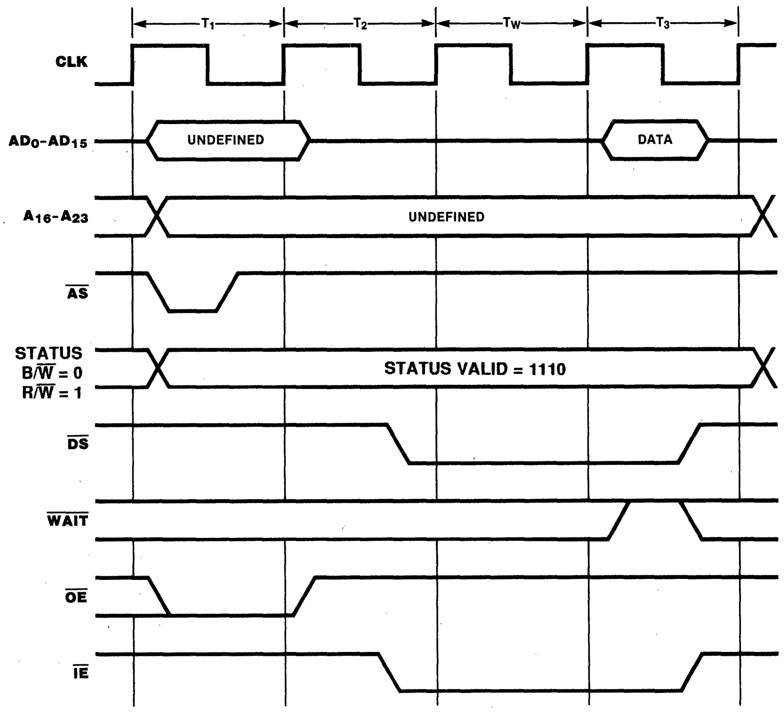


Figure 57. EPU to CPU Timing

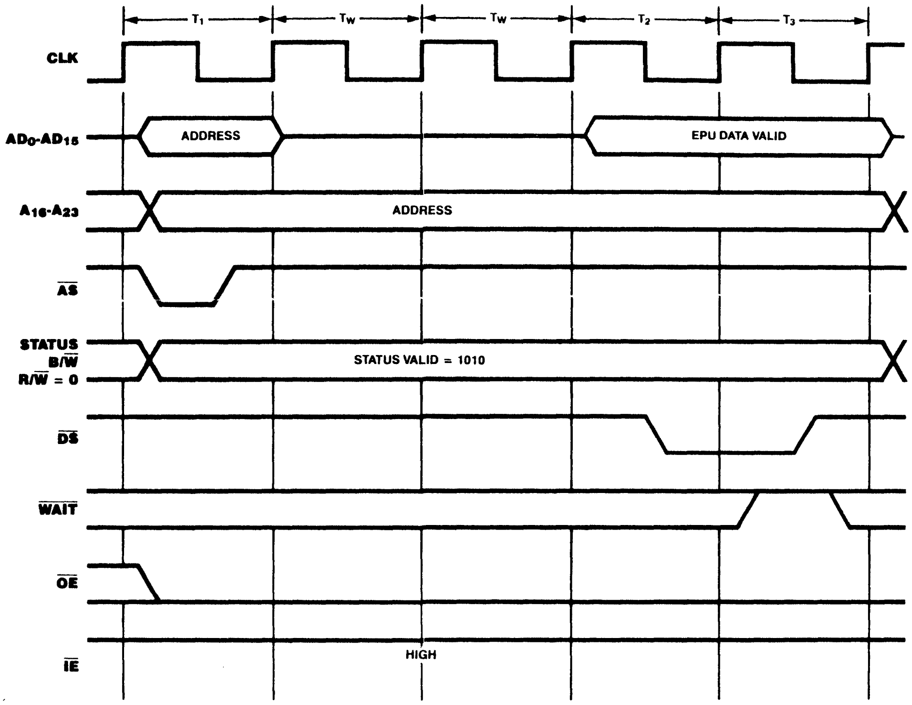


Figure 58. EPU Write to Memory

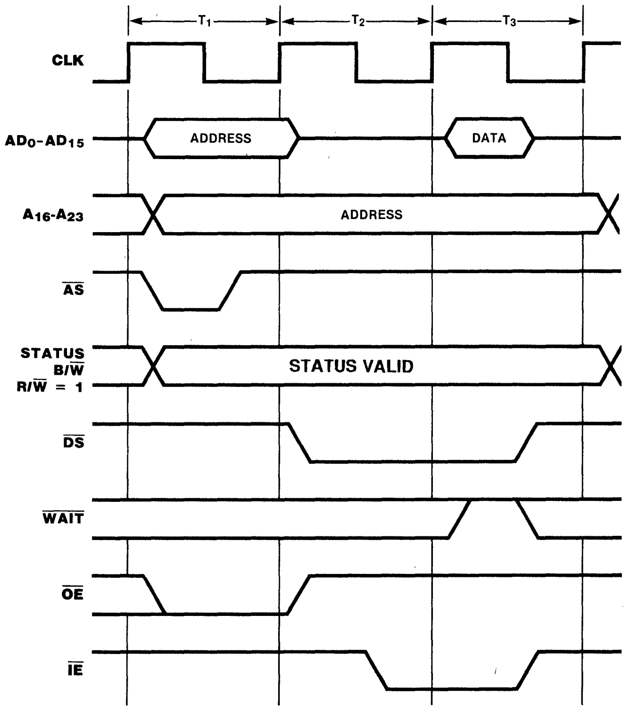


Figure 59. Memory to EPU Timing

If a maskable interrupt is requested and the Master Status register indicates that requests on that line are to be accepted, or if the $\overline{\text{NMI}}$ latch is set, the next possible bus transaction is an interrupt acknowledge transaction that results in an identifier from the highest-priority interrupting device being read off the AD lines. This data is used as specified by the current interrupt mode.

Bus Requests. To generate transactions on the bus, a potential external bus master (such as a DMA Controller) must gain control of the bus by making a bus request. A bus request is initiated by pulling $\overline{\text{BUSREQ}}$ Low. Several bus requesters can be wired-OR to the $\overline{\text{BUSREQ}}$ pin; priorities are resolved externally to the CPU, usually by a priority daisy chain.

The asynchronous $\overline{\text{BUSREQ}}$ signal generates an internal $\overline{\text{BUSREQ}}$, which is synchronous. If the external $\overline{\text{BUSREQ}}$ is Low at the beginning of any processor clock cycle, the internal $\overline{\text{BUSREQ}}$ will cause the bus acknowledge line ($\overline{\text{BUSACK}}$) to be asserted after the current bus transaction is completed or after the write transaction of a TSET instruction. The CPU then enters Bus Disconnect state and gives up control of the bus. All Z280 Output pins except $\overline{\text{BUSACK}}$ are 3-stated.

The on-chip DMA channels have higher priority than the off-chip devices requesting the external bus via $\overline{\text{BUSREQ}}$.

RESET

A hardware reset puts the Z280 MPU into a known state and optionally initializes the Bus Timing and Initialization control register of the Z280 MPU to a system specifiable value. A reset begins at the end of any processor clock cycle if the $\overline{\text{RESET}}$ line is Low. However, if a bus transaction is in progress it is allowed to be completed. A system reset overrides all other operations of the chip, including interrupts, traps and bus requests. A reset should be used to initialize a system as part of the power-up sequence.

The $\overline{\text{RESET}}$ input must be asserted for a minimum of 128 processor clock cycles. Within this time the Z280 lines assume their reset values. For either bus, the AD lines are 3-stated, and all control outputs are forced High. While $\overline{\text{RESET}}$ is asserted, the CLK output is the processor clock frequency scaled by four.

The $\overline{\text{RESET}}$ line is sampled on the rising edge of an internal clock (derivative of XTALi). When the $\overline{\text{RESET}}$ line is sampled High (de-asserted), the state of the $\overline{\text{WAIT}}$ line is also noted: if $\overline{\text{WAIT}}$ is asserted, then the contents of the AD lines are used to program the Bus Timing and Initialization register, otherwise the constant 00 hexadecimal is used. If the hardware programming initialization option is used, AD4 must be 0 when the bus is sampled and the AD6 line determines whether the UART bootstrap option is selected.

After reset, the Z280 MPU is initialized as shown in Tables 10 and 11.

The following registers are unaffected:

- CPU register file, including user Stack Pointer
- Page Descriptor registers
- Interrupt/Trap Vector Table Pointer register

On the rising edge of $\overline{\text{RESET}}$, if Bus Request is asserted the Z280 MPU will grant the bus before fetching the first instruction from location 0.

After $\overline{\text{RESET}}$ has returned to High, the CPU begins to operate unless the Bootstrap UART feature is utilized.

Table 10. Effect of a Reset on Z280 CPU and MMU Registers

Register	Value Loaded on Reset (Hexadecimal)	Comments
Program Counter	0000	
System Stack Pointer	0000	
I	00	
R	00	
Master Status	0000	System mode, Single-Step disabled, Breakpoint-on-Halt disabled All maskable interrupts disabled
Bus Timing and Control	**00****	No automatic wait states for I/O, upper 8M bytes of memory, or interrupt acknowledges
Bus Timing and Initialization	**00**	CLK output 2x processor clock period, no automatic wait states for lower 8 Mbytes of memory, bootstrap **mode disabled, direct clock option disabled, multiprocessor configuration disabled**
I/O Page	00	I/O Page 0 in use
Cache Control	20	Cache enabled for instructions All valid bits cleared to 0 Burst mode disabled
Trap Control	00	**EPA trap enabled, I/O not privileged, System Stack Overflow Warning trap disabled**
System Stack Limit	0000**	
Local Address	00	All memory transactions are made to local bus
Interrupt Status	00xx	Interrupt mode 0, nonvectored interrupts, current state of interrupt requests (indicated by xx)
Interrupt/Trap Vector Table Pointer		Unaffected
CPU Registers AF, BC, DE, HL, IX, IY, AF′, BC′, DE′, DE′, HL		Unaffected
User Stack Pointer		Unaffected
MMU Master Control	0000**	MMU disabled
MMU Page Descriptor Register, Page Descriptor Register Pointer		Unaffected

Table 11. Effect of a Reset on Z280 On-Chip Peripheral Registers

Register	Value Loaded on Reset (Hexadecimal)	Comments
Refresh	88	Refresh enabled, rate = 32
Counter/Timers:		
Configuration	00	**Timer mode, single-cycle, non-retrigger**
Command/Status	00	Timer disabled
DMA Channels:		
Master Control	0000*,**	No DMA linking, EOP disabled, Software Ready disabled
DMA0 Transaction Descriptor	0100*	DMA0 disabled, continuous mode
DMA1/2/3 Transaction Descriptor	—	EN, IE, TC, and EPS fields cleared, other fields unaffected
DMA0 Destination Address	000000	
DMA0 Count	0100	
UART:		
Configuration	00*	5 bits/character, parity disabled, external clock, x1 clock rate, loop back disabled
Transmitter Control/Status	01	Transmitter disabled, transmit buffer empty
Receiver Control/Status	00*	Receiver disabled

*Unless bootstrap mode is selected.

****Reserved bits are undefined on reads.**

ABSOLUTE MAXIMUM RATINGS

Voltage on Vcc with respect to Vss ..... -0.3V to + 7V
Voltages on all pins with respect to Vss ...... -0.3V to (Vcc + 0.3V)
Operating Ambient
Temperature . . . . .See Ordering Information
Storage Temperature . . . . –65°C to +150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

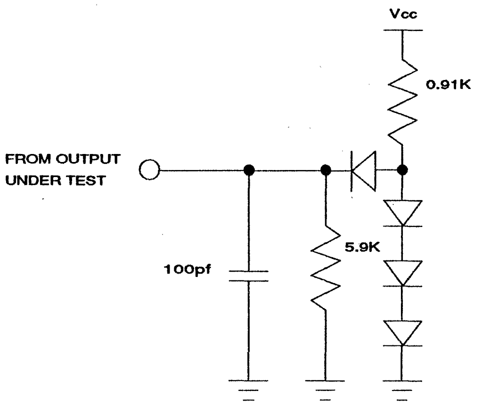
STANDARD TEST CONDITIONS

The DC Characteristics and Capacitance sections below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND (0V). Positive current flows into the referenced pin.

Available operating temperature ranges are:

■ S = 0°C to +70°C

All ac parameters assume a load capacitance of 100pf. Add 10 ns delay for each 50 pf increase in load up to a maximum of 200 pf for the data bus.



DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{IL}	Input Low Votage	-0.3	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 4.0 ma
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400 µa
V_{CC}	Operating Power Supply Voltage	4.5	5.5	V	
I_{CC}	Power Supply Current		200	ma	V_{CC} = 5.5 V XTALI = 20 MHz V_{IH} = 2.0 V V_{IL} = 0.8 V Outputs Unloaded

AC CHARACTERISTICS

Z-Bus Timing **(Refer to Figures 60 and 61)**

Number	Symbol	Parameter	Min	Max	Notes†
1	TdCr(ST)	Clock ↑ to Status Delay		20	
2	TdCr(A)	Clock ↑ to Address Valid Delay		20	
3	TdCr(ASf)	Clock ↑ to $\overline{AS}$ ↓ Delay		20	
4	TdCf(ASr)	Clock ↓ to $\overline{AS}$ ↑ Delay		20	
5	TwAS	$\overline{AS}$ Low Width	nTcXT - 20		1
6	TdCr(Az)	Clock ↑ to Address Float Delay		25	
7	TdCr(DSf)	Clock ↑ to $\overline{DS}$ ↓ Delay		20	
8	TdCf(DSr)	Clock ↓ to $\overline{DS}$ ↑ Delay		35	
9	TsD(Cf)	Data to Clock ↓ Setup	30		
10	ThD(Cf)	Data from Clock ↓ Hold	10		
11	TdCf(DSf)	Clock ↓ to $\overline{DS}$ ↓ Delay		20	
12	TdCr(D)	Clock ↑ to Data Valid Delay		20	
13	TdDSr(Dx)	$\overline{DS}$ ↑ to Data not Valid Delay	nTcXT - 40		1
14	TsW(Cf)	$\overline{WAIT}$ to Clock ↓ Setup	50		
15	ThW(Cf)	$\overline{WAIT}$ from Clock ↓ Hold	0		
16	TdCr(OEf)	Clock ↑ to $\overline{OE}$ ↓ Delay		20	
17	TdCr(OEr)	Clock ↑ to $\overline{OE}$ ↑ Delay		20	
18	TdCf(IEf)	Clock ↓ to $\overline{IE}$ ↓ Delay		20	
19	TdCf(IEr)	Clock ↓ to $\overline{IE}$ ↑ Delay		35	
20	TdA(ASr)	Address Valid to $\overline{AS}$ ↑ Delay	nTcXT - 25		1
21	TdDSr(ASf)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓ Delay	nTcXT - 40		1
22	TdASr(Ax)	$\overline{AS}$ ↑ to Address not Valid Delay	nTcXT - 30		1
24	TdDSr(A)	$\overline{DS}$ ↑ to Address Active Delay	nTcXT - 40		1
25	TdAz(DSf)	Address Float to $\overline{DS}$ ↓ Delay	0		
26	TdD(DSf)	Data Valid to $\overline{DS}$ ↓ Delay	nTcXT - 20		1
27	TwDSBh	$\overline{DS}$ High Width (Burst Mode)	nTcXT - 40		1
28	TwDSBl	$\overline{DS}$ Low Width (Burst Mode)	nTcXT - 30		1

1. TcXT = XTALi Cycle Time
   CLK = 1x (1x bus clock) : n = 1
   2x (2x bus clock) : n = 2
   4x (4x bus clock) : n = 4

†Units in nanoseconds unless otherwise specified.
$V_{IH} = 2.0V$, $V_{IL} = 0.8V$, $V_{OH} = 2.0V$, $V_{OL} = 0.8V$

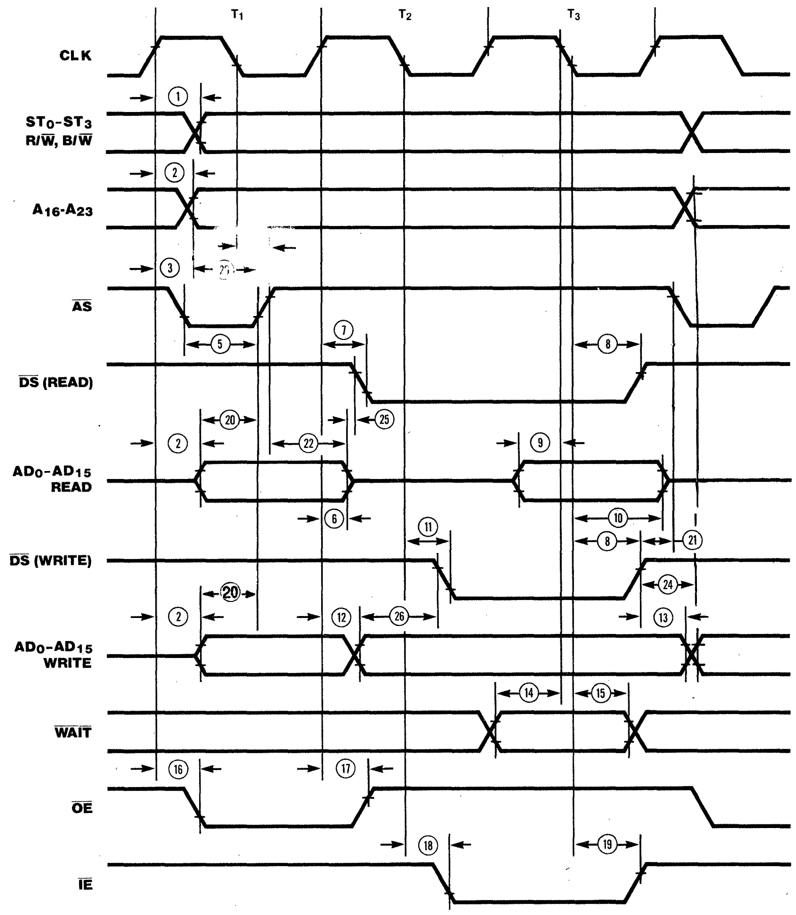


Figure 60. Z-Bus All Transactions

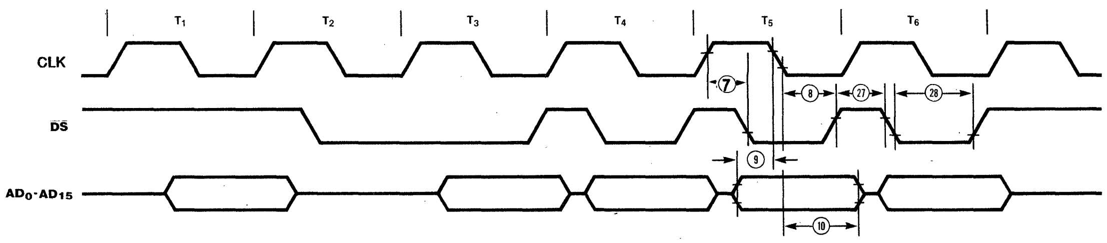


Figure 61. Z-Bus Burst Mode Timing

AC CHARACTERISTICS

Z80 Bus Timing (Refer to Figures 62 and 63)

Number	Symbol	Parameter	Min	Max	Notes†
1	TdCr(OEf)	Clock ↑ to $\overline{OE}$ ↓ Delay		20	
2	TdCr(A)	Clock ↑ to Address Valid Delay		20	
3	TdCr(ASf)	Clock ↑ to $\overline{AS}$ ↓ Delay		20	
4	TdCf(ASr)	Clock ↓ to $\overline{AS}$ ↑ Delay		20	
5	TwAS	$\overline{AS}$ Low Width	nTcXT - 20		1
6	TdCr(Az)	Clock ↑ to Address Float Delay		25	
7	TsW(Cf)	$\overline{WAIT}$ to Clock ↓ Setup	50		
8	ThW(Cf)	$\overline{WAIT}$ from Clock ↓ Hold	0		
9	TdA(ASr)	Address Valid to $\overline{AS}$ ↑ Delay	nTcXT - 25		1
10	TdASr(Ax)	$\overline{AS}$ ↑ to Address not Valid Delay	nTcXT - 30		1
11	TdCr(RDf)	Clock ↑ to $\overline{RD}$ ↓ Delay		20	
12	TdCf(RDr)	Clock ↓ to $\overline{RD}$ ↑ Delay		35	
14	TsD(Cf)	Data to Clock ↓ Setup	30		
15	ThD(Cf)	Data from Clock ↓ Hold	10		
16	TdAz(RDf)	Address Float to $\overline{RD}$ ↓ Delay	0		
19	TdCr(OEr)	Clock ↑ to $\overline{OE}$ ↑ Delay		20	
20	TdCf(IEf)	Clock ↓ to $\overline{IE}$ ↓ Delay		20	
21	TdCf(IEr)	Clock ↓ to $\overline{IE}$ ↑ Delay		35	
22	TdCr(IEr)	Clock ↑ to $\overline{IE}$ ↑ Delay		20	2
23	TdCr(RDr)	Clock ↑ to $\overline{RD}$ ↑ Delay		20	2
24	TdCf(WRf)	Clock ↓ to $\overline{WR}$ ↓ Delay		20	
25	TdCf(WRr)	Clock ↓ to $\overline{WR}$ ↑ Delay		35	
26	TdWRr(ASf)	$\overline{WR}$ ↑ to $\overline{AS}$ ↓ Delay	nTcXT - 40		1
27	TdWRr(A)	$\overline{WR}$ ↑ to Address active Delay	nTcXT - 40		1
28	TdCr(D)	Clock ↑ to Data Valid Delay		20	
29	TdWRr(Dx)	$\overline{WR}$ ↑ to Data not Valid Delay	nTcXT - 40		1
30	TdD(WRf)	Data Valid to $\overline{WR}$ ↓ Delay	nTcXT - 20		1
31	TdCf(MREQf)	Clock ↓ to $\overline{MREQ}$ ↓ Delay		20	
32	TdCf(MREQr)	Clock ↓ to $\overline{MREQ}$ ↑ Delay		35	
33	TdCr(MREQr)	Clock ↑ to $\overline{MREQ}$ ↑ Delay		20	2
34	**TdCr (IORQf)**	Clock ↑ to $\overline{IORQ}$ ↓ Delay		20	
35	TdCf(IORQr)	Clock ↓ to $\overline{IORQ}$ ↑ Delay		35	
36	**TdCf (IORQf)**	Clock ↓ to $\overline{IORQ}$ ↓ Delay		20	3
37	TdCf(M1r)	Clock ↓ to $\overline{M1}$ ↑ Delay		35	3

AC CHARACTERISTICS (Continued)

Z80 Bus Timing

Number	Symbol	Parameter	Min	Max	Notes†
38	TdCr(M1r)	Clock ↑ to $\overline{\text{M1}}$ ↑ Delay		20	2
39	TdCr(M1f)	Clock ↑ to $\overline{\text{M1}}$ ↓ Delay		20	2, 3
40	TdCf(RFSHr)	Clock ↓ to $\overline{\text{RFSH}}$ ↑ Delay		35	
41	TdCf(RFSHf)	Clock ↓ to $\overline{\text{RFSH}}$ ↓ Delay		20	
42	TdCf(HALTf)	Clock ↓ to $\overline{\text{HALT}}$ ↓ Delay.		20	

1. TcXT = XTALi Cycle Time
   CLK = 1x (1x bus clock) : n = 1
   2x (2x bus clock) : n = 2
   4x (4x bus clock) : n = 4

2. This Parameter is used for RETI (Return From Interrupt).

3. This Parameter is used for interrupt Acknowledge.

†Units in nanoseconds unless otherwise specified.
V_{IH} = 2.0V, V_{IL} = 0.8V, V_{OH} = 2.0V, V_{OL} = 0.8V

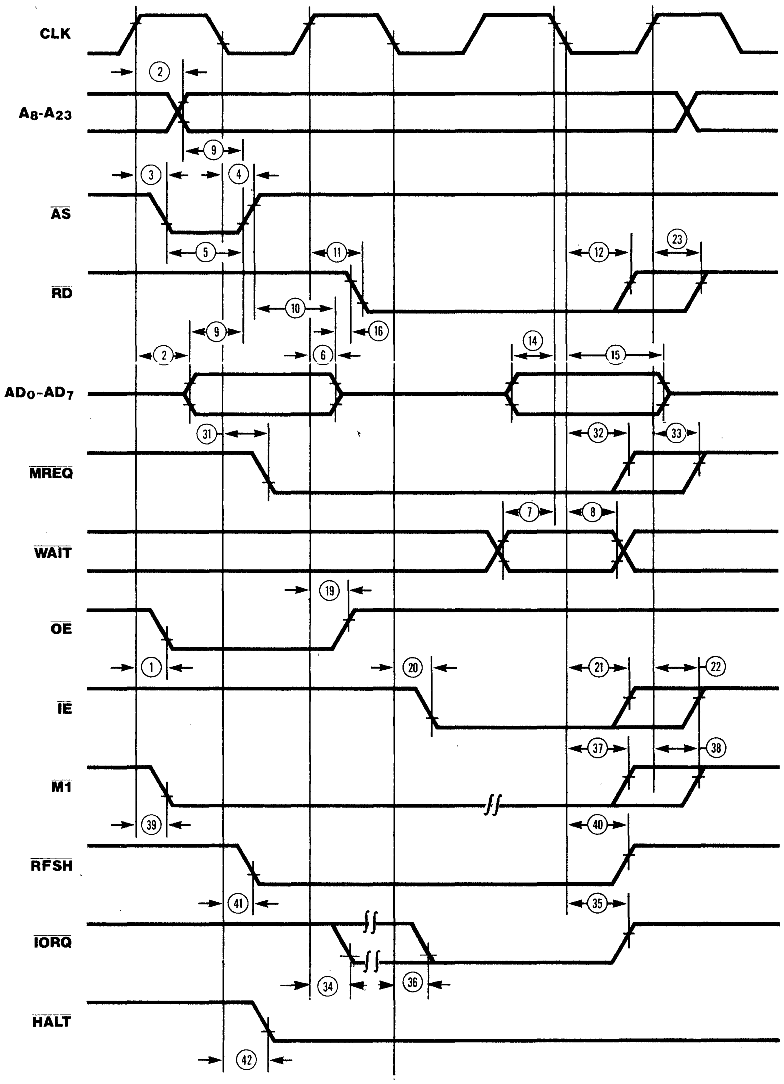


Figure 62. Z80 Bus Read Type Transactions

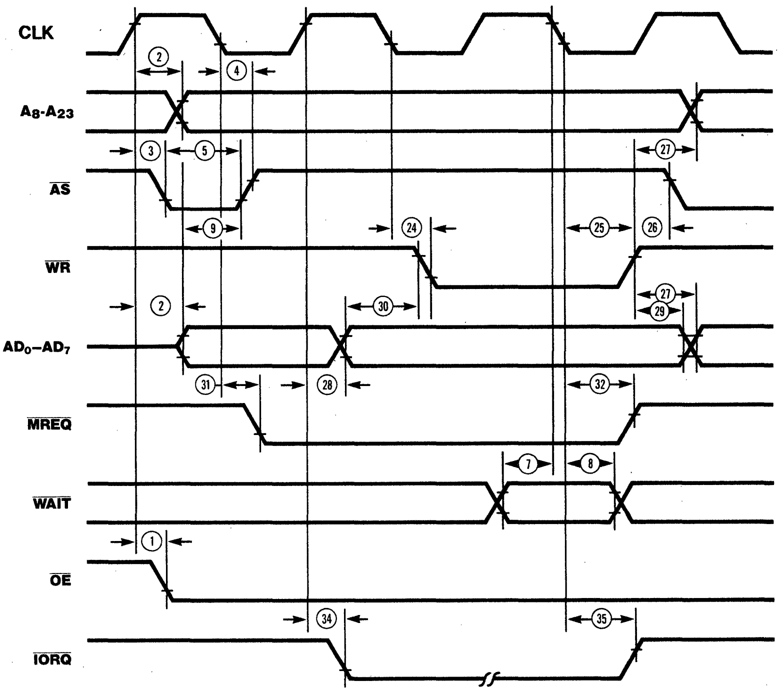


Figure 63. Z80 Bus Write Transactions

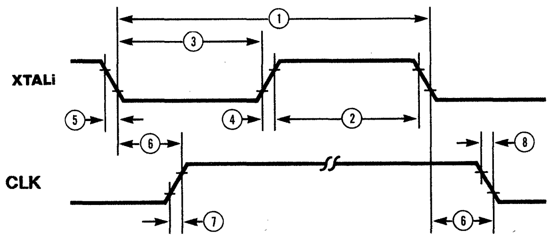


Figure 64. Z280 Clock Circuit

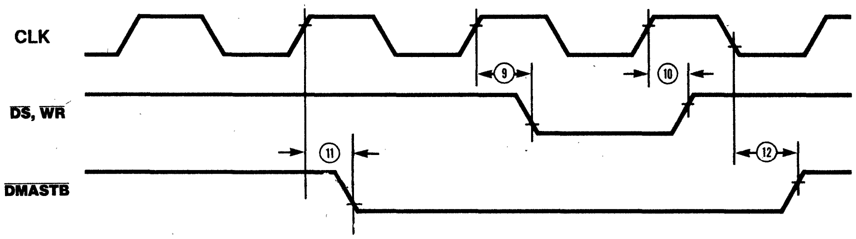


Figure 65. Flyby DMA Write to Memory

(Z-Bus: $\overline{DS}$; Z80 Bus: $\overline{WR}$)

AC CHARACTERISTICS

Z-Bus, Z80 Bus Common Signals and Peripherals Timing
(Refer to Figures 64 through 71)

Number	Symbol	Parameter	Min	Max	Notes †
1	TcXT	XTALi Cycle time	50	tbd	
2	TwXTh	XTALi High Width	15		
3	TwXTl	XTALi Low Width	15		
4	TrXT	XTALi Rise Time		10	
5	TfXT	XTALi Fall Time		10	
6	TdXTf(C)	XTAL ↓ to Clock Delay		40	
7	TrC	Clock Rise Time		12	
8	TfC	Clock Fall Time		12	
9	TdCr(CSf)	Clock ↑ to $\overline{DS}$, $\overline{RD}$, or $\overline{WR}$ ↓ Delay		20	
10	TdCr(CSr)	Clock ↑ to $\overline{DS}$, or $\overline{WR}$ ↑ Delay		20	
11	TdCr(STBf)	Clock ↑ to $\overline{DMASTB}$ ↓ Delay		20	
12	TdCf(STBr)	Clock ↓ to $\overline{DMASTB}$ ↑ Delay		35	
13	TdCr(STBr)	Clock ↑ to $\overline{DMASTB}$ ↑ Delay		20	
14	TdCf(CSr)	Clock ↓ to $\overline{DS}$ or $\overline{RD}$ ↑ Delay		35	
15	TdCf(GREQf)	Clock ↓ to $\overline{GREQ}$ ↓ Delay		35	
16	TdCf(GREQr)	Clock ↓ $\overline{GREQ}$ ↑ Delay		35	
17	TdCr(BUSACKf)	Clock ↑ to $\overline{BUSACK}$ ↓ Delay		20	
18	TdCr(BUSACKr)	Clock ↑ to $\overline{BUSACK}$ ↑ Delay		20	
19	TcCTIN	CTIN Cycle Time	10TcXT		
20	TwCTINh	CTIN High Width	4TcXT		
21	TwCTINl	CTIN Low Width	4TcXT		
22	TwCTIOh	CTIO High Width	4TcXT		1
23	TwCTIOl	CTIO Low Width	4TcXT		1
24	TdCTIN(CTIO)	CTIN to CTIO Delay	20TcXT	28TcXT	2
25	TdCf(TD)	Baud Clock ↓ to Transmit Data Delay		70	3
26	TsRD(Cr)	Receive Data to Baud Clock ↑ Setup	10		3
27	ThRD(Cr)	Receive Data from Baud Clock ↑ Hold	50		3
28	TrRESET	$\overline{RESET}$ Rise Time		10	
29	TfRESET	$\overline{RESET}$ Fall Time		10	
30	TsWAITf(RESETr)	$\overline{WAIT}$ ↓ to $\overline{RESET}$ ↑ Setup	4TcXT		4
31	ThWAITr(RESETr)	$\overline{WAIT}$ ↑ from $\overline{RESET}$ ↑ Hold	6TcXT		4
32	TsD(RESETr)	Data to $\overline{RESET}$ ↑ Setup	0		4
33	ThD(RESETr)	Data from $\overline{RESET}$ ↑ Hold	6TcXT		4
34	TrIN	Input Rise Time		20	5
35	TfIN	Input Fall Time		20	5
36	TwNMI	$\overline{NMI}$ Low Width	4TcXT		

Notes:
1. CTIO as Gate or Trigger input.
2. CTIO as Output, when CTIN causes terminal count.
3. CTIN$_1$as X1 Baud Clock input. Refer to ⑳ and ㉑ for pulse widths. Maximum frequency is ~ 2.5 MHz.
4. To program Bus Timing and Initialization Register at reset.
5. Inputs AD, $\overline{BUSREQ}$, CTIN, CTIO, $\overline{INT}$, $\overline{NMI}$, $\overline{RDY}$, RxD, $\overline{PAUSE}$, $\overline{WAIT}$

† Units in nanoseconds unless otherwise specified.
$V_{IH} = 2.0V$, $V_{IL} = 0.8V$, $V_{OH} = 2.0V$, $V_{OL} = 0.8V$

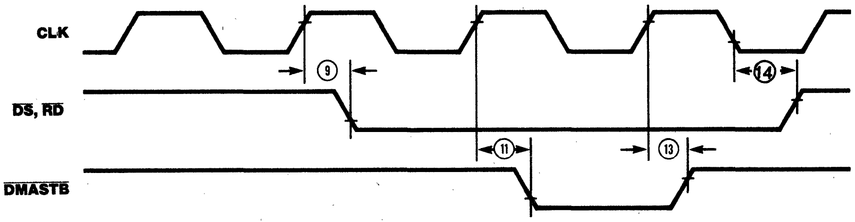


Figure 66. Flyby DMA Read from Memory (Z-Bus: $\overline{DS}$; Z80 Bus: $\overline{RD}$)

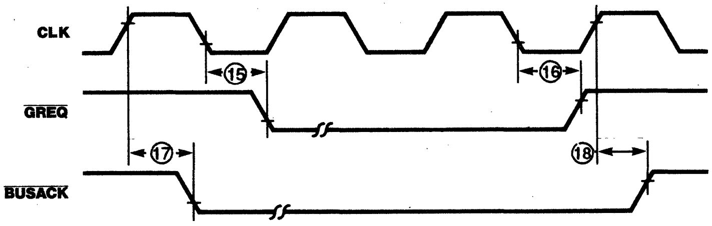


Figure 67. $\overline{GREQ}$ and $\overline{BUSACK}$ Timing

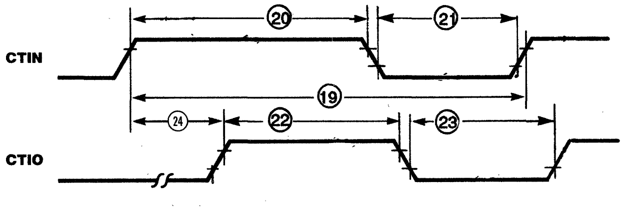


Figure 68. Counter/Timer Timing

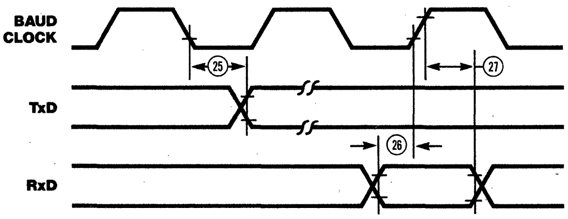


Figure 69. UART Timing

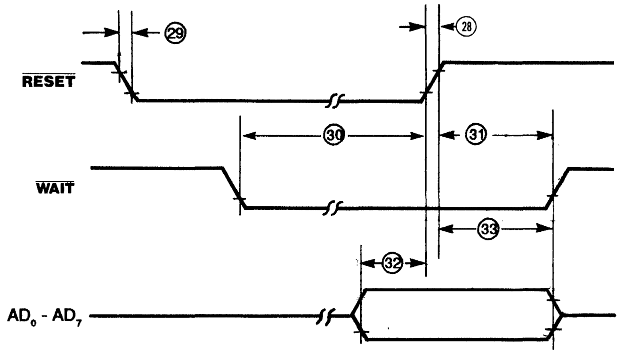


Figure 70. Reset Timing

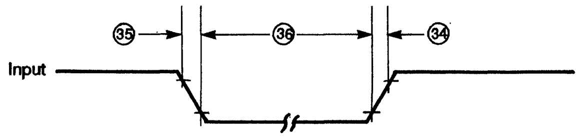


Figure 71. Inputs Timing

Z80 Family Interrupt Structure

January 1980

Zilog

Tutorial

INTRODUCTION

Interrupts provide a means of processing information on a random or asynchronous basis. The Z80 CPU and peripheral family support interrupts using a daisy-chain approach. As opposed to parallel priority resolution, the daisy chain uses an efficient, minimal-hardware method of prioritizing multiple interrupting devices. In addition, a parallel priority resolution scheme can be configured with the Z80 through the use of a priority encoder and other external hardware.

Coupled with the powerful vectored interrupt capabilities of the Z80, this approach allows the system designer great flexibility in implementing an interrupt driven system.

This document describes the Z80 CPU interrupt process and evaluates the design of the daisy-chain interrupt scheme. The reader can refer to the following documents for additional information:

Z80 Assembly Language Programming Manual	(03-0002-01)
Z80/Z80A CPU Technical Manual	(03-0029-01)
Z80/Z80A SIO Technical Manual	(03-3033-01)
Z80/Z80A PIO Technical Manual	(03-0008-01)
Microcomputer Components Data Book	(03-8032-01)

Z80 CPU INTERRUPT PROCESSING

The Z80 uses two types of interrupts: maskable ($\overline{INT}$ input) and non-maskable ($\overline{NMI}$ input). Maskable interrupts may be nested. The simplest maskable interrupt implementation does not provide for the nesting of interrupts, thereby obligating an interrupt service routine to complete its processing and return to the main program before another interrupt can be serviced. With nested interrupts, an interrupt service routine can be interrupted either by an interrupt that invokes the same routine (reentrant type) or by a higher priority interrupt that invokes a different interrupt service routine. The Z80 family components allow the user to implement a powerful interrupt-driven system utilizing these concepts.

When both types of interrupts are employed, the Z80 CPU will service them in a specific sequence. Both the $\overline{INT}$ and $\overline{NMI}$ inputs are sampled by the CPU on the rising edge of CLK in the last T state of the last Machine (M) cycle of any instruction. However, if $\overline{BUSRQ}$ is active at the same time, it will be processed before any interrupts. Figure 1 illustrates the Z80 interrupt service sequence.

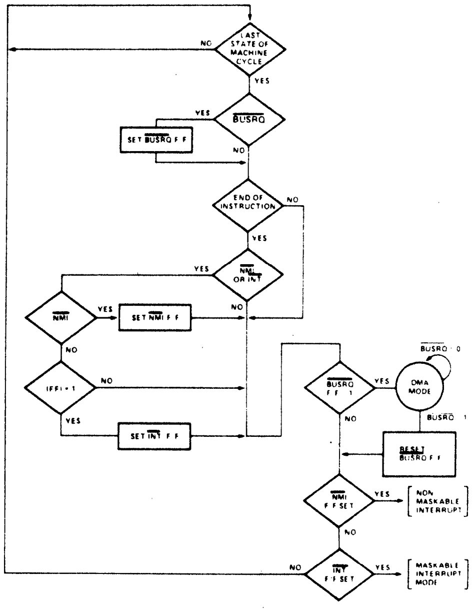


Figure 1. Z80 Flow Diagram Interrupt Sequence

Non-Maskable Interrupts

The non-maskable interrupt ($\overline{NMI}$) is different from the maskable interrupt in several respects. $\overline{NMI}$ is always enabled and cannot be disabled by the programmer. It is employed when very fast response is desired independent of the maskable interrupt status and can be used for interrupt conditions like a power fail detect. $\overline{NMI}$ is an edge-sensitive signal that has a lower priority than $\overline{BUSRQ}$ and higher priority than $\overline{INT}$. When the CPU acknowledges an occurrence of $\overline{NMI}$, the processor begins a normal opcode fetch. How-

This application note refers to products as Z80 "A", "B" etc. to specifiy the speed grade.
We are no longer using those characters for the speeds. For more details, please refer to the ordering information section.

ever, the data read from memory is ignored and instead the CPU restarts its operation from location 66H. The restart operation involves pushing the Program Counter onto the stack, jumping to location 66H, and continuing to process there. During this time, the status of the maskable interrupt condition is preserved and maskable interrupts are disabled, until either an EI instruction is executed or a RETN instruction is used to exit the $\overline{\text{NMI}}$ service routine.

The RETN instruction is discussed in detail in the Z80 CPU Technical Manual. Figure 2 shows the timing used for $\overline{\text{NMI}}$ interrupts.

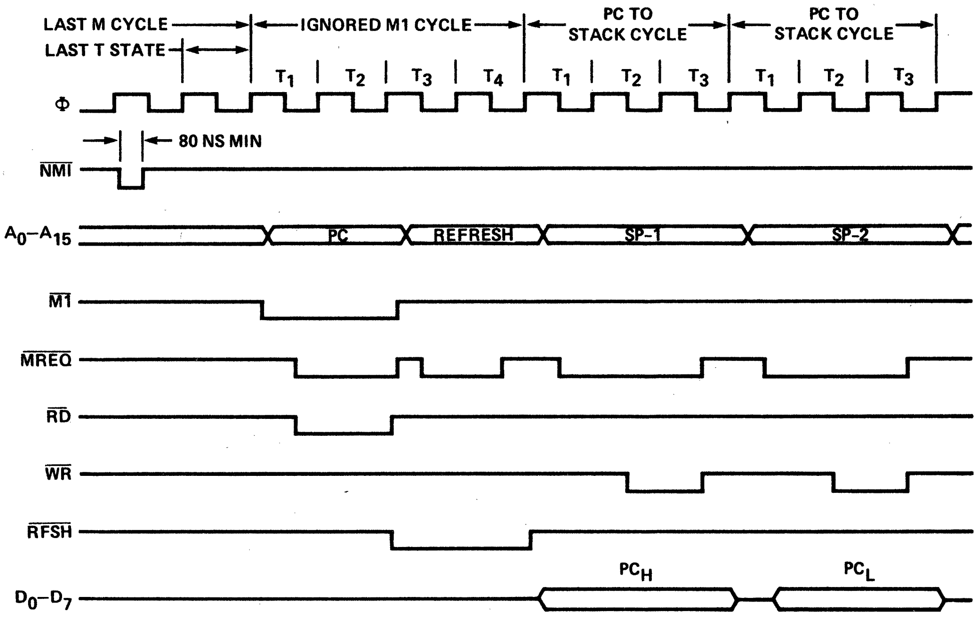


Figure 2. Non-maskable Interrupt Request Operation

Maskable Interrupts

Maskable interrupts ($\overline{\text{INT}}$) are acknowledged with a lower priority than the $\overline{\text{NMI}}$ but allow the programmer more flexibility. $\overline{\text{INT}}$ is enabled under software control by way of the EI instruction and disabled via the DI instruction. When the Z80 CPU samples $\overline{\text{INT}}$ and it is active, the processor begins an interrupt acknowledge cycle so long as $\overline{\text{BUSRQ}}$ and $\overline{\text{NMI}}$ are not active. The processor does not use an interrupt acknowledge signal but instead issues the acknowledge by executing a special $\overline{\text{M1}}$ cycle. During an interrupt acknowledge cycle, $\overline{\text{RD}}$ is inactive, $\overline{\text{IORQ}}$ is active, and two wait states are automatically added.

Since the Z80 peripheral devices have logic to interpret this special cycle with no additional external circuitry, a minimal amount of hardware is needed by the system and there is no loss in efficiency. Figure 3 shows the detailed timing for the Z80 CPU interrupt acknowledge cycle.

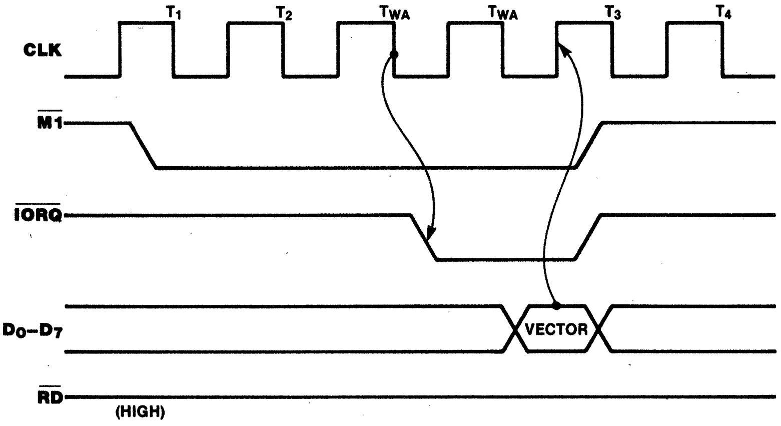


Figure 3. Interrupt Acknowledge Cycle

There are also three modes of operation for servicing maskable interrupts. These are Mode 0, Mode 1, and Mode 2. Any particular mode is selected by the programmer using the IM instruction. Figure 4 illustrates the processing sequence for each interrupt mode.

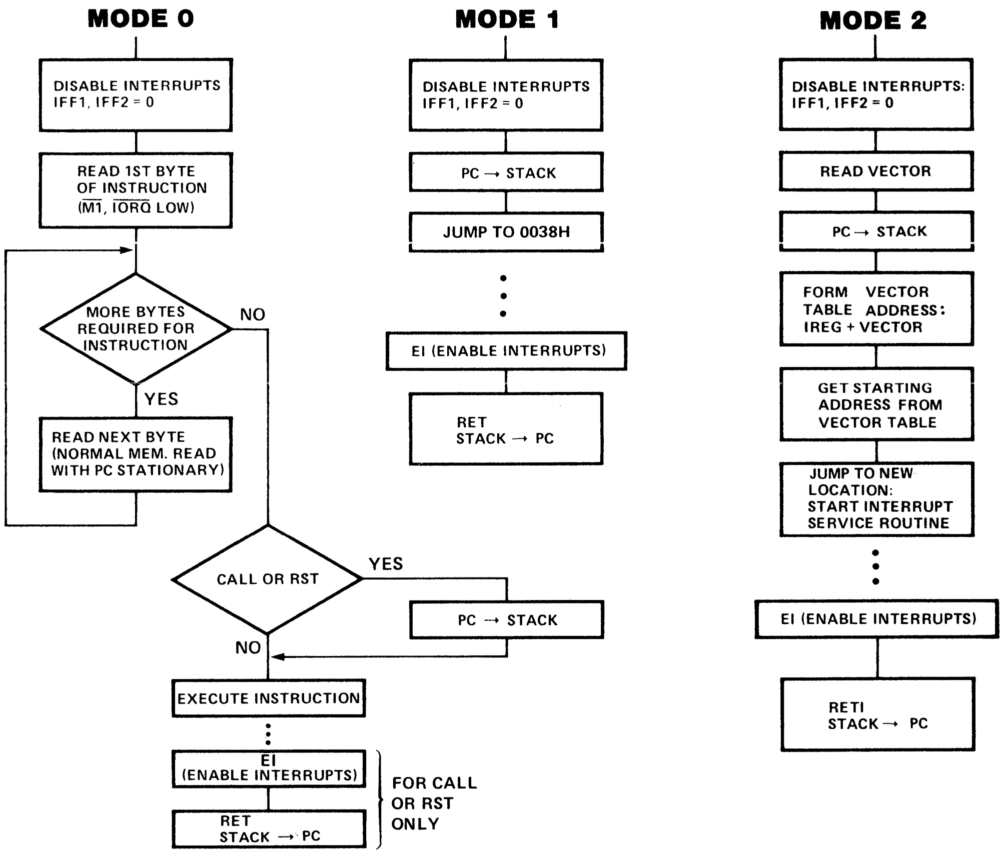

MODE 0
DISABLE INTERRUPTS IFF1, IFF2 = 0
READ 1ST BYTE OF INSTRUCTION (M1, IORQ LOW)
MORE BYTES REQUIRED FOR INSTRUCTION
NO
YES
READ NEXT BYTE (NORMAL MEM. READ WITH PC STATIONARY)
CALL OR RST
YES
PC → STACK
NO
EXECUTE INSTRUCTION
EI (ENABLE INTERRUPTS)
RET STACK → PC
FOR CALL OR RST ONLY
MODE 1
DISABLE INTERRUPTS IFF1, IFF2 = 0
PC → STACK
JUMP TO 0038H
EI (ENABLE INTERRUPTS)
RET STACK → PC
MODE 2
DISABLE INTERRUPTS: IFF1, IFF2 = 0
READ VECTOR
PC → STACK
FORM VECTOR TABLE ADDRESS: IREG + VECTOR
GET STARTING ADDRESS FROM VECTOR TABLE
JUMP TO NEW LOCATION: START INTERRUPT SERVICE ROUTINE
EI (ENABLE INTERRUPTS)
RETI STACK → PC

Figure 4. Maskable Interrupt Sequences

Maskable Interrupt Mode 0

In the maskable interrupt Mode 0 (as with the 8080 interrupt response mode), the interrupting device places an instruction on the data bus for execution by the Z80 CPU. The instruction used is normally a Restart (RST) instruction, since this is an efficient one-byte call to any of eight subroutines located in the first 64 bytes of memory. (Each subroutine is a maximum of eight bytes.) However, any instruction may be given to the Z80 CPU.

The first byte of a multibyte instruction is read during the interrupt acknowledge cycle. Subsequent bytes are read in by normal memory read cycles. The Program Counter remains at its preinterrupt state, and the user must insure that memory will not respond to these read sequences, since the instruction must come from the interrupt hardware. Timing for the additional bytes of a multibyte instruction is the same as for a single byte instruction (see NMI in Figure 2).

When an interrupt is recognized by the CPU, succeeding interrupts are automatically disabled. An EI instruction can be executed anytime after the interrupt sequence begins. The subroutine can then be interrupted, allowing nested interrupts to be used. The nesting process may proceed to any level as long as all pertinent data is saved and restored correctly.

Upon RESET, the CPU automatically sets interrupt Mode 0.

Maskable Interrupt Mode 1

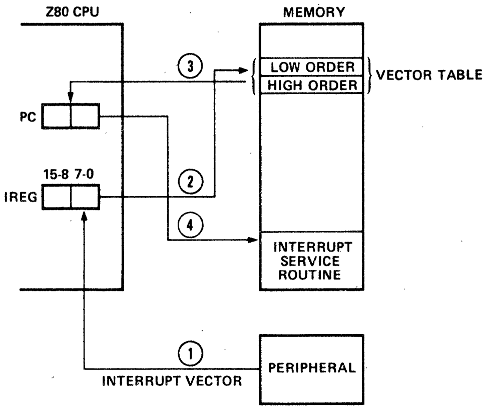
Interrupt Mode 1 provides minimally complex peripherals access to interrupt processing. It is similar to the NMI interrupt, except that the CPU automatically CALLs to location 38H instead of 66H. As with the NMI, the CPU pushes the Program Counter onto the stack automatically (Figure 2).

Maskable Interrupt Mode 2 (Vectored Interrupts)

The Z80 CPU interrupt vectoring structure allows the peripheral device to identify the starting location of the interrupt service routine.

Mode 2 is the most powerful of the three maskable interrupt modes. It allows an indirect call to any memory location by a single 8-bit vector supplied by the peripheral. In this mode, the peripheral generating the interrupt places the vector onto the data bus in response to an interrupt acknowledge. The vector then becomes the least significant eight bits of the 16-bit indirect pointer, whereas the I register in the CPU forms the most significant eight bits. This address points to an even address in the vector table which then becomes the starting address of the interrupt service routine. Interrupt processing thus starts at an arbitrary 16-bit address, allowing any location in memory to begin the service routine. Since the vector is used to identify two adjacent bytes that form a 16-bit address, the CPU requires an even starting address for the vector's low byte. Figure 5 shows the sequence of events for processing vectored interrupts.

The I register is loaded by the user from the A register. There is no restriction on its value other than its pointing to a valid memory location.



NOTES:
1. Interrupt vector generated by peripheral is read by CPU during interrupt acknowledge cycle.
2. Vector combined with I register contents form 16-bit memory address pointing to vector table.
3. Two bytes are read sequentially from vector table. These 2 bytes are read into PC.
4. Processor control is transferred to interrupt service routine and execution continues.

Figure 5. Vector Processing Sequence

Return from Maskable Interrupt

When execution of the interrupt service routine is complete, return to the main program (or another service routine) occurs differently in each mode. In Mode 0, the method of return depends on which instruction was executed by the CPU. If an RST instruction is used, a simple RET suffices. In Mode 1, the CPU treats the interrupt as a CALL instruction, so an RET is used. Mode 2, however, uses the vector information from the peripheral chip to identify the source of the recognized interrupt, and a method of resetting the peripheral's interrupt condition must be found. This is accomplished by using the RETI instruction. If Mode 2 is used by the programmer, the RETI instruction must be executed in order to utilize the daisy chain properly. Figure 6 shows the RETI instruction timing for the Z80 CPU. A more complete description of how RETI affects the peripherals is given in Chapter 3.

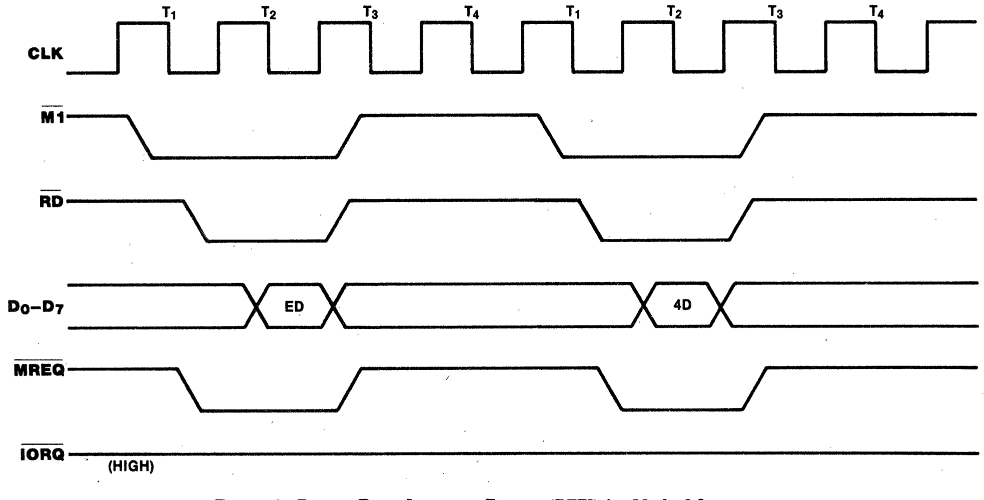


Figure 6. Return From Interrupt Timing (RETI) for Mode 2 Interrupts

Halt Exit Using Interrupts

Whenever a software halt instruction is executed, the CPU enters the Halt state by executing No-OPs (NOPs) until an interrupt or RESET is received. Each NOP consists of one $\overline{M1}$ cycle with four T states. The CPU samples the state of the $\overline{NMI}$ and $\overline{INT}$ lines on the rising edge of each T4 clock (Figure 7).

When an interrupt exists on either line, the subsequent cycle will be either a memory read operation ($\overline{NMI}$) or an interrupt acknowledge ($\overline{INT}$). The timing in Figure 7 shows a maskable interrupt causing the CPU to exit the Halt state.

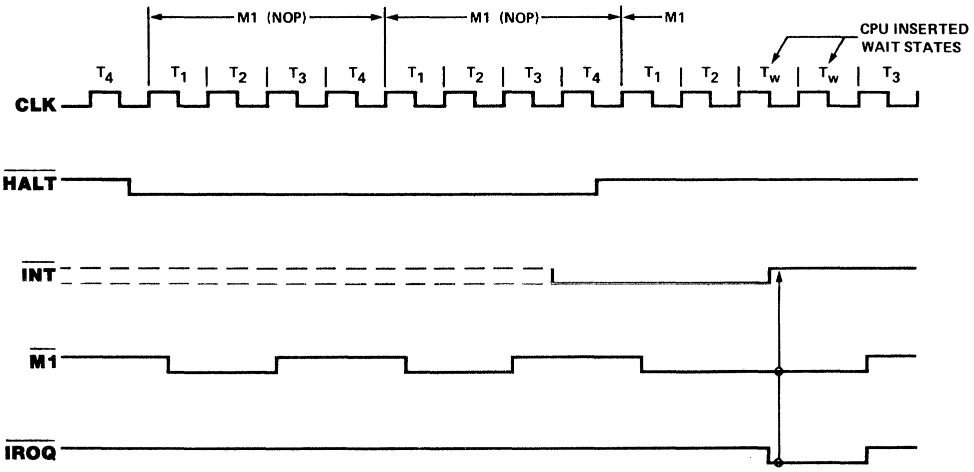


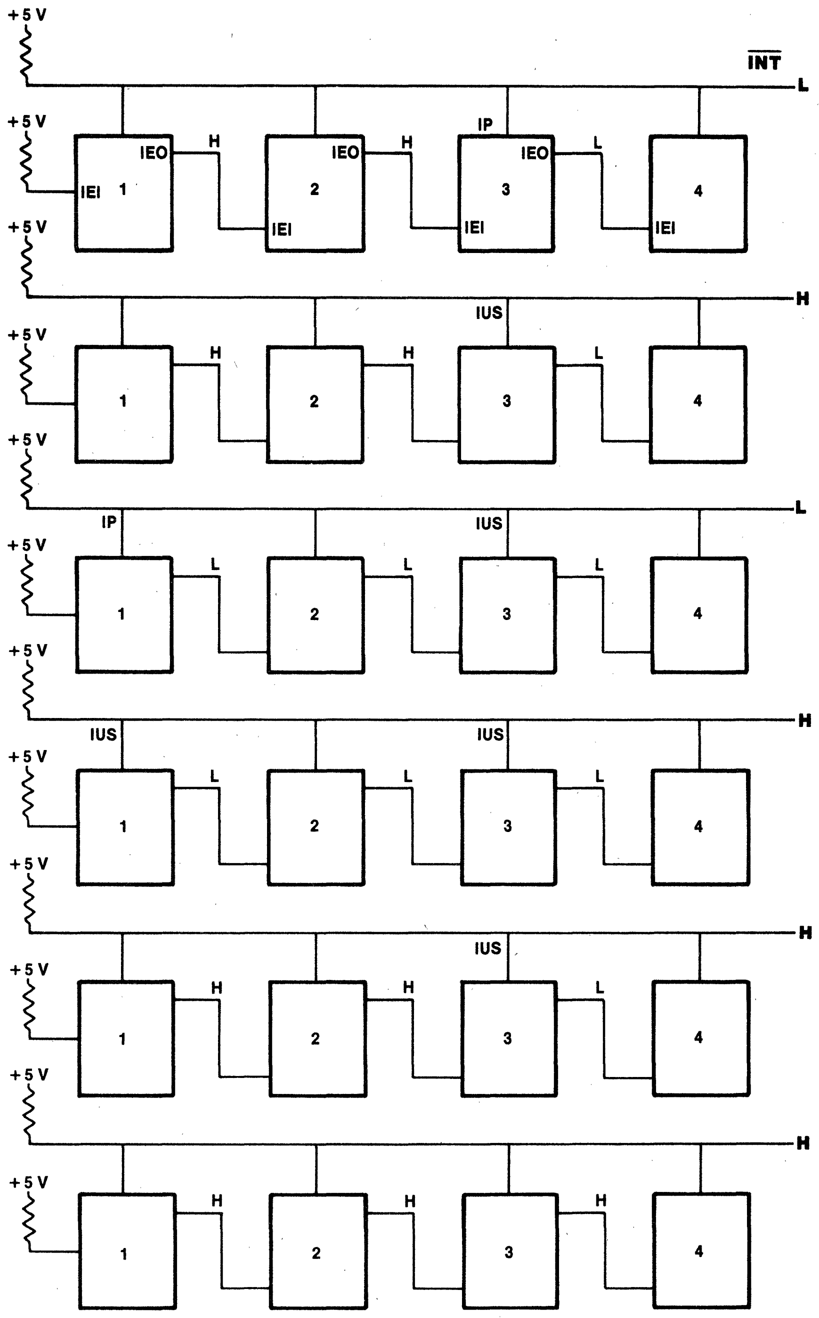
Figure 7. Exit Halt State with Maskable Interrupt

INTERRUPT PROCESSING BY Z80 PERIPHERALS

Understanding maskable interrupt processing requires a familiarity with how the Z80 peripherals respond to the CPU interrupt sequence. The Z80 family products were designed around the daisy-chain interrupt configuration, which utilizes minimal external hardware (compared to parallel contention resolution interrupt priority networks). Many devices handle interrupts via a handshake arrangement, e.g. the use of interrupt request and interrupt acknowledge signals. This is the most straightforward and probably the fastest method of implementing prioritization using more than one interrupting device. However, this method requires a separate interrupt request signal for each peripheral device and either a separate acknowledge signal for each device or a software acknowledge. Extra hardware is needed to provide contention resolution should two or more devices request an interrupt simultaneously. With the Z80 product family, however, such extra hardware is unnecessary and the software does not need to remove the interrupt request from the peripheral device. This is made possible through use of the daisy-chain priority network, which can best be visualized as a type of bucket brigade.

The Z80 peripheral products implement this daisy chain with just three extra signal lines on each chip: interrupt enable input (IEI), interrupt enable output (IEO), and interrupt request ($\overline{INT}$). The interrupt request line is an open-drain circuit that is OR wired to the $\overline{INT}$ pins of the other devices in the chain and connected to the $\overline{INT}$ pin on the Z80 CPU. This line provides the interrupt request to the CPU.

The IEI and IEO lines provide the means for establishing priority among several requesting devices. The priority of a device is determined by its position in the chain. The IEI pin of the highest priority device in the chain is connected to +5 volts. The IEO pin of the same device is connected to the IEI pin of the next highest priority device. The IEO pin of that device goes to the IEI pin of the next lower device, as shown in Figure 8, and so on to the last device in the chain, where the IEO pin is left open. When a device has an interrupt pending, it activates its $\overline{INT}$ output which requests service from the CPU and brings its IEO pin Low, thereby preventing the lower devices in the chain from responding to further interrupt operations. When the CPU acknowledges the interrupt, the requesting device removes its interrupt request ($\overline{INT}$) signal. After the interrupt processing is completed, the peripheral will reset itself with an RETI instruction, which will bring IEO High and restore the chain to its quiescent state.



NOTES:

1. Device 3 has an interrupt pending (IP set), which causes its IEO pin to go low preventing device 4 from interrupting.
2. CPU acknowledges the interrupt and device 3 has its interrupt under service (IUS set). The device's IP is then reset.
3. Device 1 requests service, suspending device 3 processing. (Assuming interrupts were reenabled.)
4. Device 1 has its interrupt under service.
5. CPU completes processing for device 1 and returns to device 3 service routine
6. CPU completes processing for device 3 and the daisy chain returns to quiescent state.

Figure 8. Z80 Peripheral Device Interrupt Processing Sequence

Interrupt Acknowledge Operation

The Z80 peripherals are acknowledged by the CPU and then serviced by an appropriate interrupt service routine. The acknowledge to the peripherals is accomplished by the CPU executing a special $\overline{M1}$ cycle in which $\overline{IORQ}$ goes active instead of $\overline{MREQ}$ and $\overline{RD}$. Whenever $\overline{M1}$ goes active, all peripheral devices are inhibited from changing their interrupt status. This allows time for IEO to propagate through the other devices in the chain before $\overline{IORQ}$ goes active. As soon as $\overline{IORQ}$ and $\overline{M1}$ go active, the peripheral device that has its IEI High and an interrupt pending gates an 8-bit vector onto the data bus. (See Figure 9 for timing details.) This 8-bit vector, which was programmed into the peripheral device, is combined with the contents of the I register in the CPU to form a 16-bit address value. During the time that $\overline{M1}$ and $\overline{IORQ}$ are active, the requesting device removes the $\overline{INT}$ signal (since the CPU has acknowledged it) and waits for a return operation. If the peripheral device has its IEI pin High and has had an interrupt acknowledged, then it completes the interrupt cycle and releases IEO (when it sees an RETI instruction [ED-4D sequence] on the data bus). This restores the chain to its normal state so that lower priority interrupts can occur.

The Z80 peripherals monitor $\overline{M1}$ and $\overline{RD}$ for the interrupt acknowledge cycle. Since $\overline{RD}$ goes active before $\overline{IORQ}$, the peripheral devices assume an interrupt acknowledge cycle if $\overline{M1}$ is active and $\overline{RD}$ is not. This reduces the time required for the internal device logic to respond to $\overline{IORQ}$ when it goes active.

Thus, a very powerful interrupt-driven system can be implemented with minimal hardware, simple software, and high efficiency using the Z80 family components.

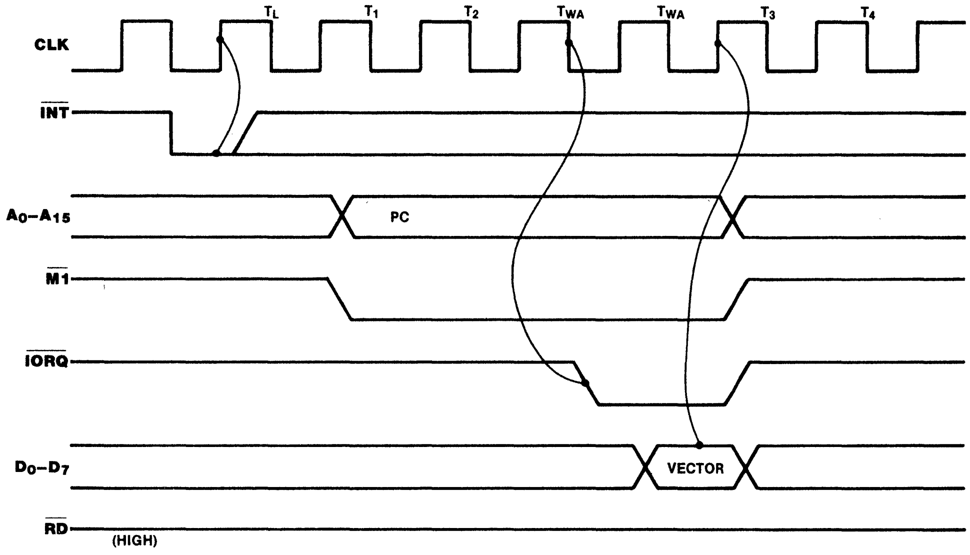


Figure 9. Peripheral Interrupt Acknowledge

Return from Interrupt Operation

When the CPU executes an RETI instruction, the device with an interrupt under service resets its interrupt condition, provided that IEI is High. All Z80 peripheral products sample the data bus for this instruction when $\overline{M1}$ goes active along with $\overline{RD}$.

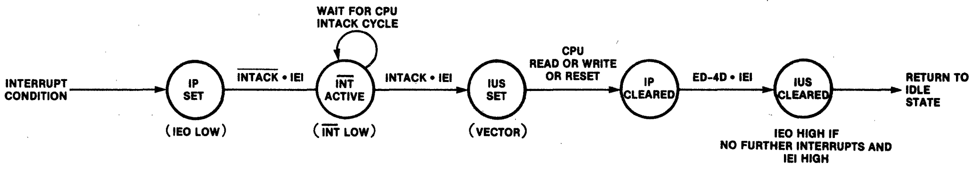
The RETI instruction decode by the peripheral device has certain characteristics that the designer should be aware of. Since a peripheral can request an interrupt (activate $\overline{INT}$ and bring IEO Low) at any time, it is possible for a device whose interrupt is currently under service to have its IEI pin Low. This is undesirable, since such a condition prevents the peripheral from resetting IUS properly. To overcome this problem, all Z80 family peripherals bring IEO High momentarily when the ED is seen during the ED-4D instruction fetch. The device whose interrupt is under service does not allow IEO to go High, but when it sees IEI High, it will reset itself when the 4D byte is fetched.

Figure 10 shows the relationship of IP and IUS to $\overline{INT}$, IEI, and IEO. IP is set by an interrupt condition on the peripheral (such as the transmit buffer becoming empty) whenever interrupts are enabled. However, IP being set will only cause $\overline{INT}$ to go active (requesting an interrupt) if IUS is not set and IEI is High. IP is not necessarily cleared by the interrupt acknowledge cycle. Some specific action must be taken within the service routine, such as filling a transmit buffer. Under these conditions, IUS becomes set and disables IEO to prevent lower priority devices in the chain from responding to an interrupt cycle. IUS is cleared when IEI is High and the peripheral decodes a valid "ED-4D" instruction. Thus,

$$IP = \overline{INTACK} * INT\_COND$$

and

$$IEO = IEI * \overline{IUS} * (\overline{IP} + "ED")$$



a) State Diagram of Z80 Peripherals During Interrupt Cycle.

IEI	IP	IUS	IEO
0	X	X	0
1	X	1	0
1	1	0	0
1	0	0	1

b) Truth Table of Daisy Chain During Idle or Interrupt Acknowlege Condition.

IEI	IP	IUS	IEO
0	X	X	0
1	X	1	0
1	X	0	1

c) Truth Table of Daisy Chain During "ED" Decode of Opcode Fetch. Note That IP Is Not Part of IEO Condition.

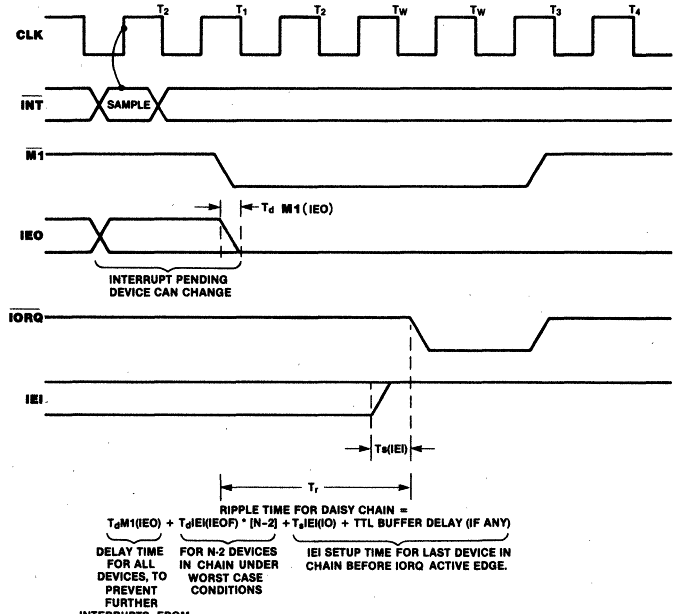
Figure 10. Z80 Peripheral Interrupt States

DAISY CHAIN DESIGN CONSIDERATIONS

There are several aspects of the Z80 family daisy chain implementation that deserve further attention.

First, since the peripheral devices must be able to monitor the data bus in order to decode the RETI instruction properly, a means of allowing them access to the data bus must be provided if buffers are used. This can be done by simply enabling the buffers from the data bus to the peripheral for all conditions except I/O read and interrupt acknowledge. Since the peripheral must assert an 8-bit vector during interrupt acknowledge, the buffers must also accommodate this.

Second, because the peripheral devices have a finite time during which IEI and IEO can stabilize within, the propagation delay of the devices must be taken into consideration. Since a device can change its interrupt status until reaching the active edge of $\overline{M1}$ during interrupt acknowledge, the time from this edge until $\overline{IORQ}$ becomes active is the time in which the daisy chain must stabilize. Figure 11 shows the timing relationships involved in this process.



CLK
T2
T1
T2
TW
TW
T3
T4
INT
SAMPLE
M1
Td M1(IEO)
IEO
INTERRUPT PENDING DEVICE CAN CHANGE
IORQ
IEI
Ts(IEI)
Tr
RIPPLE TIME FOR DAISY CHAIN =
TdM1(IEO) + TdIEI(IEOF) * [N-2] + TsIEI(IO) + TTL BUFFER DELAY (IF ANY)
DELAY TIME FOR ALL DEVICES, TO PREVENT FURTHER INTERRUPTS, FROM M1 ACTIVE EDGE.
FOR N-2 DEVICES IN CHAIN UNDER WORST CASE CONDITIONS
IEI SETUP TIME FOR LAST DEVICE IN CHAIN BEFORE IORQ ACTIVE EDGE.

Figure 11. Interrupt Acknowledge Peripheral Propagation Delay

The Z80 CPU automatically inserts two wait states during $\overline{\text{INTACK}}$, allowing a worst-case time for a chain of four devices to become settled (when using Z80A CPU and peripherals at 4MHz). If more devices are in the chain, some other means of stabilizing the chain must be provided. This can be done either by adding additional wait states to the $\overline{\text{INTACK}}$ cycle or by providing logic to the peripherals that allows faster propagation time down the chain. Figure 12 shows circuitry that provides both additional wait states and an interrupt look-ahead circuit when more than four peripheral devices are connected to the daisy chain.

When adding wait states to the Z80 CPU interrupt acknowledge cycle, care must be taken to insure that $\overline{\text{IORQ}}$ goes active at the proper time. Normally, the CPU activates $\overline{\text{IORQ}}$ on the falling edge of the clock during the first wait cycle. If external logic is used to insert additional wait states, these are appended to the two wait states already generated by the CPU. Because $\overline{\text{IORQ}}$ goes active during the first wait state and the peripherals assert their vectors when $\overline{\text{IORQ}}$ becomes active, $\overline{\text{IORQ}}$ must be inhibited until the daisy chain becomes stable. This can be done simply by adding a few gates to the wait logic (Figure 13). $\overline{\text{IORQ}}'$ is the delayed $\overline{\text{IORQ}}$ that activates the peripheral devices.

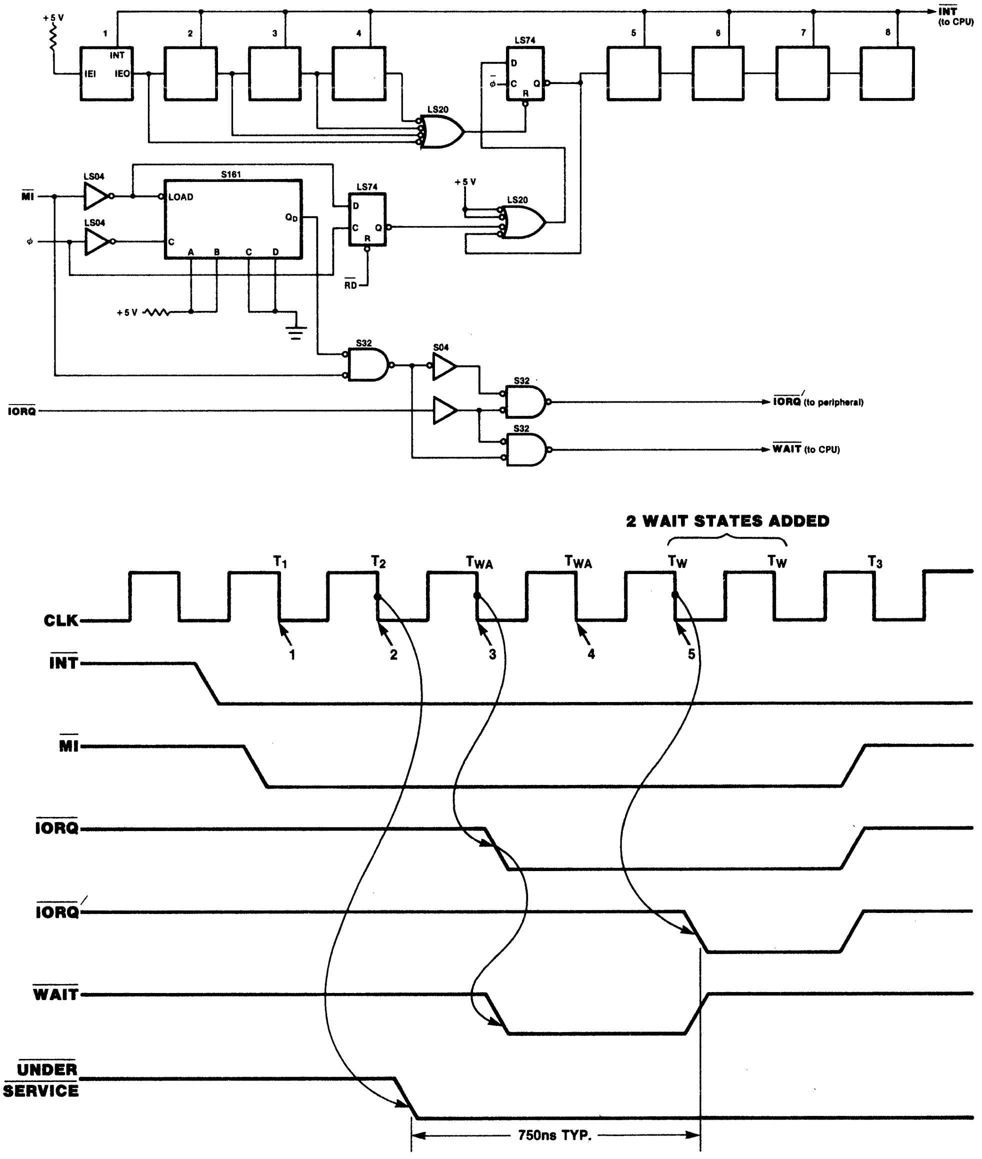


Figure 12A. Daisy Chain Look-Ahead Logic for More Than Four Peripheral Devices

The propagation delay through the peripheral devices applies during the return from interrupt condition, also. Worst-case timing involves the lowest priority device that has an interrupt under service and the highest priority device that has an interrupt pending. When the ED part of the RETI opcode is fetched, the peripheral devices must decode it, and the highest priority device must bring its IEO pin High. This IEO high signal must then propagate through the chain down to the lowest priority device before the 4D part of RETI is decoded. Figure 14 shows the timing relationships involved. This timing is not as critical as the interrupt acknowledge timing at 4 MHz, but should be considered if wait states are being added to the $\overline{\text{INTACK}}$ cycle.

If using nested interrupts with a large daisy chain, the programmer should be careful not to place the RETI opcodes too close together. Since RETI is 14 cycles long, this is generally not a problem unless a very long chain is used.

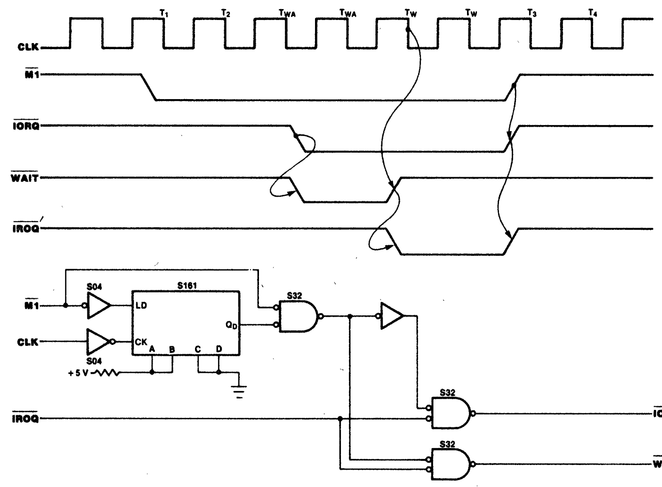
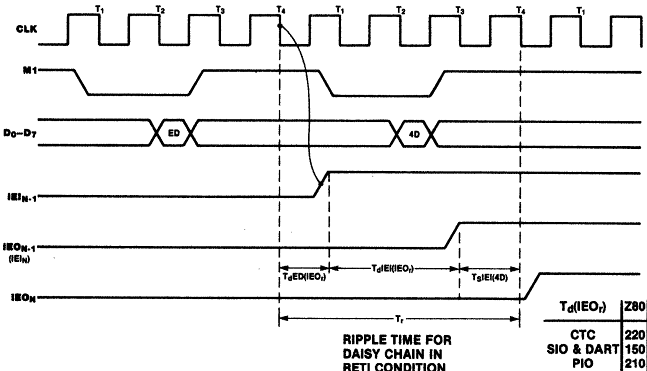


Figure 13. Wait State Logic for Interrupt Acknowledge Cycle. Counter Preset Value Should Be 5-n, Where n = # Wait State Added



$T_d(IEO_r)$	Z80	Z80A
CTC	220	160
SIO & DART	150	100
PIO	210	160
DMA	210	160

NOTES:
1. Setup time for IEI to "4D" decode ≈ 200ns (4.0 MHz).
2. Must look at IEI during ED-4D because nested interrupts allow more than 1 IUS latch to be set at one time.
3. Delay time from ED decode with IP set to IEO high ≈ 300ns (typ) 400ns (max) @2.5 MHz. This in in addition to ripple time for other devices in chain.

$$T_r \geq T_dED(IEO_r) + \underbrace{T_dIEI(IEO_r) * [N\text{-}2]}_{\text{for N-2 devices}} + T_sIEI(4D)$$

$T_dED(IEO_r)$ = Delay time from "ED" decode to IEO rise.
$T_dIEI(IEO_r)$ = Delay time from IEI high to IEO rise.
$T_sIEI(4D)$ = Setup time for IEI during "4D" decode. (For last device in chain.)

Figure 14. Daisy Chain Interrupt Timing (RETI Condition)

SPECIAL CASES OF INTERRUPTS

Interfacing Zilog 8500 series peripheral products (CIO, FIO, SCC, etc.) to the Z80 CPU is a little different from interfacing the Z80 peripherals to the CPU.

The primary difference between the Z80-type peripherals and the 8500-type peripherals is in the interrupt acknowledge circuitry. Functionally, they are the same, as can be seen in the timing diagrams of Figure 15. However, the 8500 peripherals do not sample $\overline{M1}$, $\overline{RD}$, and $\overline{IORQ}$ for the interrupt acknowledge, but have an explicit $\overline{INTACK}$ pin to signal the interrupt acknowledge. Also, since the 8500 peripherals have a software reset for the interrupt under service flip-flop, these devices do not require a special return opcode to do that operation. The user need only be concerned with the interrupt acknowledge timing when using the 8500-type peripherals.

Figure 16 shows a circuit that provides wait states for the Z80 CPU interrupt acknowledge cycle in addition to $\overline{INTACK}$ generation. The $\overline{IORQ}'$ circuitry can be omitted if no Z80 family peripheral devices are used.

In each case, the 8500 peripheral component requires $\overline{INTACK}$ and $\overline{RD}$ to be active in order for the interrupt vector to be made available to the CPU. The logic shown provides for this.

This circuitry also permits extended interrupt acknowledge times to allow for the daisy chain propagation delay and the vector response delay, so that larger chains can be implemented.

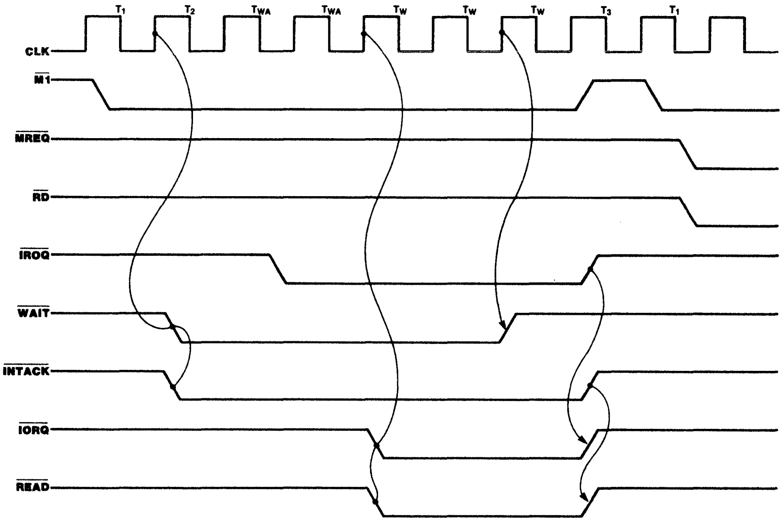


Figure 15. Timing for 8500 Peripherals During Interrupt Acknowledge.

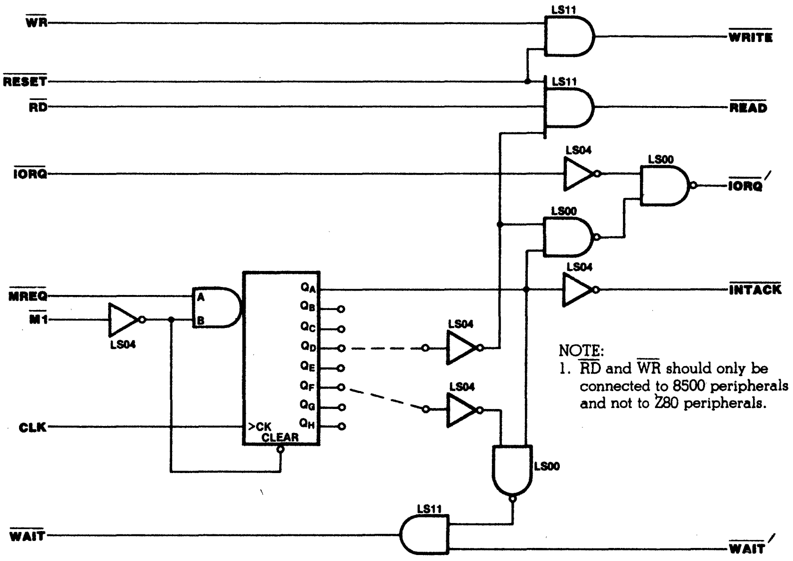


Figure 16. Interface Logic For Connecting 8500 Series Peripherals To Z80 System

Interrupt During RESET

A RESET to the Z80 CPU does several things as far as interrupts are concerned. The I register, which contains the upper eight bits of the 16-bit interrupt address value, is reset to 0, and the interrupt mode is set to Mode 0. Maskable interrupts are disabled until the programmer instructs the CPU to execute an EI instruction, just as if a DI instruction were executed. If an $\overline{\text{NMI}}$ occurs during the RESET operation, the CPU executes one instruction after the RESET condition and before acknowledging the $\overline{\text{NMI}}$. Processing then continues as usual.

Using the Z80® SIO In Asynchronous Communications

July 1980

Application Note

Zilog

SECTION 1

Introduction.

The Z80 Serial Input/Output (SIO) controller is designed for use in a wide variety of serial-to-parallel input and parallel-to-serial output applications. In this application note, only asynchronous applications are considered. The emphasis is almost completely on software implementation, with only modest reference to hardware considerations.

While reference is made only to the Z80 SIO, the entire text also applies to the Z80 DART, which is functionally identical to the Z80 SIO in asynchronous applications.

Protocol

Communication, either on an external data link or to a local peripheral, occurs in one of two basic formats: synchronous or asynchronous. In synchronous communication, a message is sent as a continuous string of characters where the string is preceded and terminated by control characters; the preceding control characters are used by the receiving device to synchronize its clock with the transmitter's clock. In asynchronous communication, which is described in this application note, there is no attempt at synchronizing the clocks on the transmitting and receiving devices. Instead, each fixed-length character (rather than character string) is preceded and terminated by "framing bits" that identify the beginning and end of the character. The time between bits within a character is approximately constant, since the clocks or "baud rates" in the transmitter and receiver are selected to be the same, but the time between characters can vary.

Thus, in asynchronous communication, each character to be transmitted is preceded by a "start" framing bit and followed by one or more "stop" framing bits. A start bit is a logical 0 and a stop bit is a logical 1. The receiver will look for a start bit, assemble the character up to the number of bits the SIO has been programmed for, and then expect to find a stop bit. The time between the start and stop bits is approximately constant, but the time between characters can vary. When one character ends, the receiving device will wait idly for the start of the next character while the transmitter continues to send stop or "marking" bits (both the stop bits and the marking bits are logical 1). Figure 1 illustrates this. A very common application of asynchronous communication is with keyboard devices, where the time between the operator's keystrokes can vary considerably.

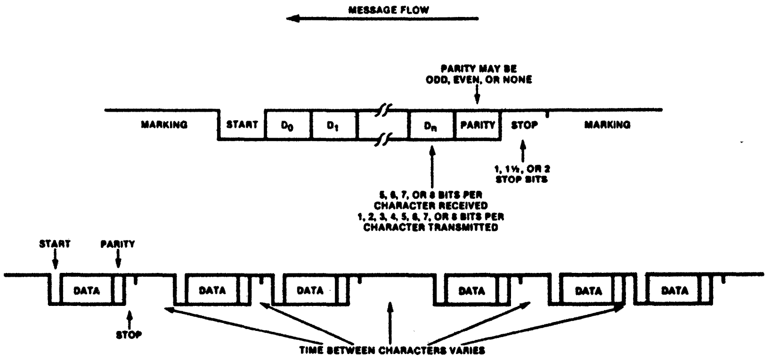

MESSAGE FLOW
PARITY MAY BE ODD, EVEN, OR NONE
MARKING
START
D0
D1
Dn
PARITY
STOP
MARKING
1, 1½, OR 2 STOP BITS
5, 6, 7, OR 8 BITS PER CHARACTER RECEIVED 1, 2, 3, 4, 5, 6, 7, OR 8 BITS PER CHARACTER TRANSMITTED
START
PARITY
DATA
STOP
TIME BETWEEN CHARACTERS VARIES

Figure 1. Asynchronous Data Format

This application note refers to products as Z80 "A", "B" etc. to specifiy the speed grade. We are no longer using those characters for the speeds. For more details, please refer to the ordering information section.

Protocol (Continued)

If the transmitter's clock is slightly faster than the receiver's clock, the transmitter can be programmed to send additional stop bits, which will allow the receiver to catch up. If the receiver runs slightly faster than the transmitter, then the receiver will see somewhat larger gaps between characters than the transmitter does, but the characters will normally still be received properly. This tolerance of minor frequency deviations is an important advantage of using asynchronous I/O. Note however that errors, called "framing errors," can still occur if the transmitter and receiver differ substantially in speed, since data bits may then be erroneously treated as start or stop bits.

Modes

The SIO may be used in one of three modes: Polled, Interrupt, or Block Transfer, depending on the capabilities of the CPU. In Polled mode the CPU reads a status register in the SIO periodically to determine if a data character has been received or is ready for transmission. When the SIO is ready, the CPU handles the transfer within its main program.

In Interrupt mode, which is far more common, the SIO informs the CPU via an interrupt signal that a single-character transfer is required. To accomplish this, the CPU must be able to check for the presence of interrupt signals (or "interrupt requests") at the end of most instruction cycles. When the CPU detects an interrupt it branches to an interrupt service routine which handles the single-character transfer. The beginning memory address of this interrupt service routine can be derived, in part, from an "interrupt vector" (8-bit byte) supplied by the SIO during the interrupt acknowledge cycle.

In Block Transfer mode, the SIO is used in conjunction with a DMA (direct memory access) controller or with the Z80 or Z8000 CPU block transfer instructions for very fast transfers. The SIO interrupts the CPU or DMA only when the first character of a message becomes available, and thereafter the SIO uses only its Wait/Ready output pin to signal its readiness for subsequent character transfers. Due to the faster transfer speeds achievable, Block Transfer mode is most commonly used in synchronous communication and only rarely in asynchronous formats. It is therefore not treated with specific examples in this application note.

Since Polled mode requires CPU overhead regardless of whether or not an I/O device desires attention, Interrupt mode is usually the preferred alternative when it is supported by the CPU. Note that the choice of Polled or Interrupt mode is independent of the choice of synchronous or asynchronous I/O. This latter choice is usually determined by the type of device to which the system is communicating.

SIO Configurations

The SIO comes in four different 40-pin configurations: SIO/0, SIO/1, SIO/2, and SIO/9. The first three of these support two independent full-duplex channels, each with separate control and status registers used by the CPU to write control bytes and read status bytes. The SIO/9 differs from the first three versions in that it supports only one full-duplex channel. The product specifications for these versions explain this in full.

There are 41 different signals needed for complete two-channel implementation in the SIO/0, SIO/1, and SIO/2, but only 40 pins are available. Therefore, the versions differ by either omitting one signal or bonding two signals together. The dual-channel asynchronous-only Z80 DART has the same pin configuration as the SIO/0.

SIO-CPU Hardware Interfacing

The serial-to-parallel and parallel-to-serial conversions required for serial I/O are performed automatically by the SIO. The device is connected to a CPU by an 8-bit bidirectional data path, plus interrupt and I/O control signals.

The SIO was designed to interface easily to a Z80 CPU, as shown in Figure 2. Other microprocessors require a small amount of external logic to generate the necessary interface signals.

The SIO provides a sophisticated vectored-interrupt facility to signal events that require CPU intervention. The interrupt structure is based on the Z80 peripheral daisy chain. Non-Z80 microprocessors that are unable to utilize external vectored interrupts require some additional external logic to utilize efficiently this interrupt facility. Some non-Z80 system designs do not utilize the vectored interrupt structure of the SIO at all. Instead, these require the CPU to poll the SIO's status through the data bus or to use non-vectored SIO interrupts.

Microprocessors such as the 8080 and 6800 need some signal translation logic to generate SIO read/write and clock timing. CPU signals which synchronize a peripheral device read or write operation are gated to form the proper I/O signals for the SIO. The SIO is selected by some processor-dependent function of the address bus in a memory or I/O addressing space.

Reference Material

In the next section we begin with a discussion of features common to all forms of asynchronous I/O. This is followed by discussions of polled asynchronous I/O and interrupt asynchronous I/O. Next is a series of frequently asked questions about the SIO when used in asynchronous applications. Finally, an example of a simple interrupt-driven asynchronous application is given and discussed in detail. For a complete understanding of the material covered, the following publications are needed:

- *Z80 SIO Product Specification or Z80 DART Product Specification*
- *Z80 SIO Technical Manual*
- *Z80 Family Program Interrupt Structure*
- *Z80 CPU Technical Manual*
- *Z80 Assembly Language Programming Manual*

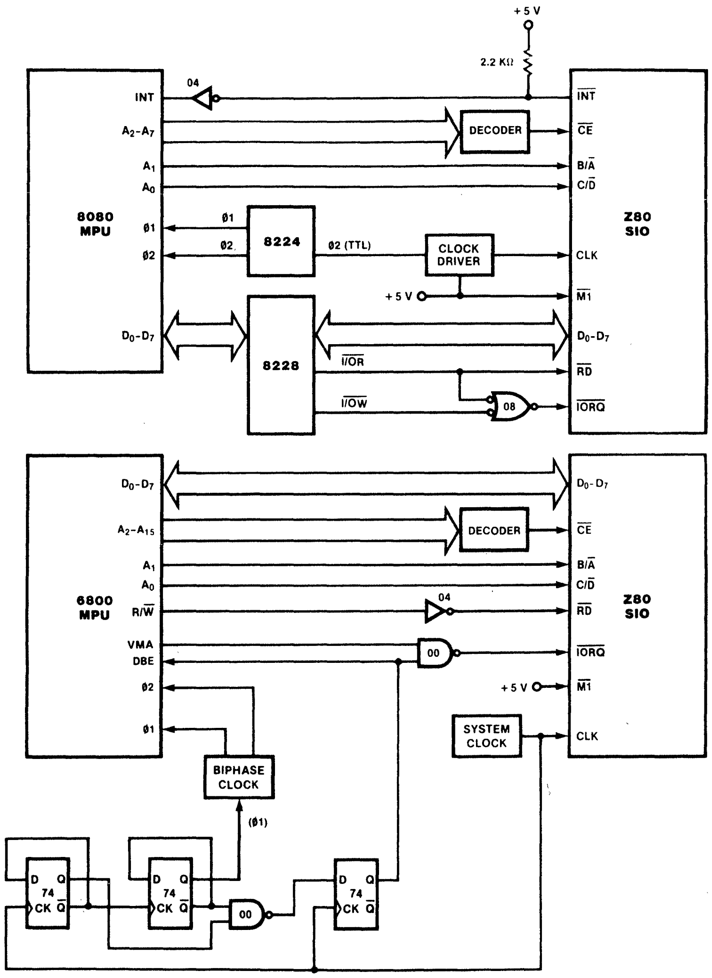


Figure 2. SIO Hardware Interfacing

SECTION 2

Operational Considerations.

All of the SIO options to be discussed here are software controllable and are set by the CPU. Thus, use of the SIO begins with an initialization phase where the various options are set by writing control bytes. These options are established separately for each of the two channels supported by the SIO if both channels are used. Before giving an overview of how initialization is done, we will describe some of the basic characteristics of SIO operations that are common to both the Polled and Interrupt-driven modes.

Addressing the SIO

The CPU must have a means to identify any specific I/O device, including any attached SIO. In a Z80 CPU environment, this is done by using the lower 8 bits of the address bus (A_0-A_7). Typically, the A_1 bit is wired to the SIO's $B/\overline{A}$ input pin for selecting access to Channel A or Channel B, and the A_0 bit is wired to the SIO's $C/\overline{D}$ input pin for selecting the use of the data bus as an avenue for transferring control/status information (C) or actual data messages (D). The remaining bits of the address bus, A_2-A_7, contain a port address that uniquely identifies the SIO device. These latter six lines are usually wired to an external decoding chip which activates that SIO's Chip Enable ($\overline{CE}$) input pin when its address appears on A_2-A_7 of the address bus.

The bar notation drawn above the names of certain signal lines, such as $B/\overline{A}$ and $C/\overline{D}$, refer to signals which are interpreted as active when their logic sense—and voltage level—is Low. For example, the $B/\overline{A}$ pin specifies Channel B of the SIO when it carries a logic 1 (high voltage) and it specifies Channel A when it carries a logic 0 (low voltage).

Asynchronous Format Operations

Bits per Character. The SIO can receive or transmit 5, 6, 7, or 8 bits per character. This can be different for transmission and reception, and different for each channel. ASCII characters, for example, are usually transmitted as 7 bits. The SIO can in fact transmit fewer than 5 bits per character when set to the 5-bit mode; this is discussed further in the section entitled "Questions and Answers."

Parity. A parity bit is an additional bit added to a character for error checking. The parity bit is set to 0 or 1 in order to make the total number of 1s in the character (including parity bit) even or odd, depending on whether even or odd parity is selected. The SIO can be set either to add an optional parity bit to the "bits per character" described above, or not to add such a bit. When a parity bit is included, either even or odd parity can be chosen. This selection can be made independently for each channel.

Start and Stop Bits. There are two types of framing bits for each character: start and stop. When transmitting asynchronously, the SIO automatically inserts one start bit (logic 0) at the beginning of each character transmitted. The SIO can be programmed to set the number of stop bits inserted at the end of each character to either 1, 1½, or 2. The receiver always checks for 1 stop bit. Stop bits refer to the length of time that the stop value, a logic 1, will be transmitted; thus 1½ stop bits means that a 1 will be transmitted for the length of clock time that 1½ bits would normally take up. A logic 1 level that continues after the specified number of stop bits is called a "marking" condition or "mark bits."

CPU-SIO Character Transfers

The SIO always passes 8-bit bytes to the CPU for each character received, no matter how many "bits per character" are specified in the SIO initialization phase. If the number of "bits per character" is less than eight, parity and/or stop bits will be included in the byte sent to the CPU. The received character starts with the least-significant bit (D_0) and continues to the most-significant bit; it is immediately followed by the parity bit (if parity is enabled) and by the stop bit, which will be logic 1 unless there is a framing error. The remainder of the byte, if space is still available, is filled with logic 1s (marking). If the "bits per character" is eight, then the byte sent to the CPU will contain only the data bits. In all cases, the start bit is stripped off by the SIO and is not transmitted to the CPU.

Clock Divider

The SIO has five input pins for clock signals. One of these inputs (CLK) is used only for internal timing and does not affect transmission or reception rates. The other four clock inputs ($\overline{RxCA}$, $\overline{TxCA}$, $\overline{RxCB}$, and $\overline{TxCB}$) are used for timing the reception and transmission rates in Channels A and B. Only these last four are involved in "clock dividing." A clock divider within the SIO can be programmed to cause reception/transmission clocking at the actual input clock rate or at 1/16, 1/32, or 1/64 of the input clock rate. The receiver and transmitter clock divisions within a given channel must be the same, although their input clock rates can be different. The x1 clock rate can be used only if the transitions of the Receive clock are synchronized to occur during valid data bit times.

Auto Enables

The SIO has an Auto Enables feature that allows automatic SIO response and telephone answering. When Auto Enables is set for a particular channel, a transition to logical 0 (Low input level) on the respective Data Carrier Detect ($\overline{DCD}$) input will enable reception, and a transition to logical 0 on the respective Clear To Send ($\overline{CTS}$) input will enable transmission. This is described below under the heading "Modem Control."

Special Receive Conditions

There are three error conditions that can occur when the SIO is receiving data. Each of these will cause a status bit to be set, and if operating in Interrupt mode, the SIO can optionally be programmed to interrupt the CPU on such an error. The error conditions are called "special receive conditions" and they include:

- **Framing error.** If a stop bit is not detected in its correct location after the parity bit (if used) or after the most-significant data bit (if parity is not used), a framing error will result. The start bit preceding the character's data bits is not considered in determining a framing error, although character assembly will not begin until a start bit is detected.
- **Parity error.** If parity bits are attached by the external I/O device and checked by the SIO while receiving characters, a parity error will occur whenever the number of logic 1 data bits in the character (including the parity bit) does not correspond to the odd/even setting of the parity-checking function.
- **Receiver overrun error.** SIO buffers can hold up to three characters. If a character is received when the buffers are full (i.e., characters have not been read by the CPU), an SIO receiver overrun error will result. In this case, the most recently received character overwrites the next most recently received character.

Modem Control

Five signal lines on the SIO are provided for optional modem control, although these lines can also be used for other general-purpose control functions. They are:

$\overline{RTS}$ (Request To Send). An output from the SIO to tell its modem that the SIO is ready to transmit data.

$\overline{DTR}$ (Data Terminal Ready). An output from the SIO to tell its modem that the SIO is ready to receive data.

$\overline{CTS}$ (Clear To Send). An input to the SIO from its modem that enables SIO transmission if the Auto Enables function is used.

$\overline{DCD}$ (Data Carrier Detect). An input to the SIO from its modem that enables SIO reception if the Auto Enables function is used.

$\overline{SYNC}$ (Synchronization). A spare input to the SIO in asynchronous applications. This input may be used for the Ring Indicator function, if necessary, or for general-purpose inputs.

In most applications of asynchronous I/O that use modems, the $\overline{RTS}$ and $\overline{DTR}$ control lines and the Auto Enables function are activated during the initialization sequence, and they are left active until no further I/O is expected. This causes the SIO to tell its modem continuously that the SIO is ready to transmit and receive data, and it allows the modem to enable automatically the SIO's transmission and reception of data. Figure 3 illustrates this.

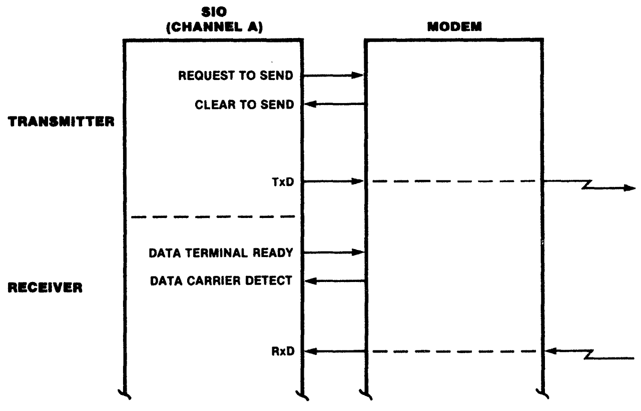


Figure 3. Modem Control (Single Channel)

External/Status Interrupts

A change in the status of certain external inputs to the SIO will cause status bits in the SIO to be set. In the Polled Mode, these status bits can be read by the CPU. In the Interrupt mode, the SIO can also be programmed to interrupt the CPU when the change occurs. There are three such "external/status" conditions that can cause these events:

- **DCD.** Reflects the value of the $\overline{DCD}$ input.
- **CTS.** Reflects the value of the $\overline{CTS}$ input.
- **Break.** A series of logic 0 or "spacing" bits.

Note that the DCD and CTS status bits are the inverse of the SIO lines, i.e., the DCD bit will be 1 when the $\overline{DCD}$ line is Low.

Any transition in any direction (i.e., to logic 0 or to logic 1) on any of these inputs to the SIO will cause the related status bit to be latched and (optionally) cause an interrupt. The SIO status bits are latched after a transition on any one of them. The status must be reset (using an SIO command) before new transitions can be reflected in the status bits.

Initialization

The SIO contains eight write registers for Channel B (WR0-WR7) and seven write registers for Channel A (all except write register WR2). These are described fully in the *Z80 SIO Technical Manual* and are summarized in Appendix B. The registers are programmed separately for each channel to configure the functional personality of the channel. WR2 exists only in the Channel B register set and contains the interrupt vector for both channels. Bits in each register are named D_7 (most significant) through D_0. With the exception of WR0, programming the write registers requires two bytes: the first byte is to WR0 and contains pointer bits for selection of one of the other registers; the second byte is written to the register selected. WR0 is a special case in that all of the basic commands can be written to it with a single byte.

There are also three read registers, named RR0 through RR2, from which status results of operations can be read by the CPU (see Appendix B). Both channels have a set of read registers, but register RR2 exists only in Channel B.

Let us now look at the typical sequence of write registers that are loaded to initialize the SIO for either Polled or Interrupt-driven asynchronous I/O. Figure 4 illustrates the sequence. Except for step E, this loading is done for each channel when both are used. Steps E and F are described further in the section on "Interrupt-Driven Environments."

Registers WR6 and WR7 are not used in asynchronous I/O. They apply only to synchronous communication.

The related publications on the SIO should be referred to at this point. They will be necessary in following the discussion of functions. In particular, the following material should be reviewed:

Z80 SIO Technical Manual, pages 9-12 ("Asynchronous Operation")

Z80 SIO Technical Manual, pages 29-37 ("Z80 SIO Programming")

A. Load WR0. This is done to reset the SIO.

B. Load WR4. This specifies the clock divider, number of stop bits, and parity selection. Since register WR4 establishes the general form of I/O for which the SIO is to be used, it is best to set WR4 values first.

C. Load WR3. This specifies the number of receive bits per character, Auto Enable selection, and turns on the receiver enabling bit.

D. Load WR5. This specifies the number of transmit bits per character, turns off the bit that transmits the Break signal, turns on the bits indicating Data Terminal Ready and Request To Send, and turns on the transmitter enabling bit.

E. Load WR2. (Interrupt mode only and Channel B only.) This specifies the interrupt vector.

F. Load WR1. (Interrupt mode only.) This specifies various interrupt-handling options that will be explained later.

NOTES:
Steps A through F are performed in sequence.
*Channel B only.
†Interrupt mode only. Polling mode begins I/O after step D.

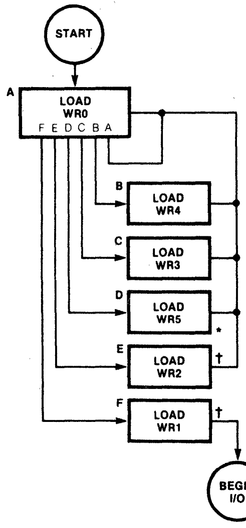


Figure 4. Typical Initialization Sequence (One Channel)

SECTION 3

Polled Environments.

In a typical Polled environment, the SIO is initialized and then periodically checked for completion of an I/O operation. Of course, if the checking is not frequent enough, received characters may be lost or the transmitter may be operated at a slower data rate than that of which it is capable. Initialization for Polled I/O follows the general outline described in the last section. We now give an overview of routines necessary for the CPU to check whether a character has been received by the SIO or whether the SIO is ready to transmit a character.

Character Reception

To check whether a character has been received, and to obtain a received character if one is available, the sequence illustrated in Figure 5 should be followed after the SIO is initialized. We assume that reception was enabled during initialization; if it was not, the Rx Enable bit in register WR3 must be turned on before reception can occur. This must be done for each channel to be checked.

Bit D_0 of register RR0 is set to 1 by the SIO if there is at least one character available to be received. The SIO contains a three-character input buffer for each channel, so more than one character may be available to be received. Removing the last available character from the read buffer for a particular channel turns off bit D_0.

If bit D_0 of register RR0 is 0, then no character is available to be received. In this case it is recommended that checks be made of bit D_7 to determine if a Break sequence (null character plus a framing error) has been received. If so, a Reset External/Status Interrupts command should be given; this will set the External/Status bits in register RR0 to the values of the signals currently being received. Thus, if the Break sequence has terminated, the next check of bit D_7 will so indicate. It may also be desirable to check bit 3 of register RR0 which reports the value of the Data Carrier Detect (DCD) bit.

In any case, if bit D_0 of register RR0 is 0, polled receive processing terminates with no character to receive. Depending on the facilities of the associated CPU, this step may be repeated until a character is available (or possibly a time-out occurs), or the CPU may return to other tasks and repeat this process later.

If bit D_0 of register RR0 is 1, then at least one character is available to be read. In this case, the value of register RR1 should first be read and stored to avoid losing any error information (the manner in which it is read is explained later). The character in the data register is then read. Note that the character must be read to clear the buffer even if there is an error found.

Finally, it is necessary to check the value stored from register RR1 to determine if the character received was valid. Up to three bits need to be checked: bit 6 is set to 1 for a framing error, bit 5 is set to 1 for a receiver overrun error (which occurs when the receive buffers are overwritten, i.e., no character has been removed and more than three characters have been received), and bit 4 is set to 1 for a parity error (if parity is enabled at initialization time). In case of a receiver overrun or parity error, an Error Reset command must be given to reset the bits.

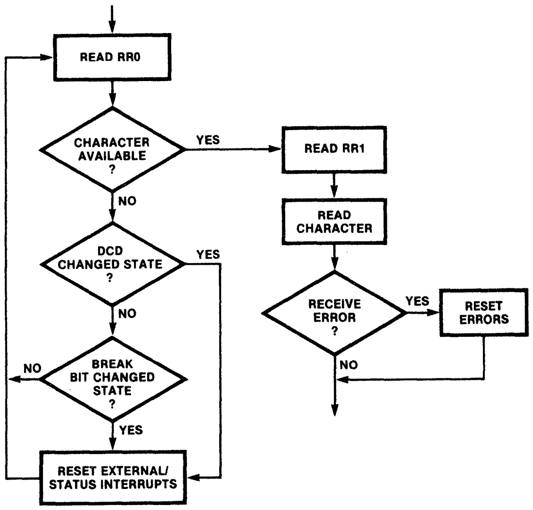


Figure 5. Polled Receive Routine

Character Transmission

To check that an initialized SIO is ready to transmit a character on a channel, and if so to transmit the character, the steps illustrated in Figure 6 should be followed. We assume that the Request To Send (RTS) bit in WR5, if required by the external receiving device, and the Transmit (Tx) Enable bit were set at initialization.

Depending on the external receiving device, the following bits in register RR0 should be checked: bit 3 (DCD), to determine if a data carrier has been detected; bit 5 (CTS), to determine if the device has signalled that it is clear to send; and bit 7 (Break), to determine if a Break sequence has been received. If any of these situations have occurred, the bits in register RR0 must be reset by sending the Reset External/Status Interrupts command, and the transmit sequence must be started again.

Next, bit 2 of register RR0 is checked. If this bit is 0, then the transmit buffer is not empty and a new character cannot yet be transmitted. Depending on the capabilities of the CPU, this is repeated until a character can be transmitted (or a timeout occurs), or the CPU may return to other tasks and start again later.

If bit 2 of register RR0 is 1, then the transmit buffer is empty and the CPU may pass the character to be transmitted to the SIO, completing the transmit processing. On the Z80 CPU, this is done with an OUT instruction to the SIO data port.

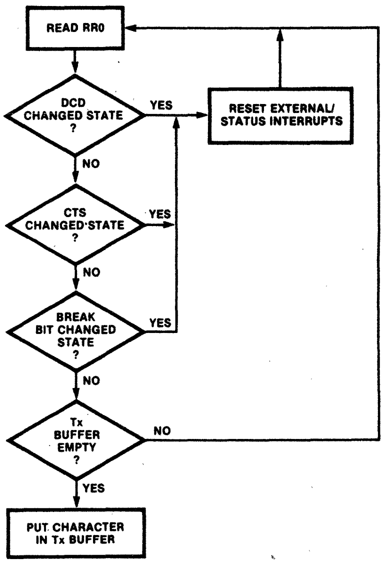

READ RR0
DCD CHANGED STATE ?
YES
RESET EXTERNAL/ STATUS INTERRUPTS
NO
CTS CHANGED STATE ?
YES
NO
BREAK BIT CHANGED STATE ?
YES
NO
Tx BUFFER EMPTY ?
NO
YES
PUT CHARACTER IN Tx BUFFER

Figure 6. Polled Transmit

Assumptions for an Example

Now let us consider some examples in more detail. We assume we are given an external device to which we will input and output 8-bit characters, with odd parity, using the Auto Enables feature. We will support this device with I/O polling routines following the patterns illustrated in Figures 5 and 6. We assume that the CPU will provide space to receive characters from the SIO as fast as the characters are received by the SIO, and that the CPU will transfer characters as fast as the output can be accomplished by the SIO.

We specify this example by giving the control bytes (commands) written to the SIO and the status bytes that must be read from the SIO. Recall that to write a command to a register, except register WR0, the number of the register to be written is first sent to register WR0; the following byte will be sent to the named register. Similarly, to read a register other than RR0 (the default), the number of the register to be read is sent to register WR0; the following byte will return the register named.

Initialization

We begin with the initialization code for the SIO. This follows the outline illustrated in Figure 4. In the following sample code, each time register WR0 is changed to point to another register, the Reset External/Status Interrupts command is given simultaneously. Whenever a transition on any of the external lines occurs, the bits reporting such a transition are latched until the Reset External/Status Interrupts command is given. Up to two transitions can be remembered by the SIO. Therefore, it is desirable to do at least two different Reset External/Status Interrupts commands as late as possible in the initialization so that the status bits reflect the most recent information. Since it doesn't hurt, we include these commands each time WR0 is changed to point to another register. This is an easy way to code the initialization to insure that the appropriate resets occur.

In the example below, the logic states on the $C/\overline{D}$ control line and the system data bus (D_7–D_0) are illustrated, together with comments.

Initialization (Continued)

	Bits sent to the SIO								
$C/\overline{D}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	**Effects and Comments**
1	0	0	0	1	1	0	0	0	Channel Reset command sent to register WR0 (D_5-D_3).
1	0	0	0	1	0	1	0	0	Point WR0 to WR4 (D_2-D_0) and issue a Reset External/ Status Interrupts command (D_5-D_3). Throughout the initialization, whenever we point WR0 to another register, we will also issue this command for the reasons noted above.
1	1	1	0	0	1	1	0	1	Set WR4 to indicate the following parameters (from left to right): A. Run at 1/64 the input clock rate (D_7-D_6). B. Disable the sync bits and send out 2 stop bits per character (D_5-D_2). C. Enable odd parity (D_1-D_0).
1	0	0	0	1	0	0	1	1	Point WR0 to WR3.
1	1	1	1	0	0	0	0	1	Set WR3 to indicate the following: A. 8-bit characters to be received (D_7-D_6). B. Auto Enables on (D_5). C. Receive (Rx) Enable on (D_0).
1	0	0	0	1	0	1	0	1	Point WR0 to WR5.
1	1	1	1	0	1	0	1	0	Set WR5 to indicate the following: A. Data Terminal Ready (DTR) on (D_7). B. 8-bit characters to be transmitted (D_6-D_5). C. Break not to be transmitted (D_4). D. Transmit (Tx) Enable on (D_3). E. Request To Send (RTS) on (D_1).

Reset and Error Sequences

In the receive and transmit routines that follow, we treat errors such as a transition on the Data Carrier Detect line by calling for a "reset sequence" to set the values in read register RR0 to reflect the current values found at the pins. This sequence consists of giving the Reset External/Status Interrupts command and beginning the driver over again. The command takes the form of a write to register WR0:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	1	0	0	0	0

Permits the status bits in RR0 to reflect current status.

This command does not turn off the latches for such things as parity errors stored in bits 4-6 of register RR1. When such an error occurs and the latches must be reset, we will call for an "error sequence." This sequence consists of giving the Error Reset command and beginning the driver over again. The command also takes the form of a write to register WR0:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	1	0	0	0	0

Resets the latches in register RR1.

When specifying the result of reading register RR0 or RR1 or specifying data, we will indicate the values read as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
D	D	D	D	D	D	D	D

Read a byte from the designated register..

Receive and Transmit Routines

Now we will first give an example of the receive routine. This parallels the preceding discussion of "Character Reception."

The framing error in this routine is reported on a character-by-character basis and it is not necessary to execute an "error sequence" if it is the only error received. However, it is not harmful to do so.

Next, we give an example of transmission code that parallels the above discussion on "Character Transmission."

Receive and Transmit Routines (Continued)

	Bits sent and received								
C/$\overline{D}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	**Effects and Comments (Receive Routine)**
1	D	D	D	D	D	D	D	D	Read a byte from RR0 (the default read register); if $D_0 = 0$ then no character is ready to be received. In this case, if D_7 (Break) or D_3 (Data Carrier Detect) have changed state, then execute a "reset sequence." If $D_0 = 0$ and D_7 and D_3 have not changed state, then no character is ready to be received; either loop on this read or try again later.
1	0	0	0	0	0	0	0	1	Point WR0 to read from RR1; we will now check for errors in the character read. Note that Reset External/Status Interrupt Commands are not done normally to avoid losing a line-status change.
1	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Read a byte from RR1; if either bit $D_6 = 1$ (framing error), $D_5 =$ (receive overrun error), or $D_4 = 1$ (parity error), the character is invalid and an "error sequence" should be executed after the following step.
0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Read in the data byte received. This must be done to clear the SIO buffer even if an error is detected.

	Bits sent and received								
C/$\overline{D}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	**Effects and Comments (Transmit Routine)**
1	D	D	D	D	D	D	D	D	Read a byte from RR0; if either bit D_3 (Data Carrier Detect), D_5 (Clear To Send) or D_7 (Break) have changed state, a "reset sequence" should be executed. If D_3, D_5 and D_7 have not changed state, then if $D_2 = 0$, the transmit buffer is not yet empty and a transmit cannot take place; either loop, reading RR0, or try again later.
0	D	D	D	D	D	D	D	D	Send the data byte to be transmitted.

SECTION 4

Interrupt-Driven Environments.

In a typical interrupt-driven environment, the SIO is initialized and the first transmission, if any, is begun. Thereafter, further I/O is interrupt driven. When action by the CPU is needed, an SIO interrupt causes the CPU to branch to an interrupt service routine after the CPU first saves state information.

In common usage, if I/O is interrupt driven, all interrupts are enabled and each different type of interrupt is used to cause a CPU branch to a different memory address. There is perhaps one frequent exception to this: parity errors are sometimes checked only at the end of a sequence of characters. The SIO facilitates this kind of operation since the parity error bit in read register RR1 is latched; once the bit is set it is not reset until an explicit reset operation is done. Thus, if a parity error has occurred on any character since last reset, bit 4 in register RR1 will be set. It is then possible to set register WR1 so that parity errors do not cause an error interrupt when a character is received. The user then has the obligation to poll for the value of the parity bit upon completion of the sequence.

SIO initialization for Interrupt mode normally requires two steps not used in Polled mode: an interrupt vector (if used) must be stored in write register WR2 of Channel B and write register WR1 must be initialized to specify the form of interrupt handling. It is preferable to initialize the interrupt vector in WR2 first. In this way an interrupt that arrives after the enabling bits are set in WR1 will cause proper interrupt servicing.

Interrupt Vectors

The interrupt vector, register WR2 of Channel B, is an 8-bit memory address. When an interrupt occurs (and note that an interrupt can only occur after interrupts have been enabled by writing to register WR1) the interrupt vector is normally taken as one byte of an address used by the CPU to find the location of the interrupt service routine. It is also possible to cause the particular type of interrupt condition to modify the address vector in WR2 before branching, resulting in a branch to a different memory location for each interrupt condition. This is a very useful construct; it permits short, special-purpose interrupt routines. The alternative, to have one general-purpose interrupt routine which must determine the situation before proceeding, can be quite inefficient. This is usually undesirable since the speed of interrupt-service routines is often a critical factor in determining system performance.

Interrupt Vectors (Continued)

There are at most eight different types of interrupts that the SIO may cause, four for each of the two channels. If bit 1 in register WR1 of Channel B has been turned on so that an interrupt will modify the interrupt vector, the three bits (1-3) of the vector will be changed to reflect the particular type of interrupt. These interrupts follow a hardware-set priority as follows, starting with the highest priority:

Channel A Special Receive Condition sets bits 3-1 of WR1 to 111,

Channel A Character Received sets bits 3-1 to 110,

Channel A Transmit Buffer Empty sets bits 3-1 to 100,

Channel A External/Status Transition sets bits 3-1 to 101.

Channel B Special Receive Condition sets bits 3-1 to 011,

Channel B Character Received sets bits 3-1 to 010,

Channel B Transmit Buffer Empty sets bits 3-1 to 000,

Channel B External/Status Transition sets bits 3-1 to 001.

For example, suppose that the interrupt vector had the value 11110001 and the Status Affects Vector bit is enabled, along with all interrupt-enable bits. When an External/Status transition occurs in Channel A, the three zeros (bits 3-1) would be modified to 101, yielding an interrupt vector of 11111011. The value of the interrupt vector, as modified, may be tained by reading register RR2 in Channel B.

Note that when a character is received, either the Special Receive Condition or Rx Character Available interrupt will occur, depending on whether or not an error occurred; the two will never occur simultaneously. Therefore, these two interrupts have equal priority. Note also that you can select not to be interrupted on some of the eight conditions; in this case, the presence of a particular condition for which interrupts are not desired can be determined by polling.

Suppose that interrupts have been enabled for all possible cases, and that the Status Affects Vector bit has also been enabled, allowing a different routine to handle each possible interrupt. As each interrupt causes a branch to a location only two bytes higher than the last interrupt, it is not possible to place a routine directly at the location where the vectored interrupt branches. In a Z80 CPU environment, these addresses refer to a table in memory which contains the actual starting location of the interrupt service routine. Also, since the state information saved by a CPU is rarely all of the information necessary to properly preserve a computation state, a typical interrupt service routine will begin by saving additional information and end by restoring that information. This is shown briefly in the examples of code in Appendix A.

It is possible to connect several SIOs using the interrupt mechanism and the IEI and IEO lines on the SIO to determine a priority for interrupt service. This mechanism is discussed on page 42 of the *Z80 SIO Technical Manual* and in the *Z80 Family Program Interrupt Structure Manual.* We do not go into it further in this application note.

Initialization

In general, the initialization procedure illustrated in Figure 4 can still be followed. All six steps (A through F) are required here. After completing the first four steps, which are the same as initialization for polled I/O, it is necessary to load an interrupt vector into WR2 of Channel B. Information is then written into register WR1 specifying which interrupts are to be enabled and whether a specific kind of interrupt should modify the interrupt vector.

Now let us give an example. As in the polled example, we assume that we are given a device to which we will input and output 8-bit characters, with odd parity, using the Auto Enables feature. We also assume the CPU will provide space to store characters as received.

We do not discuss the SIO commands and registers in detail. This is done in the *Z80 SIO Technical Manual.* A summary of the register bit assignments taken from the *Z80 SIO Serial Input/Output Product Specification* is included at the end of this note. Recall that to write a register other than register WR0, the number of the register to be written is first sent to register WR0, and the following byte will be sent to the named register. Similarly, to read a register other than RR0 (the default), the number of the register to be read is first written to register WR0 and the next byte read will return the contents of the register named.

In our example below, each time register WR0 is changed to point to another register, the Reset External/Status Interrupts command is also given. Whenever a transition on any of the external/status lines occurs, the bits reporting the transition are latched until the Reset External/Status Interrupts command is given. Up to two transitions can be remembered by the internal logic of the SIO. Therefore, it is desirable to do at least two different Reset External/Status Interrupt commands as late as possible in the initialization so that the status bits reflect the most recent information. Since it doesn't hurt, we give these commands each

Initialization (Continued)

time WR0 is changed to point to another register. This is an easy way to code the initialization to assure that the appropriate resets occur.

The columns below show the logic states on the C/$\overline{D}$ control line and the system data bus (D_7-D_0), together with comments.

C/$\overline{D}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Effects and Comments
1	0	0	0	1	1	0	0	0	Channel Reset command sent to register WR0 (D_5-D_3).
1	0	0	0	1	0	1	0	0	Point WR0 to WR4 (D_2-D_0) and issue a Reset External/Status Interrupts command (D_5-D_3). Throughout the initialization, whenever we point WR0 to another register we will also issue a Reset External/Status Interrupts command for the reasons noted above.
1	1	1	0	0	1	1	0	1	Set WR4 to indicate the following parameters (from left to right): A. Run at 1/64 the clock rate (D_7-D_6). B. Disable the sync bits and send out 2 stop bits per character (D_5-D_2). C. Enable odd parity (D_1-D_0).
1	0	0	0	1	0	0	1	1	Point WR0 to WR3.
1	1	1	1	0	0	0	0	1	Set WR3 to indicate the following: A. 8-bit characters to be received (D_7-D_6). B. Auto Enables on (D_5). C. Rx Enable on (D_0).
1	0	0	0	1	0	1	0	1	Point WR0 to WR5.
1	1	1	1	0	1	0	1	0	Set WR5 to indicate the following: A. Data Terminal Ready (DTR) on (D_7). B. 8-bit characters to be transmitted (D_6-D_5). C. Break not to be transmitted (D_4). D. Tx Enable on (D_3). E. Request To Send (RTS) on (D_1).
1	0	0	0	1	0	0	1	0	Point WR0 to WR2 (Channel B only).
1	1	1	1	0	0	0	0	0	Set the interrupt vector to point to address 11100000 (which is hex E0 and decimal 224). Once interrupts are enabled, they will cause a branch to this memory location, modified as described above if the Status Affects Vector bit is turned on (which it will be here). This vector is only set for Channel B, but it applies to both channels. It has no effect when set in Channel A.
1	0	0	0	1	0	0	0	1	Point WR0 to WR1.
1	0	0	0	1	0	1	1	1	Set WR1 to indicate the following: A. Cause interrupts on all characters received, treating a parity error as a Special Receive Condition interrupt (D_4-D_3). B. Turn on the Status Affects Vector feature, causing interrupts to modify the status vector—meaningful only on Channel B, but will not hurt if set for Channel A (D_2). C. Enable interrupts due to transmit buffer being empty (D_1). D. Enable External/Status interrupts (D_0).

Special Receive Condition Interrupts

A Special Receive Condition interrupt occurs (a) if a parity error has occurred, (b) if there is a receiver overrun error (data is being overwritten because the channel's three-byte receiver buffer is full and a new character is being received), or (c) if there is a framing error. The processing in this case is the following:

1. Issue an Error Reset command (to register WR0) to reset the latches in register RR1.
2. Read the character from the read buffer and discard it to empty the buffer.

It may be desirable to read and store the value of register RR1 to gather statistics on performance or determine whether to accept the character. In some applications, a character may still be acceptable if received with a framing error.

In specifying the result of reading register RR0, RR1, or specifying data, we will indicate the values as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
D	D	D	D	D	D	D	D

Read a byte from the designated register.

We now present an example of processing a Special Receive Condition interrupt.

				Bits sent and received					
C/$\overline{D}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	**Effects and Comments**
1	0	0	0	0	0	0	0	1	If we need to know what kind of error occurred, we point WR0 to read from RR1. Note that the Reset External/Status Interrups command is not used. This avoids losing a valid interrupt.
1	D	D	D	D	D	D	D	D	Read a byte from RR1; one or more of bit D_6 (framing error), D_5 (receive overrun error), or D_4 (parity error) will be 1 to indicate the specific error.
1	0	0	1	1	0	0	0	0	Give an Error Reset command to reset all the error latches.
0	D	D	D	D	D	D	D	D	Read in the data byte received. This must be done to clear the receiver buffer, but the character will generally be disregarded.

Received (Rx) Character Interrupts

When an Rx Character Available interrupt occurs, the character need only be read from the read buffer and stored. If parity is enabled with character lengths of 5, 6, or 7 bits, the received parity bit will be transferred with the character. Any unused bits will be 1s.

External/ Status Interrupts

To respond to an External/Status Interrupt, all that is necessary is to send a Reset External/Status Interrupts command. However, if you wish to find the specific cause of the interrupt, it is necessary to read register RR0. In this case, the complete processing takes the following form:

				Bits sent and received					
C/$\overline{D}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	**Effects and Comments**
1	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Read register RR0; bit D_7 (Break), D_5 (Clear To Send), or D_3 (Data Carrier Detect) will have had a transition to indicate the cause of the interrupt.
1	0	0	0	1	0	0	0	0	Give a Reset External/Status Interrupts command to set the latches in RR0 to their current values and stop External/Status Interrupts until another transition occurs.

Transmit (Tx) Buffer Empty Interrupts

The final kind of interrupt is a Tx Buffer Empty interrupt. If another character is ready to be transmitted on this channel, a Tx Buffer Empty interrupt indicates that it is time to do so. To respond to this interrupt, you need only send the next character. If no other character is ready to transmit, it may be desirable to mark the availability of the transmit mechanism for future use. In addition, you should send a Reset Tx Interrupt Pending command. This command prevents further transmitter interrupts until the next character has been loaded into the transmitter buffer.

The Reset Tx Interrupt Pending command to WR0 takes the following form:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	0	1	0	0	0

Reset Tx Interrupt Pending command; no Tx Empty Interrupts will be given until after the next character has been placed in the transmit buffer.

Z80 Assembler Code

To take these examples further, let us use Z80 Assembler code to implement the routines for a single channel. We assume that the location stored in register WR2 points to the appropriate interrupt service routine. We also assume that the following constants have already been defined:

SIOctrl. The address of the SIO's Channel B control port (we assume Channel B in order to include code to initialize the interrupt vector).

SIOdata. The address of the SIO's Channel B data port.

X. An address pointing to locations in memory that will be used to store various values.

We will write data as binary constants; the "B" suffix indicates this. In most cases, binary constants will be referred to by the command names. We begin with the initialization routine:

```
INIT:   LD    C,SIOctrl      ;place the address of the SIO in the C register for
                             ; use in subsequent output
        LD    A,00011000B    ;load Channel Reset command in A register
        OUT   (C) ,A         ;give Channel Reset command

        LD    A,00010100B    ;write to register WR0 pointing it to register WR4
        OUT   (C) ,A
        LD    A,11001101B    ;output basic I/O parameters to WR4
        OUT   (C) ,A

        LD    A,00010011B    ;write to register WR0 pointing it to register WR3
        OUT   (C) ,A
        LD    A,11100001B    ;output receive parameters to WR3
        OUT   (C) ,A

        LD    A,00010101B    ;write to register WR0 pointing it to register WR5
        OUT   (C) ,A
        LD    A,11101010B    ;output transmit parameters to WR5
        OUT   (C) ,A

        LD    A,00010010B    ;write to register WR0 pointing it to register WR2
                             ; (Channel B only)
        OUT   (C) ,A
        LD    A,11100000B    ;output the interrupt vector to WR2; in this case it is
                             ; decimal location 224
        OUT   (C) ,A

        LD    A,00010001B    ;write to register WR0 pointing it to register WR1
        OUT   (C) ,A
        LD    A,00010111B    ;output interrupt parameters to WR1
        OUT   (C) ,A

        RET                  ;return from initialization routine
```

Now let us look first at some sample codes for the Special Receive Condition interrupt routine, following the example above.

This is followed by a simple receive interrupt routine that will fetch the character received and store it in a temporary location.

```
SIOspecint:  PUSH  AF             ;save registers which will be used in this routine
             LD    A,00000001B    ;write to register WR0 pointing it to register RR1
             OUT   (SIOctrl) ,A
             IN    A,(SIOctrl)    ;fetch register RR1
             LD    (X) ,A         ;store result for later error analysis
             LD    A,00110000B    ;send an Error Reset command to reset device
                                  ; latches
             OUT   (SIOctrl) ,A
             IN    A,(SIOdata)    ;fetch the character received—we will discard this
                                  ; character since an error occurred during its
                                  ; reception
             POP   AF             ;restore saved registers
             EI                   ;enable interrupts
             RETI                 ;return from interrupt
```

Z80 Assembler Code (Continued)

```
SIOrecint:    PUSH    AF             ;save registers which will be used in this routine
              IN      A,(SIOdata)    ;fetch the character received
              LD      (X) ,A         ;store result for later use
              POP     AF             ;restore saved registers
              EI                     ;enable interrupts
              RETI                   ;return from interrupt
```

Of course, this last routine is probably far too simple to be useful. It is more likely that an interrupt routine will fill up a buffer of characters. A more complex example of a receive interrupt routine is contained in the chapter entitled "A Longer Example."

We now give a simple interrupt routine for an External/Status Interrupt, again assuming that the status contents of SIO register RR0 are stored in temporary location X:

```
SIOextint:    PUSH    AF             ;save registers which will be used in this routine
              LD      A,00010000B    ;send a Reset External/Status Interrupts command
              OUT     (SIOctrl) ,A
              IN      A,(SIOctrl)    ;fetch register RR0
              LD      (X) ,A         ;store result for later analysis
              POP     AF             ;restore saved registers
              EI                     ;enable interrupts
              RETI                   ;return from interrupt
```

Finally, we give the processing for a transmit interrupt routine in the case where no more characters are to be transmitted.

It is likely that this code would just be a portion of a more general transmit interrupt routine which would transmit a buffer-full of information at a time. A more complex example is included in the section entitled "A Longer Example."

```
SIOtrnint:    PUSH    AF             ;save registers which will be used in this routine
              LD      A,00101000B    ;send a Reset Tx Interrupt Pending command
              OUT     (SIOctrl) ,A
              POP     AF             ;restore saved registers
              EI                     ;Enable Interrupts
              RETI                   ;Return From Interrupt
```

SECTION 5

Questions and Answers.

Hardware Considerations

Q: Can a sloppy system clock cause problems in SIO operation?

A: Yes; the specifications for the system clock are very tight and must be met closely to prevent SIO malfunction. The clock high voltage must be greater than $V_{CC} - 0.6V$ but less than $+5.5V$. The clock low voltage must be greater than $-0.3V$ but less than $+0.45V$. The transitions between these two levels must be made in less than 30 ns. This does not apply to the $\overline{RxC}$ and $\overline{TxC}$ inputs which are standard TTL levels.

Q: When is a received character available to be read?

A: Data will be available a maximum of 13 system clock cycles from the rising edge of the $\overline{RxC}$ signal which samples the last bit of the data.

Q: What is the maximum time between character-insertion for transmission and next-character transmission?

A: This will vary depending on the speed of the line over which the character is being transmitted.

Q: Are the control lines to the SIO synchronous with the system clock so that noise may exist on the buses any time before setup requirements are satisfied?

A: Yes.

Q: In asynchronous use must receiver and transmitter clock rates be the same?

A: No, the SIO allows receive and transmit for each channel to use a different clock (thus up to four different clocks for receiving and transmitting data can be used on each SIO). However, the clock multiplier for each channel must be the same.

Q: Do Wait states have to be added when using the SIO with other processors other than the Z80 CPU?

A: No, provided that setup times specified for the SIO are met.

Q: If the Auto Enables bit in register WR3 is set, will a change in state on the $\overline{DCD}$ (Data Carrier Detect) or $\overline{CTS}$ (Clear To Send) lines still cause an interrupt?

A: Yes, provided that External/Status Interrupts are enabled (bit 0 in register WR1).

Q: Is the $\overline{M1}$ line used by the SIO if no interrupts are enabled?

A: No, and in this case the $\overline{M1}$ input should be tied high.

Q: Will the SIO continue to interrupt for a condition if the condition persists and the interrupt remains enabled?

A: Yes.

Q: What is the maximum data rate of the SIO?

A: It is 1/5 the rate of the system clock (CLK). For example, if the system clock operates at 4 MHz, the SIO's maximum transfer rate is 800K bits (100K bytes) per second.

Q: What pins are edge sensitive and should be strapped to avoid strange interrupts?

A: The external synchronization ($\overline{SYNC}$) pins and any other external status pins that are not used, including $\overline{CTS}$, and $\overline{DCD}$.

Q: What happens if the transmitter or receiver is disabled, while processing a character, by turning off its associated enable bit (bit 3 in register WR5 for transmit or bit 0 in register WR3 for receive)?

A: The transmitter will complete the character transmission in an orderly fashion. The receiver, however, will not finish. It will lose the character being received and no interrupt will occur.

Register Contents

Q: Does the Tx Buffer Empty (bit 2 in register RR0 get set when the last byte in the buffer is in the process of being shifted out?

A: No. The bit is set when the transmit buffer has already become empty. Similarly, the Tx Buffer Empty interrupt will not occur until the buffer is empty. The same is true for reception: the Rx Character Available bit (bit 0 in register RR0) is not set until the entire character is in the receive buffer, and the Rx Character Available interrupt will not occur until the entire character has been moved into the buffer.

Q: If an Rx Overrun error occurs (and bit 5 of register RR1 becomes latched on) because a new character has arrived, which character gets lost?

A: The most recently received character overwrites the next most recently received character.

Q: Does the Reset External/Status Interrupts command reset any of the status bits in register RR0?

A: No. However, when a transition occurs on any of the five External/Status bits in register RR0, all of the status bits are latched in their current position until a Reset External/Status Interrupts command is issued. Thus, the command does permit the appropriate bits of register RR0 to reflect the current signal values and should be done immediately after processing each transition on the channel.

Special Uses

Q: If the CPU does not have the return from interrupt sequence (RETI instruction on the Z80 CPU), how may the SIO be informed of the completion of interrupt handling?

A: This may be done by writing the Return From Interrupt command (binary, 00111000) to WR0 in Channel A of the SIO.

Q. If the CPU can be interrupted but cannot be used with vectored interrupts, how should processing be done?

A: Immediately after being interrupted, proceed in a manner similar to polling the SIO for both receive and transmit. Alternatively, the Status Affects Vector bit (bit 2 in register WR1) may be set and a 0 byte placed into the interrupt vector (register WR2 in Channel B). Then, the contents of the interrupt vector can be used to determine the cause of the interrupt and the channel on which the interrupt occurred. This can be queried by reading register RR1 of Channel B. Also, $\overline{\text{M1}}$ should be tied High and no equivalent to an interrupt acknowledge should be issued.

Q: How can the Wait/Ready ($\overline{\text{W/RDY}}$) signal be used by the CPU in asynchronous I/O?

A: The $\overline{\text{W/RDY}}$ signal is most commonly used in Block Transfer Mode with a DMA, and this use is described in the *Z80 DMA Technical Manual.* However, $\overline{\text{W/RDY}}$ may be directly connected to the Z80 CPU $\overline{\text{WAIT}}$ line in order to use the block I/O instructions OTDR, OTIR, INDR, and INIR. In this case, the SIO can be used for block transfer reception. To do this, the SIO is configured to interrupt on the first character received only (by settings bits 4 and 3 of register WR1 to 01) and additional characters are sensed using the $\overline{\text{W/RDY}}$ line. The block I/O instructions decrement a byte counter to determine when I/O is complete.

Q: Can the $\overline{\text{SYNC}}$ pin have any use in asynchronous I/O?

A: It may be used as a general-purpose input. For example, by connecting it to a modem ring indicator, the status of that ring indicator can be monitored by the CPU.

Q: How can the SIO be used to transmit characters containing fewer than 5 bits?

A: First, set bits 6 and 5 in register WR5 to indicate that five or fewer bits per character will be transmitted. The SIO then determines the number of bits to actually transmit from the data byte itself. The data byte should consist of zero or more 1s, three 0s, and the data to be transmitted. Thus, beginning the data byte with 11110001 will cause only the last bit to be transmitted:

Contents of data byte
(d = arbitrary value)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	1	1	1	0	0	0	d	1
1	1	1	0	0	0	d	d	2
1	1	0	0	0	d	d	d	3
1	0	0	0	d	d	d	d	4
0	0	0	d	d	d	d	d	5

*The rightmost number of bits indicated will be transmitted.

Q: Can a Break sequence be sent for a fixed number of character periods?

A: Yes. Break is continuously transmitted as logic 0 by setting bit 4 of register WR5. You can then send characters to the transmitter as long as the Break level is desired to persist. A Break signal, rather than the characters sent, will actually be transmitted, but each bit of each character sent will be clocked as if it were transmitted. The All Sent bit, bit 0 of register RR1, is set to 1 when the last bit of a character is clocked for transmission, and this may be used to determine when to reset bit 4 of register WR5 and stop the Break signal.

Q: If a Break sequence is initiated by setting bit 4 of register WR5, will any character in the process of being transmitted be completed?

A: No. Break is effective immediately when bit 4 of WR5 is set. The "all sent" bit in register RR1 should be monitored to determine when it is safe to initiate a Break sequence.

A Longer Example.

In this section, we give a longer example of asynchronous interrupt-driven full-duplex I/O using the SIO. The code for this example is contained in Appendix A, and the basic routines are flow charted in Figures 7-12.

The example includes code for initialization of the SIO, initialization of a receive buffer interrupt routine, and a transfer routine which causes a buffer of up to 80 characters of information to be transmitted on Channel A and a buffer of up to 80 characters of information to be received from Channel A. The transfer routine stops when either all data is received or an error occurs. Completion of an operation on a buffer for both receive and transmit is indicated by a carriage return character. Additional routines (not included in this example) would be needed to call the initialization code and initiate the transfer routine. Therefore, we do not present a complete example; that would only be possible when all details of a particular communication environment and operating system were known.

The code begins by defining the value of the SIO control and data channels, followed by location definitions for the interrupt vector. There is then a series of constant definitions of the various fields in each register of the SIO. This is followed by a table-driven SIO initialization routine called "SIO__init," shown in Figure 7, which uses the table beginning at the location "SIOItable." The SIO__Init routine initializes the SIO with exactly the same

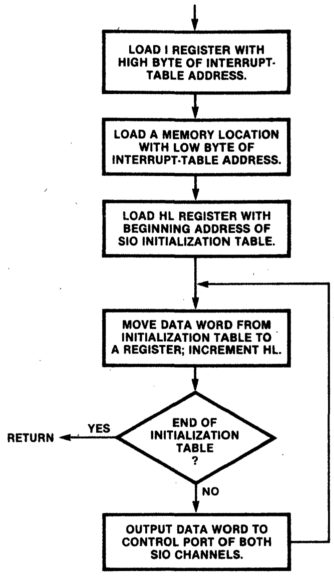

LOAD I REGISTER WITH HIGH BYTE OF INTERRUPT-TABLE ADDRESS.
LOAD A MEMORY LOCATION WITH LOW BYTE OF INTERRUPT-TABLE ADDRESS.
LOAD HL REGISTER WITH BEGINNING ADDRESS OF SIO INITIALIZATION TABLE.
MOVE DATA WORD FROM INITIALIZATION TABLE TO A REGISTER; INCREMENT HL.
END OF INITIALIZATION TABLE ?
YES
RETURN
NO
OUTPUT DATA WORD TO CONTROL PORT OF BOTH SIO CHANNELS.

Figure 7. Interrupt-Driven Initialization Routine

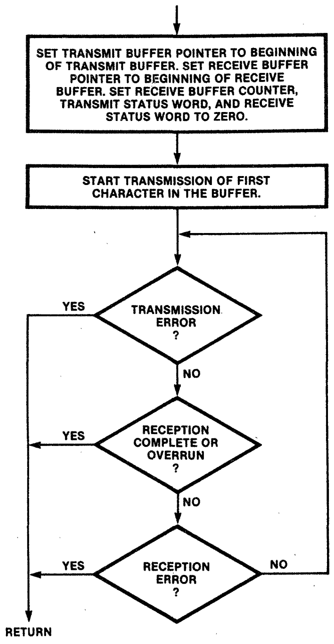

SET TRANSMIT BUFFER POINTER TO BEGINNING OF TRANSMIT BUFFER. SET RECEIVE BUFFER POINTER TO BEGINNING OF RECEIVE BUFFER. SET RECEIVE BUFFER COUNTER, TRANSMIT STATUS WORD, AND RECEIVE STATUS WORD TO ZERO.
START TRANSMISSION OF FIRST CHARACTER IN THE BUFFER.
TRANSMISSION ERROR ?
YES
NO
RECEPTION COMPLETE OR OVERRUN ?
YES
NO
RECEPTION ERROR ?
YES
NO
RETURN

Figure 8. Interrupt-Driven Transmit Routine

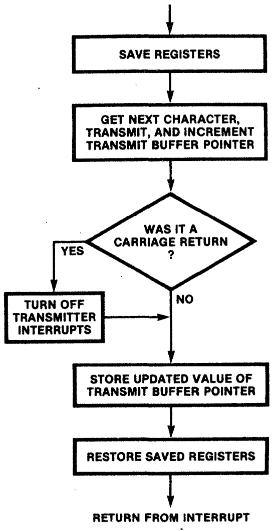

SAVE REGISTERS
GET NEXT CHARACTER, TRANSMIT, AND INCREMENT TRANSMIT BUFFER POINTER
WAS IT A CARRIAGE RETURN ?
YES
TURN OFF TRANSMITTER INTERRUPTS
NO
STORE UPDATED VALUE OF TRANSMIT BUFFER POINTER
RESTORE SAVED REGISTERS
RETURN FROM INTERRUPT

Figure 9. Transmitter Buffer Empty Interrupt Routine

A Longer Example (Continued)

parameters as the interrupt-driven example in the previous section. The table-driven version is presented simply as an alternative means of coding this material.

A short routine for filling the receive buffer with "FF" (hex) characters and buffer definitions follows the SIO_Init routine. This in turn is followed by the transfer routine, Figure 8, which begins transmitting on Channel A; transmission and reception is thereafter directed by the interrupt routines. After the transfer routine begins output, it checks for various error conditions and loops until there is either completion or an error.

Then the four interrupt routines follow: TxBEmpty, Figure 9, is called on a transmit buffer interrupt; it begins transmission of the next character in the buffer. A carriage return stops transmission. RecvChar, Figure 10, is called on a normal receive interrupt; it places the received character in the buffer if the buffer is not full and updates receive counters. The routines SpRecvChar, Figure 11, and ExtStatus, Figure 12, are error interrupts; they update information to indicate the nature of the error.

The code of this example can be used in a situation where data is being sent to a device which echoes the data sent. In such a case, the transmit and receive buffers could be compared upon completion for line or transmission errors.

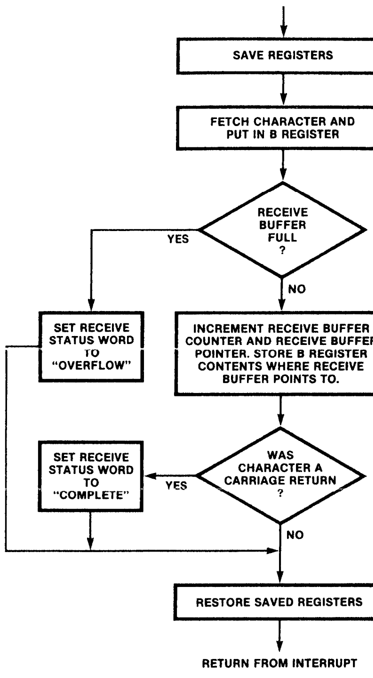

SAVE REGISTERS
FETCH CHARACTER AND PUT IN B REGISTER
RECEIVE BUFFER FULL ?
YES
NO
SET RECEIVE STATUS WORD TO "OVERFLOW"
INCREMENT RECEIVE BUFFER COUNTER AND RECEIVE BUFFER POINTER. STORE B REGISTER CONTENTS WHERE RECEIVE BUFFER POINTS TO.
SET RECEIVE STATUS WORD TO "COMPLETE"
WAS CHARACTER A CARRIAGE RETURN ?
YES
NO
RESTORE SAVED REGISTERS
RETURN FROM INTERRUPT

Figure 10. Receive Character Interrupt Routine

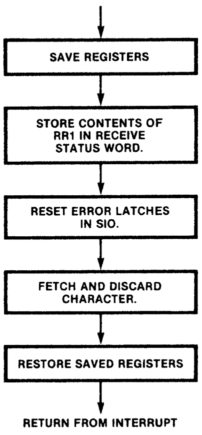

SAVE REGISTERS
STORE CONTENTS OF RR1 IN RECEIVE STATUS WORD.
RESET ERROR LATCHES IN SIO.
FETCH AND DISCARD CHARACTER.
RESTORE SAVED REGISTERS
RETURN FROM INTERRUPT

Figure 11. Special Receive Condition Interrupt Routine

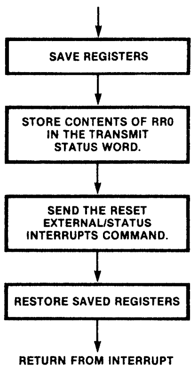

SAVE REGISTERS
STORE CONTENTS OF RR0 IN THE TRANSMIT STATUS WORD.
SEND THE RESET EXTERNAL/STATUS INTERRUPTS COMMAND.
RESTORE SAVED REGISTERS
RETURN FROM INTERRUPT

Figure 12. External/Status Interrupt Routine

Appendix A

Interrupt-Driven Code Example

SIO Port Identifiers and System Address Bus Addresses

```
SIO:        EQU     40H

SIOAData:   EQU     SIO+1
SIOACtrl:   EQU     SIO+2
SIOBData:   EQU     SIO+3
SIOBCtrl:   EQU     SIO+4
```

Table of Interrupt Vectors

The table (Int_Tab) starts at the lowest priority vector, which should be dddd000d.

```
            ORG     0D0H        ;starts at address with low
                                ; byte = 11010000
Int_Tab:    DEFW    TxBEmpty    ;interrupt types for Channel B
            DEFW    ExtStat
            DEFW    RxChar
            DEFW    SpRxCond

            DEFW    TxBEmpty    ;interrupt types for Channel A
            DEFW    ExtStat
            DEFW    RxChar
            DEFW    SpRxCond
```

Command Identifiers and Values

Includes all control bytes for asynchronous and synchronous I/O.

WR0 Commands

```
R0:         EQU     00H         ;SIO register pointers
R1:         EQU     01H
R2:         EQU     02H
R3:         EQU     03H
R4:         EQU     04H
R5:         EQU     05H
R6:         EQU     06H
R7:         EQU     07H

NC:         EQU     00H         ;Null Code
SA:         EQU     08H         ;Send Abort (SDLC)
RESI:       EQU     10H         ;Reset Ext/Stat Int
CHRST:      EQU     18H         ;Channel Reset
EIONRC:     EQU     20H         ;Enable Int On Next Rx Char
RTIP:       EQU     28H         ;Reset Tx Int Pending
ER:         EQU     30H         ;Error Reset
RFI:        EQU     38H         ;Return From Int
RRCC:       EQU     40H         ;Reset Rx CRC Checker
RTCG:       EQU     80H         ;Reset Tx CRC Generator
RTUEL:      EQU     0C0H        ;Reset Tx Under/EOM Latch
```

WR1 Commands

```
WAIT:       EQU     00H         ;Wait function
DRCVRI:     EQU     00H         ;Disable Receive interrupts
EXTIE:      EQU     01H         ;External interrupt enable
XMTRIE:     EQU     02H         ;Transmit interrupt enable
SAVECT:     EQU     04H         ;Status affects vector
FIRSTC:     EQU     08H         ;Rx interrupt on first character
PAVECT:     EQU     10H         ;Rx interrupt on all characters
                                ; (parity affects vector)
PDAVCT:     EQU     18H         ;Rx interrupt on all characters
                                ; (parity doesn't affect vector)
WRONRT:     EQU     20H         ;Wait/Ready on receive
RDY:        EQU     40H         ;Ready function
WRDYEN:     EQU     80H         ;Wait/Ready enable
```

WR2 Commands

```
IV:         EQU     00H
```

WR3 Commands

```
B5:         EQU     00H         ;Receive 5 bits/character
RENABL:     EQU     01H         ;Receiver enable
ENRCVR:     EQU     01H         ;Receiver enable
SCLINH:     EQU     02H         ;Sync character load inhibit
ADSRCH:     EQU     04H         ;Address search mode
RCRCEN:     EQU     08H         ;Receive CRC enable
HUNT:       EQU     10H         ;Enter hunt mode
AUTOEN:     EQU     20H         ;Auto enables
B7:         EQU     40H         ;Receive 7 bits/character
B6:         EQU     80H         ;Receive 6 bits/character
B8:         EQU     0C0H        ;Receive 8 bits/character
```

WR4 Commands

```
SYNC:       EQU     00H         ;Sync modes enable
NOPRTY:     EQU     00H         ;Disable parity
ODD:        EQU     00H         ;Odd parity
MONO:       EQU     00H         ;8 bit sync character
C1:         EQU     00H         ;X1 clock mode
PARITY:     EQU     01H         ;Enable parity
EVEN:       EQU     02H         ;Even parity
S1:         EQU     04H         ;1 stop bit/character
S1HALF:     EQU     08H         ;1 and a half stop bits/character
S2:         EQU     0CH         ;2 stop bits/character
BISYNC:     EQU     10H         ;16 bit sync character
SDLC:       EQU     20H         ;SDLC mode
ESYNC:      EQU     30H         ;External sync mode
C16:        EQU     40H         ;X16 clock mode
C32:        EQU     80H         ;X32 clock mode
C64:        EQU     0C0H        ;X64 clock mode
```

WR5 Commands

```
T5:         EQU     00H         ;Transmit 5 bits/character
XCRCEN:     EQU     01H         ;Transmit CRC enable
RTS:        EQU     02H         ;Request to send
SELCRC:     EQU     04H         ;Select CRC-16 polynomial
XENABL:     EQU     08H         ;Transmitter enable
BREAK:      EQU     10H         ;Send break
T7:         EQU     20H         ;Transmit 7 bits/character
T6:         EQU     40H         ;Transmit 6 bits/character
T8:         EQU     60H         ;Transmit 8 bits/character
DTR:        EQU     80H         ;Data terminal ready
```

Initialization

```
SIO_Init:   LD      HL, Int_Tab
            LD      A,H
            LD      I,A
            LD      A,L
            LD      (I_Loc),A
            LD      HL, SIOItable

Init_Loop:  LD      A,(HL)          ;loop for initialization
            INC     HL
            CP      0
            RET     Z
            OUT     (SIOACtrl),A
            OUT     (SIOBCtrl),A
            JR      Init_Loop

SIOItable:  DEFB    CR              ;table for initialization
            DEFB    R4 + RESI
            DEFB    C64 + ODD + PARITY + S2
            DEFB    R3 + RESI
            DEFB    B8 +  AUTOEN + ENRCVR
            DEFB    R5 + RESI
            DEFB    DTR + RTS + T8 + XENABL
            DEFB    R2 + RESI

I_Loc:      DEFS    1               ;location of int table
            DEFB    R1 + RESI       ;address
            DEFB    EXTIE + XMTRIE + SAVECT + PAVECT
            DEFB    0
```

Receiver Buffer Initialization

```
Buf_Init:   LD      A,BufLength   ;fill receiver buffer
            LD      B,A           ; with FF characters
            LD      HL,RBuffer    ; to detect errors
            LD      A,0FFH
Buf_1:      LD      (HL),A        ;a loop for Buf_Init
            INC     HL
            DJNZ    Buf_1
            RET
BufLength:  EQU     80            ;buffer length
XBuffer:    DEFS    BufLength     ;Tx buffer starting location
RBuffer:    DEFS    BufLength     ;Rx buffer starting location
XBufPtr:    DEFS    2             ;Tx pointer
RBufPtr:    DEFS    2             ;Rx pointer
RBufCtr:    DEFS    1             ;Rx counter
```

Transmit Routine (see Figure 8)

Initiates transmission of a buffer-full of data and terminates when an error is detected or a complete buffer has been received.

```
RxStat:     DEFS    1             ;Receive Status Word
TxStat:     DEFS    1             ;Transmit Status Word
Complete:   EQU     1
CR:         EQU     0DH
Break:      EQU     80H
EOM:        EQU     80H
Overflow:   EQU     0FFH
Transfer:   LD      HL,XBuffer    ;setup to begin Tx
            INC     HL
            LD      (XBufPtr),HL
            LD      HL,RBuffer
            LD      (RBufPtr),HL
            XOR     A             ;A=0
            LD      (RBufCtr),A
            LD      (TxStat),A
            LD      (RxStat),A
            LD      A,SIOAData    ;start Tx task
            LD      C,A
            LD      HL,(XBuffer)  ;first character
            LD      A,(HL)
            OUT     (C),A
Tloop:      LD      A,(TxStat)    ;await Tx completion or error
            CP      0
            RET     NZ
            LD      A,(RxStat)
            CP      Overflow
            RET     Z
            CP      Complete
            RET     Z
            JR      NZ,Tloop
            RET
```

Transmitter Buffer Empty Routine (see Figure 9)

```
TxBEmpty    PUSH    AF
            PUSH    BC
            PUSH    HL
            LD      HL,(XBufPtr)
            LD      A,SIOAData
            LD      C,A
            LD      A,(HL)
            OUTI
            CP      CR
            JR      NZ, TxBExit   ;last character?
            LD      A,RTIP        ;Reset Tx Int Pending
            INC     C
            OUT     (C),A         ;to control port
TxBExit:    LD      (XBufPtr),HL  ;save pointer
            POP     HL
            POP     BC
            POP     AF
            EI
            RETI
```

Receive Character Routine (see Figure 10)

```
RxChar:     PUSH    AF
            PUSH    BC
            LD      A,SIOAData
            LD      C,A
            IN      A,(C)         ;get character
            LD      B,A
            LD      A,(RBufCtr)
            CP      BufLength
            JR      Z,Over
            INC     A             ;bump counter
            LD      (RBufCtr),A
            LD      A,B
            LD      HL,(RBufPtr)  ;bump pointer
            LD      (HL),A
            INC     HL
            LD      (RBufPtr),HL
            CP      CR
            JR      NZ,RxExit
            LD      A,Complete
            LD      (RxStat),A
            JR      RxExit
Over:       LD      A,Overflow    ;indicate error
            LD      (RxStat),A
RxExit:     POP     BC
            POP     AF
            EI
            RETI
```

Special Receive Condition Routine (see Figure 11)

```
SpRxCond:   PUSH    AF
            PUSH    BC
            LD      A,SIOAData
            LD      C,A
            LD      A,R1          ;get RR1
            INC     C
            OUT     (C),A
            IN      A,(C)
            LD      (RxStat),A    ;save status
            LD      A,ER          ;Reset Errors
            DEC     C
            OUT     (C),A
            DEC     C
            IN      A,(C)         ;get character
            POP     BC
            POP     AF
            EI
            RETI
```

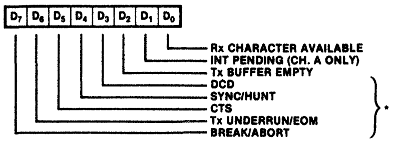
External/Status Routine (see Figure 12)

```
ExtStatus:  PUSH    AF
            PUSH    BC
            LD      A,SIOACtrl
            LD      C,A
            IN      A,(C)         ;get RR0
            LD      (TxStat),A
            LD      A,RESI        ;Reset Ext Stat Int
            OUT     (C),A
            POP     BC
            POP     AF
            EI
            RETI
END
```

Appendix B

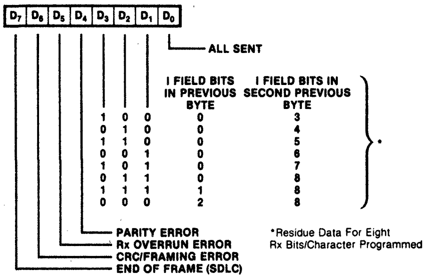
Read Register Bit Functions

READ REGISTER 0



*Used With "External/Status Interrupt" Mode

READ REGISTER 1†

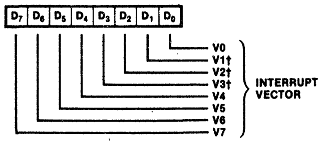


D3	D2	D1	I FIELD BITS IN PREVIOUS BYTE	I FIELD BITS IN SECOND PREVIOUS BYTE
1	0	0	0	3
0	1	0	0	4
1	1	0	0	5
0	0	1	0	6
1	0	1	0	7
0	1	1	0	8
1	1	1	1	8
0	0	0	2	8

*Residue Data For Eight Rx Bits/Character Programmed

†Used With Special Receive Condition Mode

READ REGISTER 2



†Variable if "Status Affects Vector" is Programmed

Appendix C

Write Register Bit Functions

WRITE REGISTER 0

D2	D1	D0	
0	0	0	REGISTER 0
0	0	1	REGISTER 1
0	1	0	REGISTER 2
0	1	1	REGISTER 3
1	0	0	REGISTER 4
1	0	1	REGISTER 5
1	1	0	REGISTER 6
1	1	1	REGISTER 7

D5	D4	D3	
0	0	0	NULL CODE
0	0	1	SEND ABORT (SDLC)
0	1	0	RESET EXT/STATUS INTERRUPTS
0	1	1	CHANNEL RESET
1	0	0	ENABLE INT ON NEXT Rx CHARACTER
1	0	1	RESET TxINT PENDING
1	1	0	ERROR RESET
1	1	1	RETURN FROM INT (CH-A ONLY)

D7	D6	
0	0	NULL CODE
0	1	RESET Rx CRC CHECKER
1	0	RESET Tx CRC GENERATOR
1	1	RESET Tx UNDERRUN/EOM LATCH

WRITE REGISTER 1

Bit	
D0	EXT INT ENABLE
D1	Tx INT ENABLE
D2	STATUS AFFECTS VECTOR (CH. B ONLY)

D4	D3	
0	0	Rx INT DISABLE
0	1	Rx INT ON FIRST CHARACTER *
1	0	INT ON ALL Rx CHARACTERS (PARITY AFFECTS VECTOR) *
1	1	INT ON ALL Rx CHARACTERS (PARITY DOES NOT AFFECT VECTOR) *

Bit	
D5	WAIT/READY ON R/$\overline{T}$
D6	$\overline{WAIT}$/READY FUNCTION
D7	WAIT/READY ENABLE

*Or On Special Condition

WRITE REGISTER 2 (CHANNEL B ONLY)

Bit	INTERRUPT VECTOR
D0	V0
D1	V1
D2	V2
D3	V3
D4	V4
D5	V5
D6	V6
D7	V7

WRITE REGISTER 3

Bit	
D0	Rx ENABLE
D1	SYNC CHARACTER LOAD INHIBIT
D2	ADDRESS SEARCH MODE (SDLC)
D3	Rx CRC ENABLE
D4	ENTER HUNT PHASE
D5	AUTO ENABLES

D7	D6	
0	0	Rx 5 BITS/CHARACTER
0	1	Rx 7 BITS/CHARACTER
1	0	Rx 6 BITS/CHARACTER
1	1	Rx 8 BITS/CHARACTER

WRITE REGISTER 4

Bit	
D0	PARITY ENABLE
D1	PARITY EVEN/$\overline{ODD}$

D3	D2	
0	0	SYNC MODES ENABLE
0	1	1 STOP BIT/CHARACTER
1	0	1½ STOP BITS/CHARACTER
1	1	2 STOP BITS/CHARACTER

D5	D4	
0	0	8 BIT SYNC CHARACTER
0	1	16 BIT SYNC CHARACTER
1	0	SDLC MODE (01111110 FLAG)
1	1	EXTERNAL SYNC MODE

D7	D6	
0	0	X1 CLOCK MODE
0	1	X16 CLOCK MODE
1	0	X32 CLOCK MODE
1	1	X64 CLOCK MODE

WRITE REGISTER 5

Bit	
D0	Tx CRC ENABLE
D1	$\overline{RTS}$
D2	$\overline{SDLC}$/CRC-16
D3	Tx ENABLE
D4	SEND BREAK

D6	D5	
0	0	Tx 5 BITS (OR LESS)/CHARACTER
0	1	Tx 7 BITS/CHARACTER
1	0	Tx 6 BITS/CHARACTER
1	1	Tx 8 BITS/CHARACTER

Bit	
D7	DTR

WRITE REGISTER 6

Bit	*
D0	SYNC BIT 0
D1	SYNC BIT 1
D2	SYNC BIT 2
D3	SYNC BIT 3
D4	SYNC BIT 4
D5	SYNC BIT 5
D6	SYNC BIT 6
D7	SYNC BIT 7

*Also SDLC Address Field

WRITE REGISTER 7

Bit	*
D0	SYNC BIT 8
D1	SYNC BIT 9
D2	SYNC BIT 10
D3	SYNC BIT 11
D4	SYNC BIT 12
D5	SYNC BIT 13
D6	SYNC BIT 14
D7	SYNC BIT 15

*For SDLC It Must Be Programmed to "01111110" For Flag Recognition

Using the Z80 SIO With SDLC

March 1981

Zilog

Application Brief

INTRODUCTION

This application brief describes the use of the Z80 SIO with the increasingly popular Synchronous Data Link Control (SDLC) communications protocol. A general description of the SDLC protocol and implementation of the protocol using the SIO are discussed. Descriptions for transmit and receive operations are given for use with simple contol frame sequences.

The reader should be familiar with hardware aspects of the SIO such as interfacing to the CPU and a modem. A more detailed description of the SDLC protocol is given in the IBM publication Synchronous Data Link Control General Information (document # GA27-3093-2). A description of the Z80 SIO can be found in the Zilog Data Book (document # 00-2034-A).

DESCRIPTION

Data communication today requires a communication protocol that can transfer data quickly and reliably. One such protocol, Synchronous Data Link Control (SDLC), is the link control used by the IBM Systems Network Architecture (SNA) communication package. SDLC is actually a subset of the International Standards Organization (ISO) link control called High Level Data Link Control (HDLC), which is used for international data communication.

SDLC is a Bit-Oriented Protocol (BOP). It differs from Byte-Control Protocols (BCPs), such as bisync, in having a few bit patterns for control functions instead of several special character sequences. The attributes of the SDLC protocol are position dependent rather than character dependent, so control is determined by the location of the byte as well as by the bit pattern.

A character in SDLC is sent as an octet, a group of eight bits. Several octets combine to form a message frame in such a way that each octet belongs to a particular field. Each message frame consists of an opening flag, address, control, information, Frame Check Sequence (FCS), and closing flag fields. The flag field contains a unique binary pattern, 01111110, which indicates the beginning and end of a message frame. This pattern simplifies the hardware interface in receiving devices so that multiple devices connected to a common link do not conflict with one another. The receiving devices respond only after a valid flag character has been detected. Once communication is established for a particular device, the other devices ignore the message until the next flag character is detected.

The address field contains one or more octets that are used to select a particular station on the data link. An address of all 1s is a global address code that selects all the devices on the link. When a primary station sends a frame, the address field is used to select a secondary station. When a secondary station sends a message to the primary station, the address field contains the secondary station address, i.e., the source of the message.

The control field follows the address field and contains information about the type of frame being sent. The control field consists of one octet and is always present.

The information field consists of zero or more 8-bit octets and contains any actual data transferred. However, because of the limitations of the error-checking algorithm used in the frame-check sequence, maximum recommended block size is approximately 4096 octets.

The Frame Check Sequence (FCS) follows the information field or the control field, depending on the type of message frame sent. The FCS is a 16-bit Cyclic Redundancy Code (CRC) of the bits in the address, control, and information fields. The FCS is based on the CRC-CCITT code, which uses the polynomial ($X^{16}+X^{12}+X^5+1$). The Z80 SIO contains the circuitry necessary to generate and check the FCS field.

This application note refers to products as Z80 "A", "B" etc. to specifiy the speed grade. We are no longer using those characters for the speeds. For more details, please refer to the ordering information section.

Zero insertion/deletion is a feature of SDLC that allows any data pattern to be sent. Zero insertion occurs when five consecutive 1s in the data pattern are transmitted. After the fifth 1, a 0 is inserted before the next bit is sent. The data is not affected in any way except that there is an extra 0 in the data stream. The receiver counts the 1s and deletes the 0 following the five consecutive 1s, thus restoring the original data pattern. Zero insertion and deletion is necessary because of the hardware constraint of searching for a flag character or abort sequence. Six 1s preceded and followed by a 0 indicate a flag character. Seven to 14 1s signify an abort, while an idle line (inactive) is indicated by 15 or more 1s. Under these three conditions, zero insertion/deletion is inhibited. Figure 2 illustrates the various line conditions.

SDLC protocol differs from other synchronous protocols with respect to frame timing. In bisync, for example, a host computer might interrupt transmission temporarily by sending sync characters instead of data. This suspended condition could continue as long as the receiver does not time out. With SDLC, however, it is illegal to send flags in the middle of a frame to idle the line. Such an occurrence causes an error condition and disrupts orderly operation. Therefore, the transmitting device must send a complete frame without interruption. If a message cannot be completed, the primary station sends an abort and resumes message transmission later. These conditions are discussed later in the Programming section of this brief.

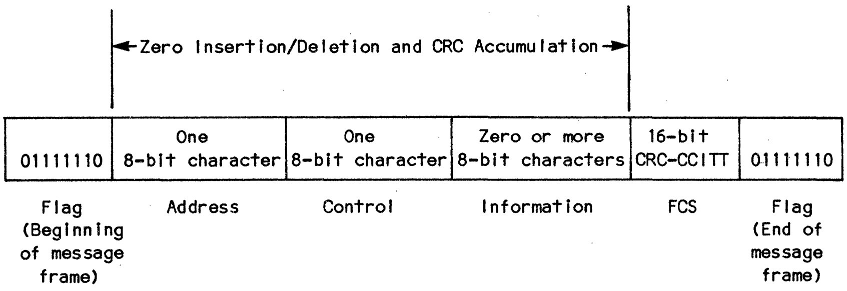

Zero Insertion/Deletion and CRC Accumulation
01111110
One 8-bit character
One 8-bit character
Zero or more 8-bit characters
16-bit CRC-CCITT
01111110
Flag (Beginning of message frame)
Address
Control
Information
FCS
Flag (End of message frame)

Figure 1. A Typical SDLC Message Frame Format

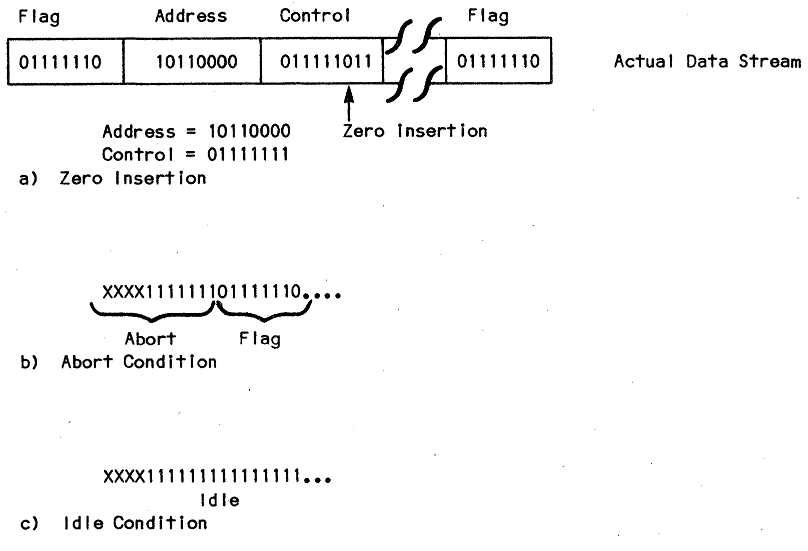


Figure 2. Bit Patterns for Various Line Conditions

PROGRAMMING THE SIO

Implementation of the SDLC protocol with the Z80 SIO is simplified by the design of the SIO. This section discusses four areas of SIO programming: initialization, transmit operation, receive operation, and exception condition processing.

Initialization defines the basic mode of operation for the SIO. Table 1 shows the sequence of steps used to initialize the SIO, along with the necessary parameters. Since vectored interrupts are used, the SIO is programmed with the status affects vector (SAV) bit (WR1, bit 2) set.

Other function bits that can be included are the external interrupt enable bit (WR1, bit 0), which results in an interrupt for each DCD or CTS change, T_X underrun or abort change; address search bit (WR3, bit 2), which when set, prevents the SIO from responding to data received unless the address byte matches the contents of WR6 or the global (FFH) address; auto enable bit (WR3, bit 5), which causes the inactive CTS level to disable the transmitter and the inactive DCD level to disable the receiver; and DTR (WR5, bit 7) and RTS (WR5, bit 1), which can be used to control a modem or other such device.

Once the SIO is initialized and the transmitter is enabled, it sends flag characters continuously until a message begins transmission. These flag characters consist of the full 8-bit pattern. Although the SIO can receive flag characters with shared 0s (0111111011111101111110...), it can only transmit flag characters without shared 0s (011111100111111001111110...).

Table 1. SIO Initialization Sequence

Register	Data	Function
0	00011000	Channel reset
2	(Vector)	Interrupt vector lower eight bits (channel B only)
4	00100000	SDLC mode
1	00011111	Interrupt control
6	(Address)	R_X address field
7	01111110	Flag field
5	11101011	T_X character length, enable, CRC enable RTS and DTR
3	11001001	R_X character length, enable, and CRC enable

TRANSMIT OPERATION

After the SIO has been initialized and enabled, it can begin sending SDLC frames by software activation of the transmitter. Activating the transmitter includes resetting the transmitter inactive semaphore (a program indicator), resetting the T_X CRC accumulation, sending a character to the SIO, and resetting the T_X underrun/EOM latch in the SIO. Figure 3 shows the sequence for transmitting a typical control message frame using interrupts.

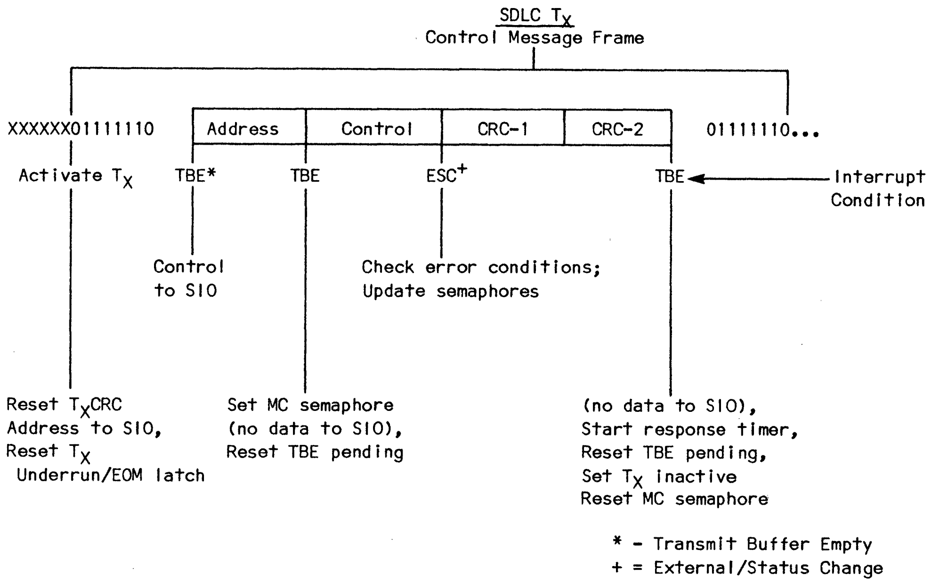

SDLC T_X
Control Message Frame
XXXXXX01111110
Address
Control
CRC-1
CRC-2
01111110...
Activate T_X
TBE*
TBE
ESC+
TBE
Interrupt Condition
Control to SIO
Check error conditions; Update semaphores
Reset T_XCRC Address to SIO, Reset T_X Underrun/EOM latch
Set MC semaphore (no data to SIO), Reset TBE pending
(no data to SIO), Start response timer, Reset TBE pending, Set T_X inactive Reset MC semaphore
* - Transmit Buffer Empty
+ = External/Status Change

Figure 3. A Typical Transmit Control Frame Sequence

When the SIO is loaded with the first data character (address byte), it stores the character in the T_X buffer until the current flag character has completed shifting. After the address byte is transferred into the shift register, a Transmit Buffer Empty (TBE) interrupt occurs. The program then loads the control character into the SIO and continues processing. The next TBE interrupt is ignored by the program (and no further data is sent to the SIO), but a Reset T_X Interrupt Pending command is issued to the SIO to clear the TBE interrupt condition. Also, the program Message completed (MC) semaphore is set so that appropriate action can be taken when the next TBE interrupt occurs.

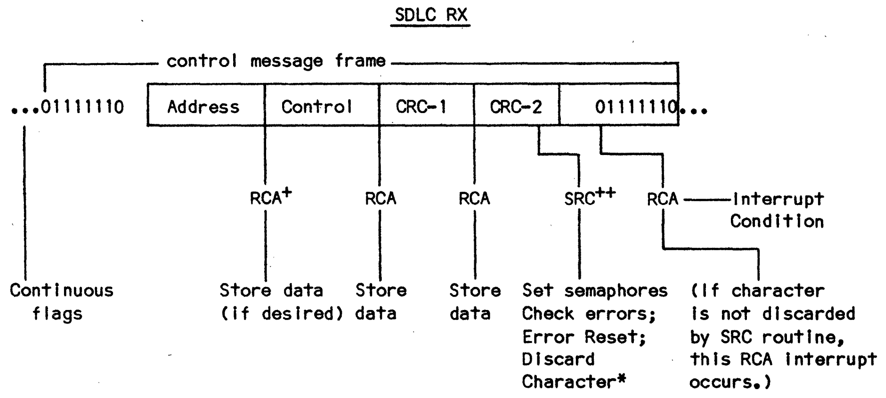
When the last data character (the control byte) has been shifted out of the SIO, the T_X underrun/EOM latch is set because the SIO buffer was not loaded with a character on the previous TBE interrupt. As a result, an External/ Status Change (ESC) interrupt occurs and the SIO begins transmitting the FCS bytes automatically. In the ESC interrupt service routine, the program checks for other condition changes including CTS, DCD, and abort, and passes the status on to the program at the next-higher level.

After the FCS bytes have been shifted out, the SIO generates a TBE interrupt to indicate that a flag character is being transmitted. The TBE interrupt service routine interprets the MC semaphore and determines that the frame has completed transmission. The program then clears the MC semaphore, sets the Transmitter Inactive semaphore, starts a timer for a response from the receiving device, and clears the TBE interrupt condition. At this point, transmission of an SDLC message frame is complete and another message frame may be sent.

If the transmitter is to be turned off, the program must allow at least a two-character time delay before disabling the transmitter. This can be accomplished by connecting the SIO T_X clock line to the input of a counter and having the counter interrupt the CPU when the bit count expires.

RECEIVE OPERATION

The SDLC receive sequence is slightly less complex than the transmit sequence. To begin, the SIO enters Hunt mode when any of three conditions occurs: receive enable, abort detect, or a software command. In Hunt mode the SIO searches for flag characters, and when it detects a flag, the SIO generates an ESC interrupt. This interrupt can be used to signal line activation or the end of an abort condition, depending upon the previous receive condition. For example, when the SIO has been initialized, the receive circuitry is enabled and immediately begins searching for flag characters (Hunt mode operation). When the first flag is detected, the SIO exits from Hunt mode, which results in an ESC interrupt, and the SIO begins searching for the address field. If the SIO is programmed for Address Search mode and an address is received that does not match the programmed address byte in the SIO, the SIO does nothing until the next flag is found, after which the SIO again searches for an address match.



NOTES

* The SRC routine normally reads the data character to clear the SIO buffer. This should be done after the program issues an Error Reset command.

+RCA = Receive Character Available

++SRC = Special Receive Condition (higher priority than RCA)

Figure 4. A Typical Receive Control Frame Sequence

If the address field matches the address byte programmed into the SIO, the SIO generates a Receive Character Available (RCA) interrupt when the address byte is ready to be transferred from the SIO to the CPU. If the SIO is programmed to interrupt on all receive characters, it generates an RCA interrupt for each character received thereafter. It should be noted that the SIO generates the RCA interrupt when a character reaches the top of the receive FIFO rather than when a character is transferred from the shift register to the FIFO. This means that if the FIFO is full of data, each character generates a separate RCA interrupt. This results in a more consistent software routine that does not need to check the receive FIFO, provided there is enough time between character transfers to allow the routine to complete the processing for each character.

After the last FCS byte of a frame is received and processed, the SIO generates a Special Receive Condition (SRC) interrupt, which is of higher priority than the RCA interrupt. In the SRC service routine, RR1 is read to determine the cause of the interrupt and the appropriate program semaphores are updated. Normal completion results in no FCS or overrun errors and the End-of-Frame bit is set. Upon completion of the SRC interrupt service routine, the program issues an Error Reset command to the SIO and reads the data port to discard the received data. If the data is not read and discarded, an RCA interrupt occurs. Now, a complete message frame and the first FCS byte are in the receive buffer.

Figure 4 shows the sequence for a typical control frame received by the SIO. If the address field byte is to be discarded, a program semaphore should initially be set to signal this to the RCA routine. After the address field has been received, the semaphore is cleared and reception continues normally. Note that upon completion of a frame, an RCA interrupt is generated for the first FCS byte and an SRC interrupt is generated for the last CRC byte.

Table 2 lists the contents of the interrupt service routines used with the SIO. The wake routine is not an interrupt service routine but is a routine called by the program on the next higher level to begin frame transmission. Once the wake routine is called, the program on the next higher level monitors the T_X active semaphore to determine when the current frame completes transmission and the next frame transmission can begin.

```
Wake:
   Clear $T_X$ inactive semaphore
   Reset $T_X$ CRC
   Data to SIO
      (Address field byte)
   Reset $T_X$ Underrun/EOM latch

Transmit Buffer Empty (TBE):
   If (MC cleared)
      If (buffer not empty)
         Data to SIO
      Else,
         Set MC semaphore
         Reset TBE condition
   Else,
      Clear MC
      Set $T_X$ inactive
      Reset TBE condition
      Start Response timer
```

```
External/Status Change (ESC):
   Clear DCD, CTS, abort semaphores
   If (abort)
      Set abort semaphore
   Else if (DCD change)
      Set DCD semaphore
   Else if (CTS change)
      Set CTS semaphore

Receive Character Available (RCA):
   If (EOF)
      Read and discard data
   Else,
      Store data

Special Receive Condition (SRC):
   Read SIO RR1
   If (EOF)
      Set EOF semaphore
   Else if (CRC error)
      Set $R_X$ CRC error semaphore
   Else if ($R_X$ overrun)
      Set $R_X$ overrun semaphore
   Issue Error Reset
   Read data & discard
```

Table 2. SIO SDLC Interrupt Service Routines

EXCEPTION CONDITION OPERATION

Most of the exception conditions encountered in the SDLC protocol have been discussed in the previous sections. They include abort detect and DCD or CTS change. This section further describes some of the more unusual conditions.

DCD and CTS Change. The program handles DCD and CTS change by updating its semaphores each time an ESC interrupt occurs. In this manner, the program on the next higher level monitors the semaphores and determines a course of action based on what these semaphores indicate.

Abort and Idle Line Detect. Abort and idle line detect are a bit more complicated, since they result in similar interrupt operations. An abort occurs during a valid message frame. If the abort time is greater than 14 bits, an idle line is detected. This detection can be done by activating a timer when the ESC interrupt that signals a marking line occurs. If another ESC interrupt occurs before the timer times out, the line is in an abort condition. If the timer times out before another ESC interrupt occurs, then the line is idle and the program can pursue an appropriate course of action. A possible mechanism for implementing the timer function is to use a programmable counter that is tied to the receive clock line to count bits. The counter is programmed for eight clock transitions and is started as soon as the SIO interrupts the CPU with an abort condition. Only eight clock transitions need to be counted because by the time the SIO generates the ESC interrupt, at least seven 1s have already passed. Figure 6 shows the abort/idle line timing and the interrupts resulting from the line changes.

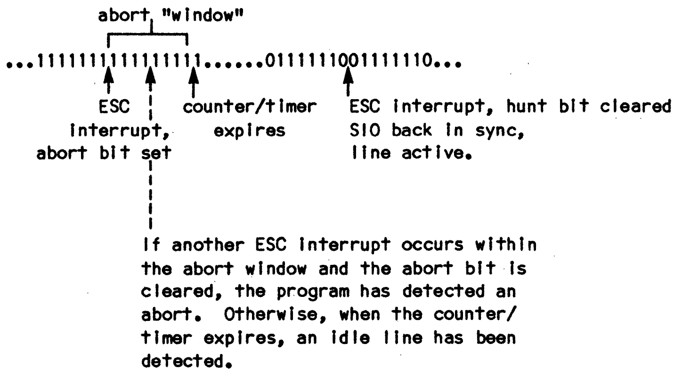

abort "window"
...111111111111111......0111111001111110...
ESC interrupt, abort bit set
counter/timer expires
ESC interrupt, hunt bit cleared SIO back in sync, line active.
If another ESC interrupt occurs within the abort window and the abort bit is cleared, the program has detected an abort. Otherwise, when the counter/timer expires, an idle line has been detected.

Figure 6. Abort/Idle Line Conditions

CONCLUSION

This brief describes implementation of the SDLC protocol using the SIO in an interrupt-driven environment. Descriptions for transmit and receive operations are given for use with simple control frame sequences. For frames that transfer data, the sequences are similar except for transmit, where a data character is sent to the SIO for a TBE interrupt. For receive, multiple RCA interrupts occur for each data byte received.

The Z80 SIO enhances system performance by minimizing CPU intervention during data transfers using the SDLC protocol. Performance can be improved further by using the Z80 DMA with the SIO, resulting in an efficient system configuration that reduces CPU interaction to a minimum.

APPENDIX

Following is the listing of a simple SIO test progam that uses the SDLC protocol. This program uses vectored interrupts to send a short SDLC control frame consisting of Address 9EH, Control 19H, and Data 81H. The response timer times the response of the receiving station after a message has been sent. If the response timer expires, the program on the next higher level normally retransmits the message frame (if the retransmit count has not yet expired). This program transmits continuously until the processor is reset or interrupted by an external source.

```
                                   TEST.SDLC
LOC   OBJ CODE M STMT SOURCE STATEMENT                          ASM 5.9

                  1  ;        SIO SDLC TEST PROGRAM
                  2
                  3  ; [0]    01-21-81/MDP           INITIAL CREATION
                  4
```

```
;       THIS PROGRAM SENDS ADDRESS 9EH, CONTROL 19H,
;       AND DATA 81H CONTINUOUSLY USING THE Z80 VECTORED
;       INTERRUPT MODE. THE SIO IS INITIALIZED TO USE
;       SDLC WITH THE BAUD RATE CLOCK SUPPLIED BY
;       HARDWARE INTERNAL TO THE SYSTEM.

;       EQUATES

ADDRESS:        EQU     9EH             ;ADDRESS FIELD
CTRL:   EQU     19H                     ;CONTROL FIELD
DATA:   EQU     81H                     ;INFORMATION FIELD
MSGLEN: EQU     1                       ;MESSAGE LENGTH
RAM:    EQU     2000H                   ;RAM ORIGIN
RAMSIZ: EQU     1000H                   ;RAM SIZE
SIODA:  EQU     0                       ;SIO PORT A DATA
SIOCA:  EQU     SIODA+1                 ;SIO PORT A CTRL
SIODB:  EQU     SIODA+2                 ;SIO PORT B DATA
SIOCB:  EQU     SIODB+1                 ;SIO PORT B CTRL
CIOC:   EQU     8                       ;CIO PORT C
CIOB:   EQU     CIOC+1                  ;CIO PORT B
CIOA:   EQU     CIOC+2                  ;CIO PORT A
CIOCTL: EQU     CIOC+3                  ;CIO CTRL PORT
BAUD:   EQU     9600                    ;ASYNC BAUD RATE
RATE:   EQU     BAUD/100
CIOCNT: EQU     9216/RATE
LITE:   EQU     0E0H                    ;LIGHT PORT
RSPCNT: EQU     100                     ;RESPONSE TIMER VALUE

;       SIO PARAMETERS

SIOWR0: EQU     0
        CHRES:  EQU     18H             ;CH. RESET CMD
        ESCRES: EQU     10H             ;ESC RESET CMD
        TBERES: EQU     28H             ;TBE RESET CMD
        RETIA:  EQU     38H             ;RETI CH. A
        ENINRX: EQU     20H             ;ENAB. INT. NEXT RX
        SRCRES: EQU     30H             ;SRC RESET CMD
        RCRCRE: EQU     40H             ;RX CRC RESET CMD
        TCRCRE: EQU     80H             ;TX CRC RESET CMD
        EOMRES: EQU     0C0H            ;EOM RESET CMD

SIOWR1: EQU     1
        WREN:   EQU     80H             ;WAIT/RDY ENABLE
        RDY:    EQU     40H             ;READY FUNCT.
        WRONR:  EQU     20H             ;WAIT/RDY ON RX
        RXIFC:  EQU     8               ;RX INT. FIRST CHAR
        RXIAP:  EQU     10H             ;RX INT. ALL + PARITY
        RXIA:   EQU     18H             ;RX INT. ALL
        SIOSAV: EQU     4               ;STATUS AFFECTS VECT.
                                        ;(CH.B ONLY)
        TXI:    EQU     2               ;TX INT. ENABLE
        EXTI:   EQU     1               ;EXT. INT. ENABLE

SIOWR2: EQU     2                       ;(CH.B ONLY)

SIOWR3: EQU     3
        RX8:    EQU     0C0H            ;RX 8 BITS
        RX6:    EQU     80H             ;RX 6 BITS
        RX7:    EQU     40H             ;RX 7 BITS
        RX5:    EQU     0               ;RX 5 BITS
        AUTOEN: EQU     20H             ;AUTO ENABLES
        HUNT:   EQU     10H             ;HUNT MODE
        RXCRC:  EQU     8               ;RX CRC ENABLE
        ADSRCH: EQU     4               ;ADDR SEARCH
        SYNINH: EQU     2               ;SYNC LOAD INHIBIT
        RXEN:   EQU     1               ;RX ENABLE

SIOWR4: EQU     4
        X64:    EQU     0C0H            ;64X CLOCK
        X32:    EQU     80H             ;32X CLOCK
        X16:    EQU     40H             ;16X CLOCK
        X1:     EQU     0               ;1X CLOCK
```

```
 LOC    OBJ CODE M STMT SOURCE STATEMENT                                    ASM 5.9
                  77          EXTSYN:  EQU    30H        ;EXT. SYNC ENABLE
                  78          SDLC:    EQU    20H        ;SDLC MODE
                  79          SYN16:   EQU    10H        ;16 BIT SYNC
                  80          SYN8:    EQU    0          ;8 BIT SYNC
                  81          STOP2:   EQU    0CH        ;2 STOP BITS
                  82          STOP15:  EQU    8          ;1.5 STOP BITS
                  83          STOP1:   EQU    4          ;1 STOP BIT
                  84          SYNCEN:  EQU    0          ;SYNC ENABLE
                  85          EVEN:    EQU    2          ;EVEN PARITY
                  86          PARITY:  EQU    1          ;PARITY ENABLE
                  87
                  88   SIOWR5: EQU     5
                  89          DTR:     EQU    80H        ;ACTIVATE DTR
                  90          TX8:     EQU    60H        ;TX 8 BITS
                  91          TX6:     EQU    40H        ;TX 6 BITS
                  92          TX7:     EQU    20H        ;TX 7 BITS
                  93          TX5:     EQU    0          ;TX 5 BITS
                  94          BREAK:   EQU    10H        ;TX BREAK
                  95          TXEN:    EQU    8          ;TX ENABLE
                  96          CRC16:   EQU    4          ;CRC-16 MODE
                  97          RTS:     EQU    2          ;ACTIVATE RTS
                  98          TXCRC:   EQU    1          ;TX CRC ENABLE
                  99
                 100   SIOWR6: EQU     6                 ;LOW SYNC OR ADDR
                 101
                 102   SIOWR7: EQU     7                 ;HIGH SYNC OR FLAG
                 103
                 104   ;       SIOFLG = FLAGS FOR SIO STATUS
                 105
                 106   ;       BIT --- SET CONDITION
                 107
                 108   ;        0       TX ACTIVE
                 109   ;        1       MESSAGE COMPLETE
                 110   ;        2       CTS ACTIVE
                 111   ;        3       DCD ACTIVE
                 112   ;        4       ABORT DETECT
                 113   ;        5       RX OVERRUN ERROR
                 114   ;        6       RX CRC ERROR
                 115   ;        7       RX END OF FRAME
                 116   *E
                 117
                 118   ;;      *** MAIN PROGRAM ***
                 119
0000             120           ORG     0
0000   C32000    121           JP      BEGIN             ;GO MAIN PROGRAM
                 122
                 123   ;       INTERRUPT VECTORS
                 124   ;       (MUST START ON EVEN BOUNDARY)
                 125
0010             126           ORG     $.AND.0FFF0H.OR.10H
                 127   INTVEC:
                 128   SIOVEC:
0010   9C00      129           DEFW    CHBTBE
0012   D100      130           DEFW    CHBESC
0014   0101      131           DEFW    CHBRCA
0016   0F01      132           DEFW    CHBSRC
0018   3B01      133           DEFW    CHATBE
001A   4301      134           DEFW    CHAESC
001C   4B01      135           DEFW    CHARCA
001E   5101      136           DEFW    CHASRC
                 137
                 138   BEGIN:
0020   314020    139           LD      SP,STAK           ;INIT SP
0023   ED5E      140           IM      2                 ;VECTOR INTERRUPT MODE
0025   3E00      141           LD      A,INTVEC/256      ;UPPER VECTOR BYTE
0027   ED47      142           LD      I,A
0029   214520    143           LD      HL,BUFFER
002C   369E      144           LD      (HL),ADDRESS      ;STORE ADDRESS
002E   23        145           INC     HL
002F   3619      146           LD      (HL),CTRL         ;STORE CTRL BYTE
0031   23        147           INC     HL
0032   3681      148           LD      (HL),DATA         ;STORE DATA BYTE
0034   CD4C00    149           CALL    INIT              ;INIT DEVICES
```

```
0037    218720      150           LD      HL,RBUF          ;SETUP READ BUFFER
003A    228520      151           LD      (RBPTR),HL
                    152  LOOP:
003D    213D00      153           LD      HL,LOOP          ;SETUP STACK FOR RETURN
0040    E5          154           PUSH    HL
0041    CD7D00      155           CALL    WAKE             ;WAKE TX
                    156  LOOP1:
0044    3A4020      157           LD      A,(SIOFLG)       ;CHECK TX ACTIVE FLAG
0047    CB47        158           BIT     0,A
0049    20F9        159           JR      NZ,LOOP1         ;LOOP IF TX ACTIVE
004B    C9          160           RET
                    161
                    162  INIT:
                    163  SIOINI:
004C    217001      164           LD      HL,SIOTA         ;INIT CH.A
004F    0E01        165           LD      C,SIOCA
0051    060A        166           LD      B,SIOEA-SIOTA
0053    EDB3        167           OTIR
0055    217A01      168           LD      HL,SIOTB         ;INIT CH.B
0058    0E03        169           LD      C,SIOCB
005A    0610        170           LD      B,SIOEB-SIOTB
005C    EDB3        171           OTIR
005E    3E00        172           LD      A,0              ;CLEAR FLAG BYTE
0060    324020      173           LD      (SIOFLG),A
                    174  CIOINI:
0063    DB0B        175           IN      A,(CIOCTL)       ;INSURE STATE 0
0065    AF          176           XOR     A                ;POINT TO REG 0
0066    D30B        177           OUT     (CIOCTL),A
0068    DB0B        178           IN      A,(CIOCTL)       ;CLEAR RESET OR STATE 0
006A    AF          179           XOR     A
006B    D30B        180           OUT     (CIOCTL),A       ;POINT TO REG 0
006D    3C          181           INC     A                ;WRITE RESET
006E    D30B        182           OUT     (CIOCTL),A
0070    AF          183           XOR     A                ;CLEAR RESET COND.
0071    D30B        184           OUT     (CIOCTL),A
0073    218A01      185           LD      HL,CLST          ;INIT CIO
0076    060E        186           LD      B,CEND-CLST
0078    0E0B        187           LD      C,CIOCTL
007A    EDB3        188           OTIR
007C    C9          189           RET
                    190
                    191  WAKE:
007D    3A4020      192           LD      A,(SIOFLG)       ;SET ACTIVE FLAG
0080    CBC7        193           SET     0,A
0082    324020      194           LD      (SIOFLG),A
0085    214520      195           LD      HL,BUFFER        ;SET BUFFER PTR
0088    224320      196           LD      (BUFPTR),HL
008B    3E03        197           LD      A,2+MSGLEN       ;SET BYTE COUNT
008D    324120      198           LD      (BYTES),A
0090    3E80        199           LD      A,TCRCRE         ;CLEAR TX CRC
0092    D303        200           OUT     (SIOCB),A
0094    CD9C00      201           CALL    CHBTBE           ;START TRANSMIT
0097    3EC0        202           LD      A,EOMRES         ;RESET EOM LATCH
0099    D303        203           OUT     (SIOCB),A
009B    C9          204           RET
                    205  *E
                    206
                    207  ;;       INTERRUPT SERVICE ROUTINES
                    208
                    209  CHBTBE:
009C    CD5901      210           CALL    SAVE             ;CH.B TX BUFFER EMPTY
009F    214020      211           LD      HL,SIOFLG        ;POINT TO FLAG BYTE
00A2    CB4E        212           BIT     1,(HL)           ;CHECK MC FLAG
00A4    201D        213           JR      NZ,CHBTB2        ;BRANCH IF MESSAGE COMPLE
TE
00A6    3A4120      214           LD      A,(BYTES)        ;CHECK BYTE COUNT
00A9    B7          215           OR      A
00AA    280F        216           JR      Z,CHBTB1         ;BRANCH IF DATA DONE
00AC    3D          217           DEC     A
00AD    324120      218           LD      (BYTES),A
00B0    2A4320      219           LD      HL,(BUFPTR)
00B3    7E          220           LD      A,(HL)
00B4    D302        221           OUT     (SIODB),A
```

```
 LOC   OBJ CODE M STMT SOURCE STATEMENT                                    ASM 5.9
00B6   23            222           INC     HL
00B7   224320        223           LD      (BUFPTR),HL
00BA   C9            224           RET
                     225   CHBTB1:
00BB   CBCE          226           SET     1,(HL)              ;SET MC FLAG
00BD   3EC0          227           LD      A,EOMRES
00BF   D303          228           OUT     (SIOCB),A
00C1   1809          229           JR      CHBTB3
                     230   CHBTB2:
00C3   CB8E          231           RES     1,(HL)              ;CLEAR MC FLAG
00C5   CB86          232           RES     0,(HL)              ;SET TX INACTIVE
00C7   3E64          233           LD      A,RSPCNT            ;START RESPONSE TIMER
00C9   324220        234           LD      (RSPTMR),A
                     235   CHBTB3:
00CC   3E28          236           LD      A,TBERES            ;RESET TBE INT. PEND.
00CE   D303          237           OUT     (SIOCB),A
00D0   C9            238           RET
                     239.
                     240   CHBESC:
00D1   CD5901        241           CALL    SAVE                ;CH.B EXTERNAL/STATUS CHG
00D4   214020        242           LD      HL,SIOFLG           ;GET FLAG BYTE
00D7   CB96          243           RES     2,(HL)
00D9   CB9E          244           RES     3,(HL)
00DB   CBA6          245           RES     4,(HL)
00DD   DB03          246           IN      A,(SIOCB)           ;READ RR0
00DF   47            247           LD      B,A                 ;STORE IN %B
00E0   CB58          248           BIT     3,B                 ;CHECK DCD BIT
00E2   C4FB00        249           CALL    NZ,SETDCD
00E5   CB68          250           BIT     5,B                 ;CHECK CTS BIT
00E7   C4FE00        251           CALL    NZ,SETCTS
00EA   CB78          252           BIT     7,B                 ;CHECK ABORT BIT
00EC   C4F800        253           CALL    NZ,SETABT
00EF   CB4E          254           BIT     1,(HL)              ;CHECK MC FLAG
00F1   2800          255           JR      Z,CHBES1            ;BRANCH IF CLEAR
                     256   CHBES1:
00F3   3E10          257           LD      A,ESCRES            ;RESET ESC
00F5   D303          258           OUT     (SIOCB),A
00F7   C9            259           RET
                     260   SETABT:
00F8   CBE6          261           SET     4,(HL)
00FA   C9            262           RET
                     263   SETDCD:
00FB   CBDE          264           SET     3,(HL)
00FD   C9            265           RET
                     266   SETCTS:
00FE   CBD6          267           SET     2,(HL)
0100   C9            268           RET
                     269
                     270   CHBRCA:
0101   CD5901        271           CALL    SAVE                ;CH.B RX CHAR AVAIL.
0104   DB02          272           IN      A,(SIODB)
0106   2A8520        273           LD      HL,(RBPTR)          ;GET READ BUFF PTR
0109   77            274           LD      (HL),A
010A   23            275           INC     HL
010B   228520        276           LD      (RBPTR),HL
010E   C9            277           RET
                     278
                     279   CHBSRC:
010F   CD5901        280           CALL    SAVE                ;CH.B SPECIAL RX COND.
0112   3E01          281           LD      A,1
0114   D303          282           OUT     (SIOCB),A           ;READ RR1
0116   DB03          283           IN      A,(SIOCB)
0118   47            284           LD      B,A                 ;SAVE IN %B
0119   214020        285           LD      HL,SIOFLG
011C   CBB6          286           RES     6,(HL)              ;CLEAR CRC ERROR FLAG
011E   CB78          287           BIT     7,B                 ;CHECK EOF BIT
0120   C43801        288           CALL    NZ,SETEFF           ;BRANCH IF NOT EOF
0123   CB70          289           BIT     6,B                 ;CHECK CRC ERROR
0125   C43501        290           CALL    NZ,SETCRC
0128   CB68          291           BIT     5,B                 ;CHECK OVRRUN BIT
012A   C43201        292           CALL    NZ,SETOVR
                     293   CHBSR1:
012D   3E30          294           LD      A,SRCRES            ;ERROR RESET CMD
```

```
 LOC    OBJ CODE M STMT SOURCE STATEMENT                           ASM 5.9
012F    D303        295          OUT     (SIOCB),A
0131    C9          296          RET
                    297  SETOVR:
0132    CBEE        298          SET     5,(HL)
0134    C9          299          RET
                    300  SETCRC:
0135    CBF6        301          SET     6,(HL)
0137    C9          302          RET
                    303  SETEFF:
0138    CBFE        304          SET     7,(HL)
013A    C9          305          RET
                    306
                    307  CHATBE:
013B    CD5901      308          CALL    SAVE            ;CH.A TX BUFFER EMPTY
013E    3E28        309          LD      A,TBERES
0140    D301        310          OUT     (SIOCA),A
0142    C9          311          RET
                    312
                    313  CHAESC:
0143    CD5901      314          CALL    SAVE            ;CH.A EXTERNAL/STATUS CHG
0146    3E10        315          LD      A,ESCRES
0148    D301        316          OUT     (SIOCA),A
014A    C9          317          RET
                    318
                    319  CHARCA:
014B    CD5901      320          CALL    SAVE            ;CH.A RX CHAR AVAIL.
014E    DB00        321          IN      A,(SIODA)
0150    C9          322          RET
                    323
                    324  CHASRC:
0151    CD5901      325          CALL    SAVE            ;CH.B SPECIAL RX COND.
0154    3E30        326          LD      A,SRCRES
0156    D301        327          OUT     (SIOCA),A
0158    C9          328          RET
                    329
                    330  ;       SAVE REGISTER ROUTINE
                    331
                    332  SAVE:
0159    E3          333          EX      (SP),HL         ;SP =    HL
015A    D5          334          PUSH    DE              ;        DE
015B    C5          335          PUSH    BC              ;        BC
015C    F5          336          PUSH    AF              ;        AF
015D    DDE5        337          PUSH    IX              ;        IX
015F    FDE5        338          PUSH    IY              ;        IY
0161    CD6F01      339          CALL    GO              ;        PC
0164    FDE1        340          POP     IY
0166    DDE1        341          POP     IX
0168    F1          342          POP     AF
0169    C1          343          POP     BC
016A    D1          344          POP     DE
016B    E1          345          POP     HL
016C    FB          346          EI
016D    ED4D        347          RETI
                    348
                    349  GO:
016F    E9          350          JP      (HL)
                    351  *E
                    352
                    353  ;;      CONSTANTS
                    354
                    355  SIOTA:
0170    00          356          DEFB    SIOWR0          ;CHAN. RESET
0171    18          357          DEFB    CHRES
0172    01          358          DEFB    SIOWR1          ;CHAN. CHARACS.
0173    D2          359          DEFB    WREN+RDY+RXIAP+TXI
0174    04          360          DEFB    SIOWR4          ;MODE
0175    4F          361          DEFB    X16+STOP2+EVEN+PARITY
0176    05          362          DEFB    SIOWR5          ;TX PARAMS.
0177    AA          363          DEFB    DTR+TX7+TXEN+RTS
0178    03          364          DEFB    SIOWR3          ;RX PARAMS.
0179    41          365          DEFB    RX7+RXEN
                    366  SIOEA:  EQU     $
                    367
 LOC    OBJ CODE M STMT SOURCE STATEMENT                              ASM 5.9
                    368  SIOTB:
017A    00          369           DEFB    SIOWR0          ;CHAN. RESET
017B    18          370           DEFB    CHRES
017C    02          371           DEFB    SIOWR2          ;VECTOR REG.
017D    10          372           DEFB    SIOVEC.AND.255
017E    04          373           DEFB    SIOWR4          ;MODE
017F    20          374           DEFB    X1+SDLC+SYNCEN
0180    01          375           DEFB    SIOWR1          ;CHAN. CHARACS.
0181    1F          376           DEFB    RXIA+SIOSAV+TXI+EXTI
0182    06          377           DEFB    SIOWR6          ;ADDRESS
0183    9E          378           DEFB    ADDRESS
0184    07          379           DEFB    SIOWR7          ;FLAG
0185    7E          380           DEFB    01111110B
0186    05          381           DEFB    SIOWR5          ;TX PARAMS.
0187    EB          382           DEFB    DTR+TX8+TXEN+RTS+TXCRC
0188    03          383           DEFB    SIOWR3          ;RX PARAMS.
0189    C1          384           DEFB    RX8+RXEN
                    385  SIOEB:   EQU     $
                    386
                    387  CLST:
018A    28          388           DEFB    28H             ;PORT B MODE
018B    00          389           DEFB    00000000B
018C    2B          390           DEFB    2BH             ;DATA DIRECTION
018D    EE          391           DEFB    11101110B
018E    1C          392           DEFB    1CH             ;CT1 MODE
018F    C2          393           DEFB    11000010B
0190    16          394           DEFB    16H             ;CT1 TC MSB
0191    00          395           DEFB    0
0192    17          396           DEFB    17H             ;       LSB
0193    60          397           DEFB    CIOCNT
0194    01          398           DEFB    1               ;MASTER CONFIG. REG.
0195    F0          399           DEFB    11110000B
0196    0A          400           DEFB    10              ;CT1 TRIGGER
0197    06          401           DEFB    00000110B
                    402  CEND:    EQU     $
                    403  *E
                    404
                    405  ;;       DATA AREA
                    406
2000                407           ORG     RAM
2000                408           DEFS    64              ;STACK AREA
                    409  STAK:    EQU     $
2040                410  SIOFLG:  DEFS    1               ;SIO FLAG BYTE
2041                411  BYTES:   DEFS    1               ;BUFFER BYTE COUNT
2042                412  RSPTMR:  DEFS    1               ;RESPONSE TIMER
2043                413  BUFPTR:  DEFS    2               ;BUFFER POINTER
2045                414  BUFFER:  DEFS    64              ;BUFFER
2085                415  RBPTR:   DEFS    2               ;READ BUFF PTR
                    416  RBUF:    EQU     $
                    417
                    418           END
```

Binary Synchronous Communication Using the Z80 SIO

Zilog

Application Note

October 1980

A popular communication protocol used to exchange information between data processing devices has been in use for some time. This protocol, developed by IBM, is called binary synchronous protocol, or bisync. The Z80 SIO provides a flexible and powerful tool for the implementation of the bisync protocol. However, there are some design considerations that require special attention. This paper will discuss these design considerations and offer an approach to using bisync with the Z80 SIO. Specific examples are presented and readers who are unfamiliar with the bisync protocol should refer to the ANSI standard (1) or the IBM publication (2) listed at the end of this paper.

Bisync is a character-oriented protocol with information transmitted in blocks between two (or more) data communication devices. The medium through which this information is conveyed is called the data link. The particular data link discussed in this paper is a point-to-point link using the ASCII transmission code. Other codes, such as EBCDIC, are not covered, but the format for bisync is basically the same. The data link consists of a master station (usually a computer) and a slave station (usually a terminal) with the associated communication gear in between--modems, phone lines, etc. The master station controls message flow by polling and selecting the slave station. Polling involves sending a general request message to the slave station(s) to determine whether or not any of the slaves have data to send (traffic). If a slave station does have traffic, it responds to the poll and the master can then select that particular slave for information exchange. Slaves can only respond to a master device and cannot initiate communication on the data link.

Information is exchanged by means of a well-defined block structure. Message blocks consist of a header, body, and trailer (Figure 1). The header is made of two or more SYN characters (hence the name bisync), a start of header (SOH) character, and addressing and control information for a particular slave station.

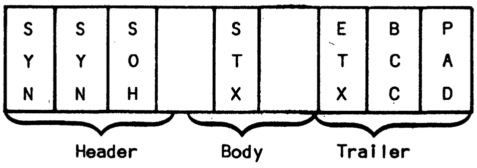


Figure 1. Basic Message Block Format for Bisync Protocol

The body begins with a start of text (STX) character and encompasses the entire text information. The body generally contains ASCII text data, although 8-bit binary data can be transmitted using transparent text mode.

The trailer contains the end of text (ETX) character and the block check character (BCC). The BCC is used for detecting errors through "cyclic redundancy checking" (CRC) or "longitudal redundancy checking" (LRC).

Error detection is essential when transferring information between data processing equipment. Since ASCII specifies only seven bits for its code, the eighth bit is used for vertical redundancy checking (VRC), more commonly known as character parity. In synchronous communications, character parity is generally odd, whereas in asynchronous communications it is even. Figure 2 shows typical ASCII characters with parity. The SIO can be programmed for 7-bit characters with odd parity enabled to minimize software overhead.

This application note refers to products as Z80 "A", "B" etc. to specifiy the speed grade. We are no longer using those characters for the speeds. For more details, please refer to the ordering information section.

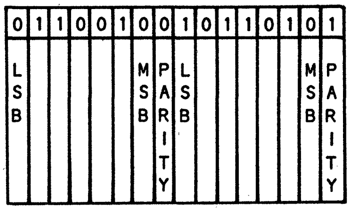


Figure 2. Odd VRC. Number of 1s should be odd.

Because VRC applies only to the individual character, the entire message block has an LRC that makes up the BCC. The LRC is a simple bit position checksum where the number of 1s for each position (0 through 6) is even for a block of data. Since the BCC is a character, LRC is subject to the same character parity rules as the rest of the data block. The LRC includes all characters, except SYN, starting with the first character after SOH or STX and up to and including ETX in the trailer (Figure 3). Since the SIO cannot calculate the LRC, the task is left up to the user. LRC can be generated on a microprocessor with little effort by taking the message block and XORing the data with an initial value of zero to provide even LRC.

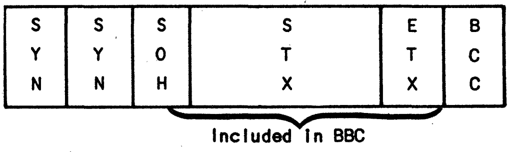


Figure 3. Characters Included in BBC

Another type of BBC is generated by a cyclic redundancy check (CRC), which results in a more powerful method of block checking. CRC-12 is used for 6-bit transmission code and CRC-16 is used for 8-bit transmission code. CRC is used in lieu of character parity and LRC, as with transparent text mode operation.

The remainder of this paper illustrates how to use the SIO in three special cases of the bisync protocol: transparent text mode, abort/interrupt procedures, and error recovery procedures.

Transparent text mode is useful in bisync when information exchanged between master and slave is not ASCII data. For example, a binary data file (object program) might be sent from master to slave. ASCII transmission code is only seven bits long making it difficult to send 8-bit binary data. One alternative is to convert the binary data to ASCII hex format at the master, transmit it to the slave and reconvert it back into binary at the slave. However, two disadvantages result from this. First, the master and slave require a means of conversion, by either software or hardware, adding cost to the data link. Since the slave (terminal) is burdened most by this, such an approach is usually not feasible. The other disadvantage is that the exchange of information is slower since two (or more) ASCII characters are sent for every eight bits of binary data. The bisync protocol has provisions for sending 8-bit binary data by using transparent text mode transmission. In this mode, character parity is disabled, allowing the full eight bits to be used for data. However, to allow control within the constraints of the protocol, there are certain limitations on the binary data pattern. The primary difference is that during transparent mode some communication control characters are preceded by a DLE character, actually making the control characters a two-character sequence. To distinguish a data byte from a control DLE, the protocol specifies insertion of another DLE. The receiver then throws away the first DLE, keeping the second as data. Table 1 shows the communication control characters that are valid during transparent mode.

Another character change occurs when the SYN character is used for line fill. Normally, the SYN character is ignored, but during transparent mode the SYN is preceded by a DLE, and both are consequently ignored by the receiver. In the event that the CPU does not have a character ready to send, the SIO automatically inserts SYN characters into the data stream. With the SIO programmed for 16-bit sync characters, two syncs are sent from the SIO (write registers WR6 and WR7) when its transmit buffer is empty. In transparent mode, the user must change WR6 and WR7 to DLE, SYN in order for the SIO to provide the proper line fill characters. In accordance with the ANSI standard, line fill characters are not included in the SIO CRC calculation during transmit. During reception in transparent mode, the software must disable CRC accumulation when the DLE SYN character sequence is detected.

While in transparent mode, the user must be concerned with the error detection codes. If parity is enabled in the SIO normally, it must be disabled during transparent mode. This change in SIO operation affects both transmit and receive and should therefore be considered if using full duplex.

Table 1. Control Codes Used In Transparent Mode

DLE	STX	Start of transparent text
DLE	ETB	End of transparent text block
DLE	ETX	End of transparent text
DLE	SYN	Idle sync
DLE	ENQ	Enquiry
DLE	DLE	DLE data
DLE	SOH	Start of transparent header

Since the SIO allows CRC enable/disable on the fly, the software can easily control CRC accumulation in both receive and transmit. During transmit, the CRC must be enabled/disabled before the character is transferred into the serial shift register. During receive, the CRC accumulation is delayed eight bits. After the character is transferred from the serial shift register into the buffer, the user has to read that character, decide whether or not to continue CRC accumulation, and disable/enable CRC before the next character is transferred to the buffer. This is not generally a problem, since character transfers occur about every 833 microseconds at 9600 baud. Table 2 shows the characters included and omitted in the CRC during transparent mode.

Table 2. Characters Included/Omitted in CRC During Transparent Mode

Omitted from CRC		Included in CRC
DLE	SYN	DLE of DLE DLE
DLE	SOH	ETX of DLE ETX
DLE	STX*	ETB of DLE ETB
		STX of DLE STX**

*If not preceded by transparent header within same block

**If preceded by DLE SOH within same block

When CRC accumulation is to be resumed, the software should enable CRC before the desired character is transferred to the receive buffer. For example, suppose a DLE pair is received during transparent text mode. The SIO generates an interrupt when the first DLE is transferred to the receive buffer. The driver program reads the DLE and immediately disables CRC. When the next interrupt occurs, the driver reads the second DLE and immediately enables CRC to include the second DLE into the CRC accumulation.

The second category of interest includes abort and interrupt procedures. There are two types of aborts: block abort and sending station abort. There are three types of interrupts: termination interrupt, reverse interrupt and temporary interrupt.

The block abort is used by the sending station when, in the process of transmitting a data block, the sending station detects an error condition in the data and decides to terminate the block so that the receiving station will discard it. In nontransparent mode, block abort is accomplished by ending the block with an ENQ character, instead of ETX or ETB. The sending station then waits for a reply from the receiver, which should be a NAK. The transparent mode procedure is identical except that a DLE ENQ character sequence is used. Since a block abort puts the data link back in nontransparent mode, NAK is the valid response the receiver should send in both transparent and nontransparent modes.

The sending station abort is similar to the block abort, except that the sending station does not necessarily do a block abort but simply ends the current message block, waits for a response or timeout, and then sends an EOT to regain control of the data link. The sending station abort is useful when transmission to a particular receiver is necessary due to a higher priority message, buffer overflow condition, error detection, etc. Once the sending station abort sequence is made, the master can perform any data link control function.

From the receiver side, a termination interrupt causes the sending station to stop transmission. Such a procedure is useful when the receiver cannot accept any more data or incurs an error condition, such as paper jam, card jam, hardware error, etc. To accomplish a termination interrupt, the receiving station sends an EOT instead of the normal response. The EOT resets all stations on the link and allows the master to issue any control sequence.

The reverse interrupt (RINT) is used when the receiving station needs to transmit during reception of several message blocks. The RINT occurs when a receiver detects a valid CRC or LRC and, instead of returning an ACK, sends a DLE "<" character sequence to signal an affirmative acknowledgement and to stop transmission of data. Some exceptions and a more detailed description of RINT can be found in the ANSI standard.

The temporary interrupt procedure, WACK (Wait Before Sending Positive Acknowledge), is used by the receiving station to indicate positive acknowledgement and an inability to receive more data. Such a response may be necessary when the receiving station cannot accept data continuously, such as during a printing operation. The WACK consists of a DLE ";" character sequence and is sent in place of an ACK or ACKn. The sending station then sends ENQs (Enquiry) until the receiving station stops sending WACKs. The sending station can resume transmitting data when the receiving station sends an ACK or ACKn.

Recovery procedures provide a means of preventing data link instability. The recovery mechanism consists mainly of timers, grouped into four basic areas, and a NAK counter. The NAK counter is used to prevent repeated NAKs from inhibiting further communications. The sending unit counts how many NAKs it receives for a particular data block so that after a predetermined number of retries, it can recover and pursue another course of action. The particular count value and course of action taken when the count expires are left up to the user.

Four timers (timer A or response timer, timer B or receiver timer, timer C or gross timer, and timer D or no activity timer) prevent the data link from getting "hung" or going idle for extended periods of time. Generally, the shortest interval is used with timer A, and the longest interval is used with timer D. For maximum system efficiency, however, the receiver timer (timer B) should timeout before the response timer (timer A). The particular implementation of these timers varies from system to system, and some flexibility of exact timer values is left up to the user.

Since it is assumed that interrupts will be used with the SIO, an interrupt driven receiver timer count is kept in memory and is reinitialized each time a character is received (receive interrupt). The same applies for the response timer, except that when a timeout occurs, the transmit driver has several options to follow.

If the SIO is set to transmit CRC on transmit underrun, then the driver could simply set its flags and not fill the buffer. This allows a normal exit, since the SIO will then send its CRC bytes. If the SIO is set to not transmit CRC on transmit underrun, then it sends sync characters (SYN SYN or DLE SYN, whichever was last written to WR6 and WR7) until the transmit buffer is filled or transmit data is set to marking.

In any event, enough time must be allowed after CRC is sent so that the receiver can properly decode CRC. Because of the character delay in the SIO during CRC accumulation, about 20 clock cycles are necessary after the last CRC byte is sent to ensure adequate decoding time. (See the SIO Technical Manual for further details.) The SIO could be programmed to send pad characters either by disabling parity and sending 8-bit FFs (hex) or by filling WR6 and WR7 with FF hex. If enabled, the SIO automatically sends whatever is in its sync registers upon transmit underrun. Multiple message blocks do not have to be separated by pad characters as long as CRC is valid for the previous message block. However, to insure adequate time for the receiver to process CRC, it is recommended that at least two pad characters follow the last character of a block.

Using the SIO for the bisync protocol is fairly straightforward. Care should be exercised when using the SIO in transparent text mode, but the implementation is greatly simplified by the SIO's flexibility, as compared to other serial communications ICs. The CRC capabilities of the SIO provide a powerful means of maintaining maximum data integrity with minimum software overhead. Coupled with the DMA and the interrupt capabilities of the Z80 processor, the user will find the SIO an excellent choice in serving data communication needs.

(1) American National Standards Institute. ANSI X3.28 - 1976.

(2) "General Information - Binary Synchronous Communications." Pub. number GA27-3004-2.

Serial Communication with the Z80A DART

Zilog

Application Brief

January 1981

INTRODUCTION

Serial data communication is among the most widely used forms of exchanging information with and between computers. The rapid expansion of this form of communication has created the need for low-cost, efficient, and flexible peripheral devices that provide the user with a wide variety of options. The Z80 DART is designed to fill this need by providing two independently programmable, asynchronous communication channels for a Z80-based system.

This application brief describes the use of the Z80 DART in a Z80-based system. Further information on the Z80 CPU and Z80 DART is available in the Zilog Data Book (document number 00-2034-A), Z8400 Z80 CPU Product Specification (document number 00-2001-A), and the Z8470 Z80 DART Product Specification (document number 00-2044-A).

HARDWARE

The hardware for this application consists of a Z8400 Z80 CPU, Z8470 Z80A DART, Z8536 CIO, 4K ROM, and 4K RAM. Figure 1 shows a block diagram of the system. The CIO supplies the bit rate clock for the DART and allows the baud rate for each channel to be determined by the software.

The DART-to-CPU interface consists of eight bidirectional data lines, seven control lines, and three daisy chain interrupt control lines. The data lines are used to transfer data between the DART and the CPU. The direction of data flow on the data lines is determined through the use of the $\overline{CE}$, $\overline{RD}$,

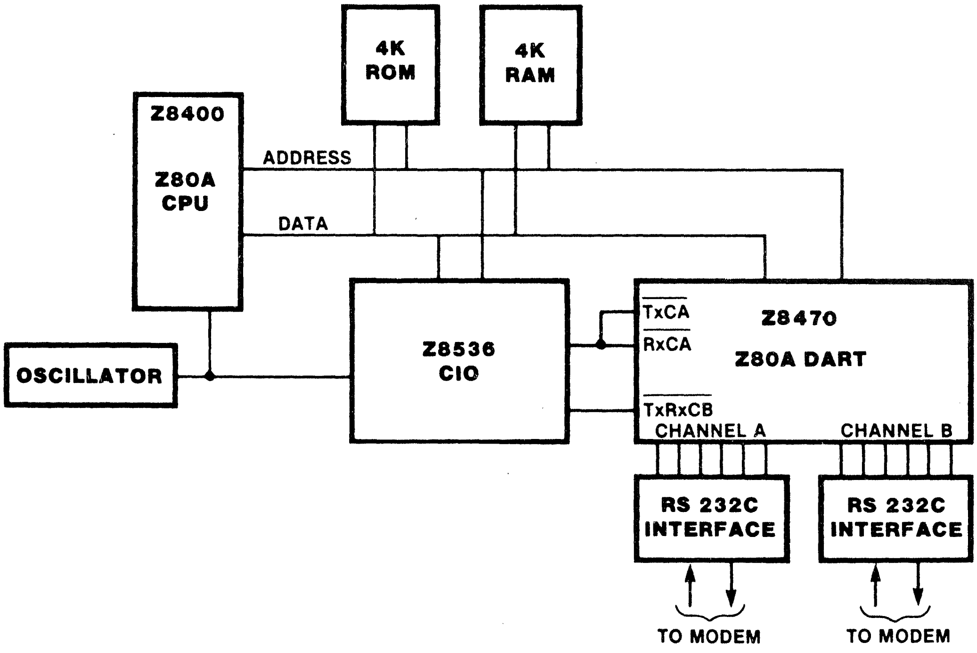


Figure 1. Z80 System Block Diagram

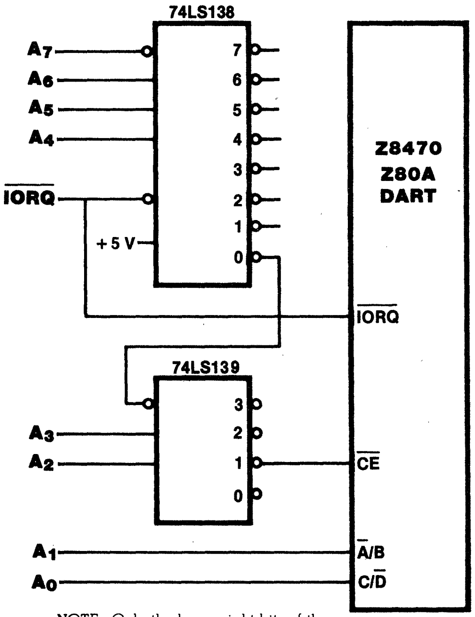
This application note refers to products as Z80 "A", "B" etc. to specifiy the speed grade. We are no longer using those characters for the speeds. For more details, please refer to the ordering information section.

and $\overline{IORQ}$ control lines. When $\overline{CE}$ and $\overline{IORQ}$ are active, a data transfer occurs between the CPU and DART. If $\overline{RD}$ is active at the same time, data is sent from the DART to the CPU. If $\overline{RD}$ is not active, data is sent from the CPU to the DART. $\overline{M1}$ signals an interrupt acknowledge cycle from the CPU in conjunction with $\overline{IORQ}$. The $\overline{RESET}$ line performs a device reset on the DART, allowing it to be placed in a known state. The remaining two control lines determine which of the four ports are being accessed. Table 1 shows the relationship of these two lines to the ports.

Table 1. DART Port Addressing

Port	C/$\overline{D}$	B/$\overline{A}$
Channel A Data	0	0
Channel B Data	0	1
Channel A Control	1	0
Channel B Control	1	1

C/$\overline{D}$ and B/$\overline{A}$ are usually tied to the lowest two CPU address lines used for I/O device selection. Figure 2 shows the device-select decode logic used in this application.



NOTE: Only the lower eight bits of the address bus are used for I/O select.

Figure 2. DART Device Select Logic

External connections to the Z80 DART include serial data and control lines and modem control lines. The serial data lines are Transmit Data (TxD) and Receive Data (RxD) for each channel. Separate transmit and receive clock inputs are available on channel A ($\overline{TxCA}$ and $\overline{RxCA}$), while a combined transmit/receive clock input is provided for channel B ($\overline{TxRxCB}$). To allow separate baud rates for both channels, $\overline{TxCA}$ and $\overline{RxCA}$ are tied together and connected to one counter/timer output, and $\overline{TxRxCB}$ is connected to another counter/timer output. This provides the user with a simple, software-programmable baud rate generator.

The modem control lines provide the user with a means of controlling some external device such as a modem. This is particularly useful for remote applications in which the CPU must determine a course of action based on the status of the modem control lines. For example, Ring Indicator ($\overline{RI}$) can be used to signal the CPU that an incoming call needs to be answered, or Data Terminal Ready (DTR) can be used in conjunction with Data Carrier Detect (DCD) to signal the modem that data communications can take place. DTR remains active as long as the DART is communicating over the serial data link. The CPU can "hang up" or disconnect the telephone connection by deactivating DTR. Finally, Request To Send (RTS) and Clear To Send (CTS) are useful in a multidrop configuration; that is, when three or more modems are connected to the same telephone line RTS is used to switch the carrier for a particular modem on or off under software control. CTS is monitored so that after RTS is activated the CPU knows when to start sending data.

The IEI, IEO, and $\overline{INT}$ lines form the Z80 daisy-chain interrupt controls that enable proper interrupt sequencing. $\overline{INT}$ is an open-drain, active Low output that is connected to the Z80 CPU $\overline{INT}$ input, along with a pullup resistor. IEI is usually connected to the preceding device in the daisy chain or is tied High if there is no preceding device. IEO is connected to the following device in the daisy chain or is left open. This application example uses interrupts with the Status Affects Vector (SAV) programming option. Interrupts are prioritized internally in the DART according to the various conditions. There are four separate interrupt groups for each channel. Table 2 shows the relative priorities of these interrupts.

Table 2. DART Interrupt Priority

Priority	Function
Highest	Ch. A Special Rx Condition
	Ch. A Rx Char. Available
	Ch. A Tx Buffer Empty
	Ch. A External/Status Change
	Ch. B Special Rx Condition
	Ch. B Rx Char. Available
	Ch. B Tx Buffer Empty
Lowest	Ch. B External/Status Change

PROGRAMMING

Programming the Z80 DART consists of two parts: initialization and program operation. Initialization includes defining the operating characteristics of the DART. This is done by writing a series of bytes to the control port of each channel. A detailed description of the programming for the DART can be found in the DART Product Specification (document number 00-2044-A). A listing containing an initialization routine for the DART can be found in the appendix of this brief.

Once initialized, the DART interrupts the CPU for certain conditions that occur. These conditions include Transmit Buffer Empty, Receive Character Available, Special Receive Condition, and External/Status Change for each channel.

The DART generates a Transmit Buffer Empty (TBE) interrupt when a character is transferred from the internal buffer to the shift register. The interrupt service routine determines whether to send another character to the DART or to issue a Reset Tx Interrupt Pending command. If a character is loaded into the DART, the interrupt condition is automatically removed. If a character is not loaded, the software issues a Reset Tx Interrupt Pending command to remove the interrupt condition and also sets an internal program status flag that signals the transmit channel as inactive. When transmission starts from an inactive condition (such as after initialization), the main program must activate the transmitter by sending a character to the DART. In this application, a call to the transmit interrupt service routine activates the transmitter after the buffer and pointers have been initialized.

The Receive Character Available (RCA) interrupt occurs after the DART transfers a character from the serial shift register to the receiver FIFO. The DART can store up to three characters in the FIFO, giving the CPU some flexibility in receive interrupt timing. Read Register 0 (RR0, bit 0) can be checked to see if any more characters are in the FIFO before exiting the interrupt service routine. If the DART is programmed so that parity does not affect the interrupt vector, parity errors must be checked in the receive service routine. This is done by writing a register pointer to the DART for Read Register 1 (RR1) and then reading the contents. The bit test instructions of the Z80 CPU are particularly useful in determining which bits are set or cleared. Processing for these errors is the same as processing for the Special Receive Condition.

The DART generates a Special Receive Condition (SRC) interrupt if it detects a parity error, overrun, or framing error during reception. When this occurs the programmer should reset the error condition by issuing an Error Reset command to the DART. After the Error Reset command is issued, the programmer should read and discard the data if necessary. If the data is not discarded, then an RCA interrupt occurs immediately after exiting the SRC service routine.

An External/Status Change (ESC) interrupt occurs when the DART detects a change in the external signals (RI, CTS, DCD) or when a receive break condition is initiated or terminated. This is useful in monitoring the interface to the modem where a software flag is set when the break condition is detected and reset when the break condition is cleared. With CTS, DCD, and RI, the same procedure is followed as with a break condition. However, if the auto enable bit is set in the DART, the DART does not transmit data until CTS becomes active, nor does it receive data until DCD becomes active.

The appendix contains the listing of a test program for the DART. While it is by no means complete, it does highlight the interrupt features of the Z80 DART.

CONCLUSION

As do other Z80 peripheral products, the Z80 DART interfaces well with the Z80 CPU. The software required to utilize the features of the DART is conducive to efficient programming. Interrupts provide a key method of maintaining efficient system operation, keeping CPU processing overhead to a minimum.

Other methods of utilizing the DART include a "polled" (noninterrupt) system. Because the Z80 CPU has three interrupt modes, the DART can be used with the CPU without vectored interrupts. However, such simplicity is usually at the expense of program size and speed.

Nevertheless, the user will find the Z80 DART a viable alternative to more expensive devices when considering the asynchronous communication requirements for any Z80 system.

APPENDIX

Following is the listing of a DART test program. Note that all interrupt service routines are dummy routines, except DATBE, which transfers characters from the buffer to Port A transmitter.

```
                                 TEST. DART
 LOC    OBJ CODE M STMT SOURCE STATEMENT                        ASM 5.9

                    73  ;;        *** MAIN PROGRAM ***
                    74
0000                75            ORG     0
0000    C32000      76            JP      BEGIN           ;GO MAIN PROGRAM
                    77
                    78  ;         INTERRUPT VECTORS
                    79
0010                80            ORG     $.AND.OFFFOH.OR.10H
                    81  INTVEC:
                    82  DRTVEC:
0010    7E00        83            DEFW    DBTBE
0012    9000        84            DEFW    DBESC
0014    8A00        85            DEFW    DBRCA
0016    A400        86            DEFW    DBSRC
0018    B800        87            DEFW    DATBE
001A    D100        88            DEFW    DAESC
001C    CB00        89            DEFW    DARCA
001E    E500        90            DEFW    DASRC
                    91
                    92  BEGIN:
0020    318320      93            LD      SP,STAK         ;INIT SP.
0023    ED5E        94            IM      2               ;VECTOR INTERRUPT MODE
0025    3E00        95            LD      A,INTVEC/256    ;UPPER VECTOR BYTE
0027    ED47        96            LD      I,A
0029    CD4800      97            CALL    INIT            ;INIT DEVICES
002C    210020      98            LD      HL,BUFFER
002F    063E        99            LD      B,62
                   100  LOOP:
0031    78         101            LD      A,B
0032    F640       102            OR      40H
0034    77         103            LD      (HL),A
0035    23         104            INC     HL
0036    10F9       105            DJNZ    LOOP
0038    360D       106            LD      (HL),13         ;CR
003A    23         107            INC     HL
003B    360A       108            LD      (HL),10         ;LF
003D    210020     109            LD      HL,BUFFER
0040    224120     110            LD      (BUFPTR),HL
0043    CDB800     111            CALL    DATBE           ;WAKE TX
                   112
0046    18FE       113            JR      $               ;LOOP FOREVER
                   114
                   115  INIT:
                   116  DRTINI:
0048    211001     117            LD      HL,DRTTA        ;INIT CH. A
004B    0E05       118            LD      C,DRTCA
004D    060A       119            LD      B,DRTEA-DRTTA
004F    EDB3       120            OTIR
0051    211A01     121            LD      HL,DRTTB        ;INIT CH. B
0054    0E07       122            LD      C,DRTCB
0056    060C       123            LD      B,DRTEB-DRTTB
0058    EDB3       124            OTIR
005A    AF         125            XOR     A               ;CLEAR FLAG BYTE
005B    324020     126            LD      (DRTFLG),A
                   127  CIOINI:
005E    DB0B       128            IN      A,(CIOCTL)      ;INSURE STATE 0
0060    AF         129            XOR     A               ;POINT TO REG 0
0061    D30B       130            OUT     (CIOCTL),A
0063    DB0B       131            IN      A,(CIOCTL)
0065    AF         132            XOR     A
0066    D30B       133            OUT     (CIOCTL),A
0068    3C         134            INC     A               ;WRITE RESET
0069    D30B       135            OUT     (CIOCTL),A
006B    AF         136            XOR     A               ;ELSE, CLEAR RESET COND.
006C    D30B       137            OUT     (CIOCTL),A
006E    3EFE       138            LD      A,0FEH          ;(FUDGE FOR CIO QUIRK)
0070    D30B       139            OUT     (CIOCTL),A
0072    D30B       140            OUT     (CIOCTL),A
0074    212601     141            LD      HL,CLST         ;INIT CIO
0077    0620       142            LD      B,CEND-CLST
0079    0E0B       143            LD      C,CIOCTL
```

```
007B   EDB3                     OTIR
007D   C9                       RET
                   *E

                   ;;           SUBROUTINES

                   ;            SETUP FOR ASYNC AS:
                   ;                    9600 BAUD
                   ;                    2 STOP BITS
                   ;                    EVEN PARITY
                   ;                    7 BIT CHARACTERS

                   ;            DRTFLG -  X X 1 1 X X 1 1
                   ;                        /   !     !
                   ;                    ERROR ASLEEP ERROR ASLEEP
                   ;                      CHANNEL B    CHANNEL A

                   DBTBE:
007E   CDF900                   CALL    SAVE            ;CH.B TX BUFFER EMPTY
0081   3E00                     LD      A,DRTWRO        ;POINT TO REG. 0
0083   D307                     OUT     (DRTCB),A
0085   3E28                     LD      A,TBERES        ;RESET TBE
0087   D307                     OUT     (DRTCB),A
0089   C9                       RET

                   DBRCA:
008A   CDF900                   CALL    SAVE            ;CH.B RX CHAR AVAIL.
008D   DB06                     IN      A,(DRTDB)       ;READ DATA
008F   C9                       RET

                   DBESC:
0090   CDF900                   CALL    SAVE            ;CH.B EXTERNAL/STATUS
0093   3E00                     LD      A,DRTWRO        ;POINT TO REG. 0
0095   D307                     OUT     (DRTCB),A
0097   3E10                     LD      A,ESCRES        ;RESET ESC
0099   D307                     OUT     (DRTCB),A
009B   3A4020                   LD      A,(DRTFLG)      ;UPDATE FLAG
009E   CBE7                     SET     4,A
00A0   324020                   LD      (DRTFLG),A
00A3   C9                       RET

                   DBSRC:
00A4   CDF900                   CALL    SAVE            ;CH.B SPECIAL RX COND.
00A7   3E00                     LD      A,DRTWRO
00A9   D307                     OUT     (DRTCB),A
00AB   3E30                     LD      A,SRCRES        ;RESET SRC
00AD   D307                     OUT     (DRTCB),A
00AF   3A4020                   LD      A,(DRTFLG)      ;UPDATE FLAG
00B2   CBEF                     SET     5,A
00B4   324020                   LD      (DRTFLG),A
00B7   C9                       RET

                   DATBE:
00B8   CDF900                   CALL    SAVE            ;CH.A TX BUFFER EMPTY
00BB   2A4120                   LD      HL,(BUFPTR)     ;GET BUFFER PTR.
00BE   46                       LD      B,(HL)          ;GET CHAR.
00BF   7D                       LD      A,L             ;UPDATE PTR.
00C0   3C                       INC     A
00C1   E63F                     AND     3FH             ;64 BYTE WRAPAROUND
00C3   6F                       LD      L,A
00C4   224120                   LD      (BUFPTR),HL
00C7   78                       LD      A,B             ;OUTPUT CHAR.
00C8   D304                     OUT     (DRTDA),A
00CA   C9                       RET

                   DARCA:
00CB   CDF900                   CALL    SAVE            ;CH.A RX CHAR AVAIL.
00CE   DB04                     IN      A,(DRTDA)
00D0   C9                       RET

                   DAESC:
00D1   CDF900                   CALL    SAVE            ;CH.A EXTERNAL/STATUS
```

```
00D4   3E00        216             LD      A,DRTWR0
00D6   D305        217             OUT     (DRTCA),A
00D8   3E10        218             LD      A,ESCRES
00DA   D305        219             OUT     (DRTCA),A
00DC   3A4020      220             LD      A,(DRTFLG)
00DF   CBC7        221             SET     0,A
00E1   324020      222             LD      (DRTFLG),A
00E4   C9          223             RET
                   224
                   225   DASRC:
00E5   CDF900      226             CALL    SAVE            ;CH. B SPECIAL RX COND.
00E8   3E00        227             LD      A,DRTWR0
00EA   D305        228             OUT     (DRTCA),A
00EC   3E30        229             LD      A,SRCRES
00EE   D305        230             OUT     (DRTCA),A
00F0   3A4020      231             LD      A,(DRTFLG)
00F3   CBCF        232             SET     1,A
00F5   324020      233             LD      (DRTFLG),A
00F8   C9          234             RET
                   235
                   236   ;         MATHEWS SAVE REGISTER ROUTINE
                   237
                   238   SAVE:
00F9   E3          239             EX      (SP),HL         ;SP =   HL
00FA   D5          240             PUSH    DE              ;       DE
00FB   C5          241             PUSH    BC              ;       BC
00FC   F5          242             PUSH    AF              ;       AF
00FD   DDE5        243             PUSH    IX              ;       IX
00FF   FDE5        244             PUSH    IY              ;       IY
0101   CD0F01      245             CALL    GO              ;       PC
0104   FDE1        246             POP     IY
0106   DDE1        247             POP     IX
0108   F1          248             POP     AF
0109   C1          249             POP     BC
010A   D1          250             POP     DE
010B   E1          251             POP     HL
010C   FB          252             EI
010D   ED4D        253             RETI
                   254
                   255   GO:
010F   E9          256             JP      (HL)
                   257   *E
                   258
                   259   ;;        CONSTANTS
                   260
                   261   DRTTA:
0110   00          262             DEFB    DRTWR0          ;CHAN.  RESET
0111   18          263             DEFB    CHRES
0112   01          264             DEFB    DRTWR1          ;CHAN.  CHARACS.
0113   13          265             DEFB    RXIAP+TXI+EXTI
0114   04          266             DEFB    DRTWR4          ;MODE
0115   4F          267             DEFB    X16+STOP2+EVEN+PARITY
0116   05          268             DEFB    DRTWR5          ;TX PARAMS.
0117   AA          269             DEFB    DTR+TX7+TXEN+RTS
0118   03          270             DEFB    DRTWR3          ;RX PARAMS.
0119   41          271             DEFB    RX7+RXEN
                   272   DRTEA:    EQU     $
                   273
                   274   DRTTB:
011A   00          275             DEFB    DRTWR0          ;CHAN.  RESET
011B   18          276             DEFB    CHRES
011C   01          277             DEFB    DRTWR1          ;CHAN.  CHARACS.
011D   17          278             DEFB    RXIAP+DRTSAV+TXI+EXTI
011E   02          279             DEFB    DRTWR2          ;VECTOR REG.
011F   10          280             DEFB    DRTVEC.AND.255
0120   04          281             DEFB    DRTWR4          ;MODE
0121   4F          282             DEFB    X16+STOP2+EVEN+PARITY
0122   05          283             DEFB    DRTWR5          ;TX PARAMS.
0123   AA          284             DEFB    DTR+TX7+TXEN+RTS
0124   03          285             DEFB    DRTWR3          ;RX PARAMS.
0125   41          286             DEFB    RX7+RXEN
```

```
                        287  DRTEB:  EQU     $
                        288
                        289  CLST:
0126     28             290          DEFB    28H              ;PORT B MODE
0127     00             291          DEFB    00000000B
0128     2B             292          DEFB    2BH              ;DATA DIRECTION
0129     EE             293          DEFB    11101110B
012A     06             294          DEFB    6                ; "       "    PORT C
012B     0E             295          DEFB    00001110B
012C     1C             296          DEFB    1CH              ;CT1 MODE
012D     C2             297          DEFB    11000010B
012E     1D             298          DEFB    1DH              ;CT2 MODE
012F     C2             299          DEFB    11000010B
0130     1E             300          DEFB    1EH              ;CT3 MODE
0131     C2             301          DEFB    11000010B
0132     16             302          DEFB    16H              ;CT1 TC MSB
0133     00             303          DEFB    0
0134     17             304          DEFB    17H              ;       LSB
0135     06             305          DEFB    CIOCNT
0136     18             306          DEFB    18H              ;CT2 TC MSB
0137     00             307          DEFB    0
0138     19             308          DEFB    19H              ;       LSB
0139     06             309          DEFB    CIOCNT
013A     1A             310          DEFB    1AH              ;CT3 TC MSB
013B     00             311          DEFB    0
013C     1B             312          DEFB    1BH              ;       LSB
013D     06             313          DEFB    CIOCNT
013E     01             314          DEFB    1                ;MASTER CONFIG. REG.
013F     F0             315          DEFB    11110000B
0140     0A             316          DEFB    10               ;CT1 TRIGGER
0141     06             317          DEFB    00000110B
0142     0B             318          DEFB    11               ;CT2 TRIGGER
0143     06             319          DEFB    00000110B
0144     0C             320          DEFB    12               ;CT3 TRIGGER
0145     06             321          DEFB    00000110B
                        322  CEND:   EQU     $
                        323  *E
                        324
                        325  ;;      DATA AREA
                        326
2000                    327          ORG     RAM
2000                    328  BUFFER: DEFS    64
2040                    329  DRTFLG: DEFS    1
2041                    330  BUFPTR: DEFS    2
2043                    331          DEFS    64               ;STACK AREA
                        332  STAK:   EQU     $
                        333
                        334          END
```

Interfacing 8500 Peripherals To The Z80

Zilog

Application Brief

December 1980

INTRODUCTION

There are several differences between the 8500 devices and the Z80 family peripheral devices, including interrupt handling, reset to the device, and daisy-chain control.

This application brief describes the hardware interface requirements and interrupt structure of the 8500 series peripherals in Z80 systems. The 8500 peripherals are general-interface versions of the Z-BUS counterparts and are designed to interface to nonmultiplexed buses (such as in a Z80 system), instead of multiplexed buses (such as in the Z8000).

CPU HARDWARE INTERFACING

The hardware interface consists of three basic groups of signals: the data bus, control and selection lines, and the interrupt control lines. Following is a table of the general interface signals used by the CPU. Additional information can be found in the peripherals' separate data sheets.

DATA BUS

D_0-D_7	Data bus, bidirectional, 3-state. This bus is used to transfer data between the CPU and the peripheral device.

CONTROL SIGNALS

A_0-A_n	Address select lines (optional). These lines are normally used to select the port and/or control registers.
CE	Chip Enable. $\overline{CE}$ should be gated with $\overline{IORQ}$ or $\overline{MREQ}$ to prevent spurious chip selects during other machine cycles.
$\overline{RD}$*	Read. $\overline{RD}$ activates chip-read circuitry and gates data from chip onto data bus (to be read by the CPU).
$\overline{WR}$*	Write. $\overline{WR}$ is used to strobe data from bus into chip.

INTERRUPT CONTROL

$\overline{INTACK}$	Interrupt acknowledge signal from CPU. This replaces the $\overline{M1}$ and $\overline{IORQ}$ generated by the Z80 CPU for interrupt acknowledge. It is used in conjunction with $\overline{RD}$ to gate the interrupt vector onto the data bus.
$\overline{INT}$, IEI IEO	Interrupt Request, Interrupt Enable Input and Interrupt Enable Output. These lines are functionally equivalent to those in the Z80 peripheral products. $\overline{INT}$ is open-drain, active Low output.

*Chip reset is accomplished by activating $\overline{RD}$ and $\overline{WR}$ simultaneously.

INTERRUPT OPERATION

Understanding the 8500 interrupt operation requires basic operational knowledge of the Interrupt Pending (IP) and Interrupt Under Service (IUS) bits in relation to the daisy chain. IP is set in the SIO by an interrupt condition, such as the transmit buffer going empty, and is used with IUS to control the $\overline{INT}$ signal. IP is not set while the CPU is executing an interrupt acknowledge cycle. Thus,

$$IP = INT * \overline{VREAD}$$

The IP latch is cleared either by a software command to the device or by an implicit action generated by the interrupt service routine. The implicit action may be triggered by the CPU reading or writing a register in the device. For example, on a serial receive device like the SIO, IP may be reset when the CPU reads the character from the receive buffer that caused the interrupt. This removes the interrupt condition, allowing other interrupts to occur.

The Interrupt Under Service (IUS) latch is used to designate the interrupt that is

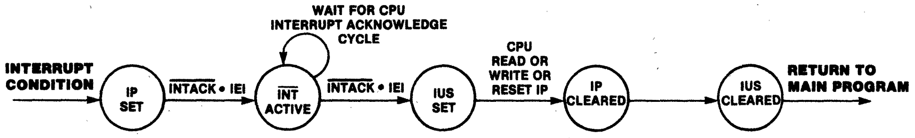
This application note refers to products as Z80 "A", "B" etc. to specifiy the speed grade. We are no longer using those characters for the speeds. For more details, please refer to the ordering information section.

currently being serviced. IUS is set when the device receives an interrupt acknowledge from the CPU while IEI is High and IP is set. If IEI is Low, the device is prevented from setting the IUS latch and thus cannot issue a vector. In this way, the daisy chain can establish relative priority among peripheral devices. IUS is cleared on the 8500 devices by an explicit software command.

The daisy chain used in the Z80 peripherals is referred to as an IP and IUS daisy chain, because the IP and IUS bits control the IEO pin and the lower portion of the chain. If IP is set, IEO can be Low even if another peripheral has an interrupt under service. When the CPU executes an RETI instruction (ED-4D opcode), the peripheral monitors the bus and resets IUS. When the CPU reads the "ED" part of RETI, peripherals with IP set and IEI High bring IEO High momentarily. This enables the device in the chain with IUS set to clear its IUS latch when the "4D" byte is read by the CPU. (IUS for a device is not cleared unless IEI is High and the "ED-4D" instruction is decoded. This allows more than one device to have IUS set so that nested interrupts can be implemented.)

On the 8500 series devices, IP is used to control the daisy chain only during the interrupt acknowledge cycle. Under normal conditions, only IUS is required to control the state of the IEO pin. Therefore, the daisy chain used in 8500 devices is referred to as an IUS daisy chain. Since IP is not a part of the daisy chain, there is no "ED" decoding pulling IEO High when IP is set. To allow more control over the daisy chain, the 8500 devices have a "Disable Lower Chain" (DLC) software command that unconditionally brings IEO Low. This can be used to deactivate parts of the daisy chain selectively, regardless of interrupt status. Figure 1 shows the functions of IP and IUS and the truth tables for each.

A unique feature of the 8500 devices is the $\overline{\text{INTACK}}$ pin. This pin acknowledges a CPU interrupt service cycle to the peripheral, allowing the peripheral to gate its vector onto the data bus. On the Z80 peripherals, interrupt acknowledge cycles from the CPU consist of a special M1 cycle where $\overline{\text{IORQ}}$ is activated instead of $\overline{\text{MREQ}}$. This limits the control of devices in systems using a processor other than the Z80. As a result, a simpler implementation has been devised, which uses additional logic to accommodate a wider variety of processors. Figure 2 shows a circuit that generates $\overline{\text{INTACK}}$ for the 8500 devices in addition to wait states. Figure 3 shows the timing for $\overline{\text{INTACK}}$ and wait generation.



INTERRUPT CONDITION
IP SET
$\overline{\text{INTACK}}$ • IEI
WAIT FOR CPU INTERRUPT ACKNOWLEDGE CYCLE
$\overline{\text{INT}}$ ACTIVE
$\overline{\text{INTACK}}$ • IEI
IUS SET
CPU READ OR WRITE OR RESET IP
IP CLEARED
IUS CLEARED
RETURN TO MAIN PROGRAM

a) State diagram of 8500 devices during interrupt cycle.

IEI	IP	IUS	IEO
0	X	X	0
1	1	0	1
1	0	1	0
1	0	0	1

b) 8500 device during idle state.

IEI	IP	IUS	IEO
0	X	X	0
1	1	0	0
1	0	1	0
1	0	0	1

c) 8500 device during $\overline{\text{INTACK}}$ cycle.

Figure 1. 8500 Device Interrupt-Processing Sequence

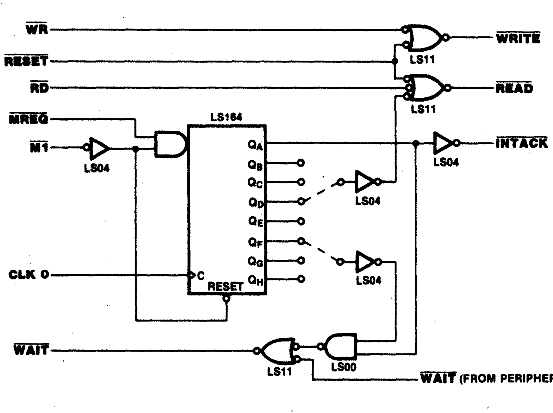
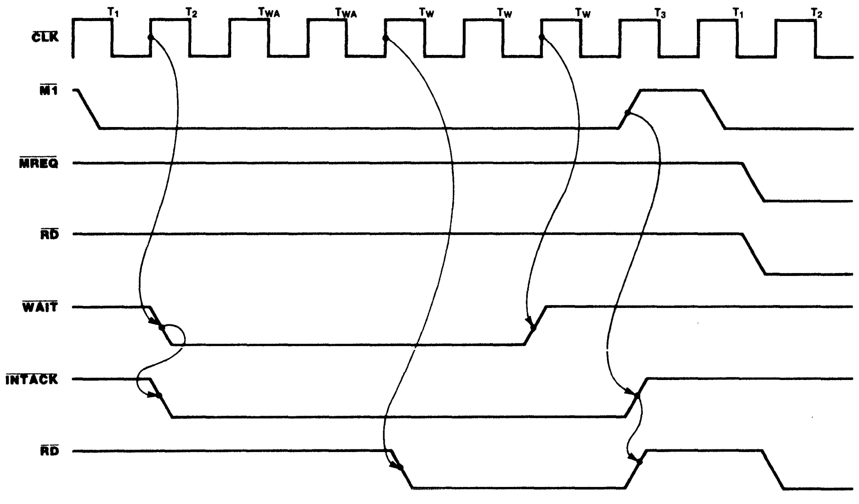


Figure 2. INTACK and WAIT Generation for 8500 Peripherals



NOTE: WAIT' is assumed to be High.

Figure 3. Timing for 8500 Peripherals During Interrupt Acknowledge Without Z80 Peripheral Logic

On long daisy chains, wait states may be necessary to allow the IEI and IEO lines time to stabilize, thus avoiding conflict between devices and preventing IUS or IP from changing erroneously. Because of the IP and IUS configurations, the daisy chain used in Z80 peripherals needs to stabilize during the interrupt acknowledge and RETI operations. However, on the 8500 devices, the daisy chain is IUS and wait states are generated for the INTACK cycle only, not for the return cycle. (There is no "ED-4D" decode.) As a result, hardware interfacing is greatly simplified and timing is less complicated than on the Z80 peripherals.

SOFTWARE CONSIDERATIONS

There are several options available for servicing interrupts on the 8500 devices. Since the vector register (or IP register) can be read at any time, the software can emulate the Z80 CPU interrupt response easily. The interrupt vector reflects the interrupt status condition, even if the peripheral is programmed to return a vector that does not reflect the status change (SAV or VIS not set). This allows a simple software routine to emulate the Z80 vector response operation, as shown in the code of Figure 4.

```
                                 AP.8500.1
Loc. Obj Code   M  Stmt Source  Statement

                       *E

                       ;        This routine emulates the Z80 vector interrupt
                       ;        operation by reading the device interrupt vector,
                       ;        forming an address from a vector table, and exe-
                       ;        cuting an indirect jump to the interrupt service
                       ;        routine.

0000  3E00             INDX:    LD     A,CIVREG     ;CURRENT INT. VECTOR REG
0002  D3E0                      OUT    (CTRL),A     ;WRITE REG. PTR.
0004  DBE0                      IN     A,(CTRL)     ;READ VECTOR REG.
0006  3C                        INC    A            ;VALID VECTOR?
0007  C8                        RET    Z            ;NO INTERRUPT - RETURN
0008  E60E                      AND    00001110B    ;MASK OTHER BITS
000A  5F                        LD     E,A          ;FORM INDEX VALUE
000B  1600                      LD     D,0
000D  211600    R               LD     HL,VECTAB    ;ADD VECTOR TABLE ADDR
0010  19                        ADD    HL,DE
0011  7E                        LD     A,(HL)       ;GET LOW BYTE
0012  23                        INC    HL
0013  66                        LD     H,(HL)       ;GET HIGH BYTE
0014  6F                        LD     L,A          ;PUT ROUTINE ADDR IN $HL
0015  E9                        JP     (HL)         ;GO TO ROUTINE !

                       VECTAB:
0016  0010                      DEFW   INT1
0018  0011                      DEFW   INT2
001A  0012                      DEFW   INT3
001C  0013                      DEFW   INT4
001E  0014                      DEFW   INT5
0020  0015                      DEFW   INT6
0022  0016                      DEFW   INT7
0024  0017                      DEFW   INT8
```

Figure 4. Z80 Vector Interrupt Response Emulation by Software

Because the 8500 devices have considerable program flexibility, a Master Interrupt Enable (MIE or IE) bit in the control register determines the device response to the CPU. If MIE is not set, interrupts are not generated to the CPU and the device ignores any interrupt response from the CPU. This is used as a global enable and simplifies the programming of interrupts so that they can be easily changed on the fly.

A SIMPLE Z80 SYSTEM

The 8500 devices interface easily to the Z80 CPU, providing a system of considerable flexibility. Figure 5 illustrates a simple system using the Z80 CPU and a Z8536 CIO in a noninterrupt environment. Since $\overline{\text{INTACK}}$ is not used, it is tied High and no additional logic is needed. Because the CIO can be used in a polled interrupt system, the $\overline{\text{INT}}$ pin is connected to the CPU. The Z80 should not be programmed for Interrupt Mode 2, because the vector from the CIO is never sent to the CPU. Instead, the CPU can be set for Interrupt Mode 1, and a global interrupt routine that reads the vector register from the CIO can determine which routine to go to when an interrupt occurs, as previously illustrated in Figure 4.

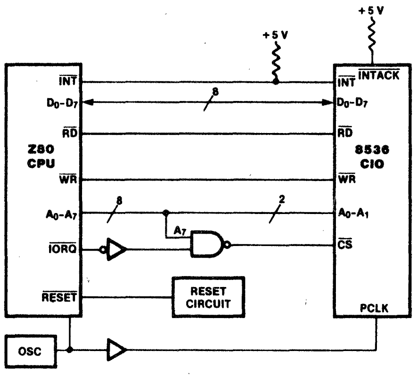
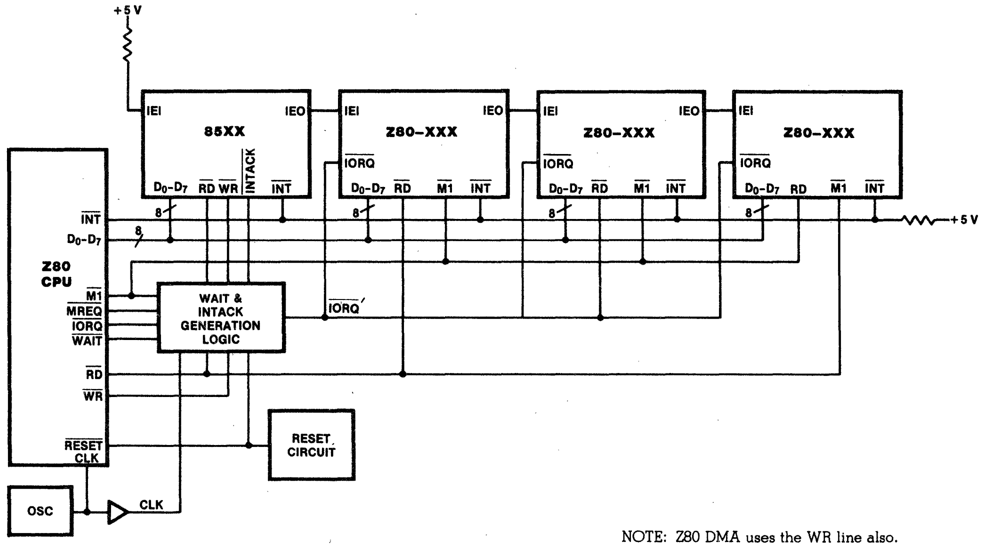


Figure 5. Non-Interrupt CPU Interface

Z80 PERIPHERALS WITH 8500 PERIPHERALS

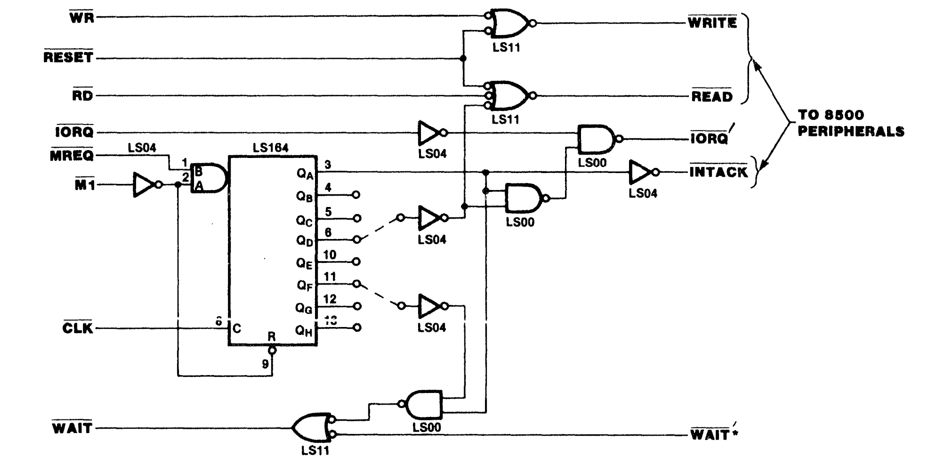
A Z80 system using a combination of Z80 family peripherals and 8500-type peripherals is easily constructed, as shown in Figure 6. There is no placement restriction on the 8500 devices within the daisy chain, but it is recommended that they be near the beginning of the chain in order to minimize propagation delays during the "ED-4D" decoding. The 8500 devices do not decode the "ED" during an opcode fetch cycle, so IEO will not change state during this time.



NOTE: Z80 DMA uses the WR line also.

Figure 6. A Z80 System Using 8500 Devices and Z80 Peripherals

Figure 7 is a diagram of the logic represented by the WAIT and INTACK logic box in Figure 6. The $\overline{\text{WAIT}}'$signal is OR-wired to the output of each peripheral device (if used). The $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals only go to the 8500 device. The Z80 peripherals are wired to the Z80 as usual. The timing for the $\overline{\text{INTACK}}$ and $\overline{\text{WAIT}}$ generation logic is illustrated in Figure 8.



*Wait from peripheral devices

Figure 7. $\overline{\text{WAIT}}$ and $\overline{\text{INTACK}}$ Generation Logic

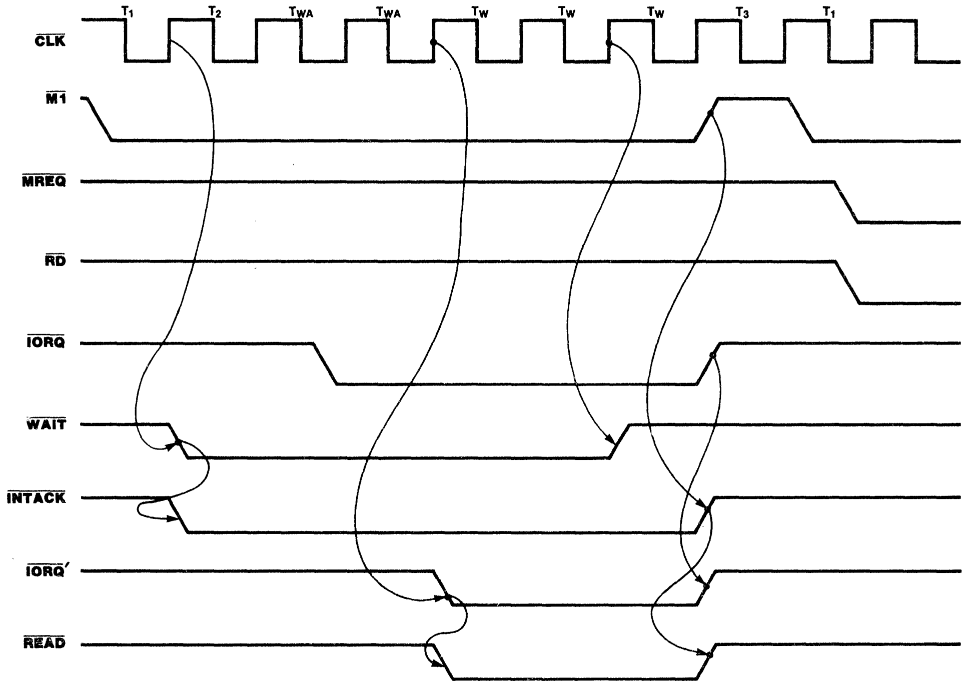


Figure 8. Timing for 8500 and Z80 Peripherals During Interrupt Acknowledge

Serial Clock Generation Using the Z8536 CIO

Zilog

Application Brief

February 1981

INTRODUCTION

When an external clock is not provided in a Z80-based system, it is often necessary to generate a bit-rate clock for serial devices. The most efficient way to accomplish this is to use a programmable counter that can change the bit-rate clock under CPU control. In this example, the Z8536 Counter/Timer I/O device (CIO) was chosen to generate the bit-rate clocks for a Z80-based statistical multiplexor project that used a Z80 SIO and a Z80 DART.

This application brief describes the use of the Z8536 CIO device in a Z80-based system for generating the bit-rate clocks for asynchronous communications. The Z8536 CIO contains the circuitry necessary to generate the clock pulses required by asynchronous communication devices.

HARDWARE

The Z8536 CIO is housed in a 40-pin package and contains both system bus interface and I/O port connections. The three 16-bit counters can be programmed to output a pulse, square wave, or one-shot waveform on the timer's corresponding output pin. Three bits of the output ports (two from Port B and one from Port C) are used as the counter/timer outputs and provide the bit-rate pulses used in this application.

Interfacing the CIO to the Z80 CPU requires eight bidirectional data lines and five control lines. The data lines are used to transfer register address and data to or from the CIO via the $\overline{RD}$, $\overline{WR}$, $\overline{CE}$, and address control lines. Two address lines (A_0 and A_1) select the port the CPU is accessing. Table 1 shows the port selected by the address bits.

Table 1. Port Addressing for the CIO

Address Line	A_1	A_0
Port C	0	0
Port B	0	1
Port A	1	0
CTRL	1	1

The control port (CTRL) is used for control register selection and parameter transfer. To select a particular register, a Register Pointer is written to the CTRL port and the data is written into or read from the register.

The CIO contains a state machine that controls the CPU interface. Upon power-up, the CIO is placed in a reset state and remains there until cleared by the program. Reset can also be initiated by issuing a command to Register 0 with bit 0 set or by a hardware condition ($\overline{RD}$ and $\overline{WR}$ simultaneously active). The reset state is described in detail in the programming section. Once the reset state is cleared, the CIO is placed in state 0, in which the control registers can be accessed by writing a Register Pointer to the CIO control port. This places the CIO in state 1, after which the next CPU access (read or write register data) causes the CIO to revert to state 0. The last register addressed may be accessed simply by reading the CIO control port. It should be noted that the Register Pointer can be written only while in state 0. Also, data can be written to a control register only after a Register Pointer has been written. Figure 1 shows the state diagram for the CIO.

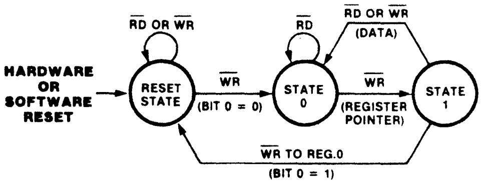


Figure 1. State Diagram for Z8536 CIO

The $\overline{RD}$ and $\overline{WR}$ control lines determine the data path direction into or out of the CIO. When activated simultaneously, they also perform the device's reset function. Figure 2 illustrates how the reset function can be implemented using external circuitry.

This application note refers to products as Z80 "A", "B" etc. to specifiy the speed grade. We are no longer using those characters for the speeds. For more details, please refer to the ordering information section.

Since interrupts are not used in this application, INTACK is tied High to prevent spurious interrupt operation of the CIO due to noise.

Each counter/timer uses one or more bits on one of the parallel ports to provide for counter input and counter/timer output. Table 2 shows which output port bits correspond to particular counter/timer inputs and outputs.

The outputs of the counter/timers (PB4, PB0, and PC0) are fed to the rest of the circuitry to supply the serial clock pulses.

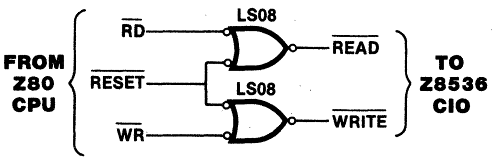


Figure 2. RESET Interface to the Z8536

Table 2. Counter/Timer External Interface Bits

Function	C/T1	C/T2	C/T3
C/T Output	PB4*	PB0	PC0
Counter Input	PB5	PB1	PC1
Trigger Input	PB6	PB2	PC2
Gate Input	PB7	PB3	PC3

*PB4 = Port B, bit 4

The last hardware consideration involves the clock input, PCLK. Since the Z8536 does not need to be synchronized with the CPU clock, PCLK can come from any source so long as it meets the timing and interface requirements. In fact, PCLK can come from a source external to the system if desired. Once inside the device, PCLK is divided by two before it is sent to the counter/timer circuits. There is no other prescaling done and the resulting clock is fed to the 16-bit counters.

PROGRAMMING

Once the hardware has been defined, the functional operation and configuration of the Z8536 are determined entirely by the software programming. Several considerations concerning initialization must be made when using the CIO. When the device receives a reset from either hardware or a software command, the reset state must be removed before any data can be written to the CIO. To clear the reset state, the user writes to register 0 with bit 0 cleared. Once the internal reset latch is cleared, the programmer can initialize the CIO and begin normal operations. The program listed in the appendix shows a reset sequence that brings the CIO to state 0 even if the previous state is undefined.

The configuration of the CIO defines the general operating characteristics of the device with respect to its internal functions. The Port Mode Specification register sets to output those bits in Port B that are used for the counter/timer outputs. In this example, Bit mode is used on Ports B and C to output the counter/timer pulses.

The Counter/Timer mode, time constant values, and trigger commands are the last parameters to be set. Finally, the Master Configuration Control register is set to enable Port B, all the counter/timers, and Port C (Port C is enabled along with the counter/timers). The Counter/Timer mode is programmed for continuous cycle square wave with external output enabled. The square-wave cycle time is two times the programmed time constant, which must be taken into account when programming time constant values. The downcounters in the CIO are 16-bit counters that are decremented by one for each internal clock cycle. The internal clock cycle is the PCLK cycle divided by two, so the time constant value is determined by the following formula:

Time Constant= PCLK / (4 * Output Frequency)

PCLK is divided by four in the formula because it is divided by two inside the CIO before being fed into the downcounter and by two again because a square wave cycle is two times the time constant value. Substituting the baud rate and a multiplier of 16 for the output frequency, the formula reduces to a simple time constant formula.

TC = PCLK / (4 * 16 * Baud Rate)

With a 3.6864 MHz PCLK input and a desired 9600 baud rate, the formula simplifies to:

$$\begin{aligned} TC &= 3{,}686{,}400 / (4 * 16 * 9600) \\ &= 57600 / 9600 \\ &= 6 \end{aligned}$$

Other 16X baud rates may be generated by using the above formula in a general form.

TC = 57600 / Baud Rate

The user must exercise caution when choosing values for the PCLK and baud rates since they must result in nearly integral time constant values. For example, a 2.4576 MHz clock input with 9600 baud and a 16X clock output give a time constant value of 4. Greater flexibility is available for selecting time constant values because the SIO does not require a square wave input when programmed for 16X, 32X, or 64X clock inputs. Pulses may be used with the SIO provided the user adheres to the SIO timing requirements.

The last operation performed on the CIO is a trigger command to "kick it off." This also includes setting the gate command bit in the Counter/Timer Command and Status registers, which allows the clock pulses to toggle the downcounter. The trigger command bit loads an initial value into the downcounter and begins operation of the counter/timer circuitry. Once triggered, the counter/timer runs continuously, performing automatic reloads to the downcounter after it reaches zero (terminal) count. At this time, the CIO is finished being programmed and the user has three clean square waveforms at the output pins.

CONCLUSION

The designer should find the Z8536 CIO a versatile and cost-effective component to satisfy his or her system needs. Coupled with other Zilog components, the Z8536 architecture enhances the performance of any Z80 system by providing the essential timing, I/O functions, and interrupt control functions necessary for efficient system operation.

The Z8536 CIO was chosen after considering device count, performance, and ease of use. Alternatives to the CIO include discrete (TTL) hardware counters and gates, external clock sources, or the Z80 CTC. These methods are generally too parts-intensive, and power consumption is therefore higher. For applications where two 8-bit ports and three counter/timers are needed, the CIO proves to be the ideal component.

APPENDIX

Following is a listing of a test program written for the Z80 CPU. This program simply initializes the CIO and then loops until stopped, with the CIO continuously providing pulses. All three counter/timers are used to generate square waves corresponding to a 16X 9600 baud clock.

```
                                TEST.CIO
 LOC   OBJ CODE M STMT SOURCE STATEMENT                              ASM 5.9

                   1  ;          CIO TEST PROGRAM
                   2  ;[1]       01-07-81/MDP            INITIAL CREATION
                   3
                   4  ;          THIS PROGRAM INITIALIZES THE THREE COUNTER
                   5  ;          TIMERS IN THE Z8536 CIO TO GENERATE SQUARE
                   6  ;          WAVES, THEN LOOPS FOREVER.
                   7
                   8  ;          PROGRAM EQUATES
                   9
                  10  CIOC:      EQU     8                ;CIO PORT C
                  11  CIOB:      EQU     CIOC+1           ;CIO PORT B
                  12  CIOA:      EQU     CIOC+2           ;CIO PORT A
                  13  CIOCTL:    EQU     CIOC+3           ;CIO CTRL PORT
                  14  BAUD:      EQU     9600             ;ASYNC BAUD RATE
                  15  RATE:      EQU     BAUD/100
                  16  CIOCNT:    EQU     576/RATE
                  17  RAM        EQU     2000H            ;RAM START ADDR
                  18  RAMSIZ     EQU     1000H            ;RAM SIZE
                  19  *E
                  20
                  21  ;          *** MAIN PROGRAM ***
                  22
 0000             23             ORG     0
                  24  BEGIN:
 0000  314020     25             LD      SP,STAK          ;INIT SP.
 0003  CD0800     26             CALL    INIT             ;INIT DEVICES
                  27
 0006  18FE       28             JR      $                ;LOOP FOREVER
                  29
                  30  INIT:
                  31  CIOINI:
 0008  DB0B       32             IN      A,(CIOCTL)       ;INSURE STATE 0
 000A  3E00       33             LD      A,0              ;REG 0 OR RESET
 000C  D30B       34             OUT     (CIOCTL),A       ;WRITE PTR OR CLEAR RESET
 000E  DB0B       35             IN      A,(CIOCTL)       ;STATE 0
 0010  3E00       36             LD      A,0              ;REG 0
 0012  D30B       37             OUT     (CIOCTL),A       ;WRITE PTR
 0014  3E01       38             LD      A,1              ;WRITE RESET
 0016  D30B       39             OUT     (CIOCTL),A
 0018  3E00       40             LD      A,0              ;CLEAR RESET
 001A  D30B       41             OUT     (CIOCTL),A
 001C  212600     42             LD      HL,CLST          ;INIT CIO
 001F  0620       43             LD      B,CEND-CLST
 0021  0E0B       44             LD      C,CIOCTL
 0023  EDB3       45             OTIR
 0025  C9         46             RET
                  47  *E
                  48
```

```
                      49  ;;      CONSTANTS
                      50
                      51  CLST:
0026   28             52          DEFB     28H              ;PORT B MODE
0027   00             53          DEFB     00000000B
0028   2B             54          DEFB     2BH              ;PORT B DIRECTION
0029   EE             55          DEFB     11101110B
002A   06             56          DEFB     06H              ;PORT C DIRECTION
002B   FE             57          DEFB     11111110B
002C   1C             58          DEFB     1CH              ;CT1 MODE
002D   C2             59          DEFB     11000010B
002E   1D             60          DEFB     1DH              ;CT2 MODE
002F   C2             61          DEFB     11000010B
0030   1E             62          DEFB     1EH              ;CT3 MODE
0031   C2             63          DEFB     11000010B
0032   16             64          DEFB     16H              ;CT1 TC MSB
0033   00             65          DEFB     0
0034   17             66          DEFB     17H              ;        LSB
0035   06             67          DEFB     CIOCNT
0036   18             68          DEFB     18H              ;CT2 TC MSB
0037   00             69          DEFB     0
0038   19             70          DEFB     19H              ;        LSB
0039   06             71          DEFB     CIOCNT
003A   1A             72          DEFB     1AH              ;CT3 TC MSB
003B   00             73          DEFB     0
003C   1B             74          DEFB     1BH              ;        LSB
003D   06             75          DEFB     CIOCNT
003E   01             76          DEFB     1                ;MASTER CONFIG. REG.
003F   F0             77          DEFB     11110000B
0040   0A             78          DEFB     0AH              ;CT1 TRIGGER
0041   06             79          DEFB     00000110B
0042   0B             80          DEFB     0BH              ;CT2 TRIGGER
0043   06             81          DEFB     00000110B
0044   0C             82          DEFB     0CH              ;CT3 TRIGGER
0045   06             83          DEFB     00000110B
                      84  CEND:   EQU      $
                      85
                      86  ;;      DATA AREA
                      87
2000                  88          ORG      RAM
2000                  89          DEFS     64               ;STACK AREA
                      90  STAK:   EQU      $
                      91
                      92          END
```

Timing in an Interrupt-Based System with the Z80® CTC

Zilog

Application Note

March 1981

INTRODUCTION

In many computer systems, an accurate time base is needed so that critically timed events do not go awry. Use of a counter or timer to monitor time-dependent activities is essential in such systems. In an interrupt-driven system, the Z80 CTC can provide regular program time intervals. Single-event counts or single-event time delays can also be implemented under program control. This application note describes both continuous time-interval operations and single-interval count operations using the Z80 CTC in a Z80 system.

HARDWARE CONFIGURATION

In the example used here, the hardware consists of a Z80 CPU with 4K bytes of RAM, 4K bytes of ROM, a Z80A SIO, and a Z80A CTC. There are two external inputs to the CTC: one is derived from the ac power line to provide 60Hz pulses; the other is connected to a transmit clock line on the SIO. One of the counter/timer outputs is connected to the SIO transmit and receive clock input, as shown in Figure 1.

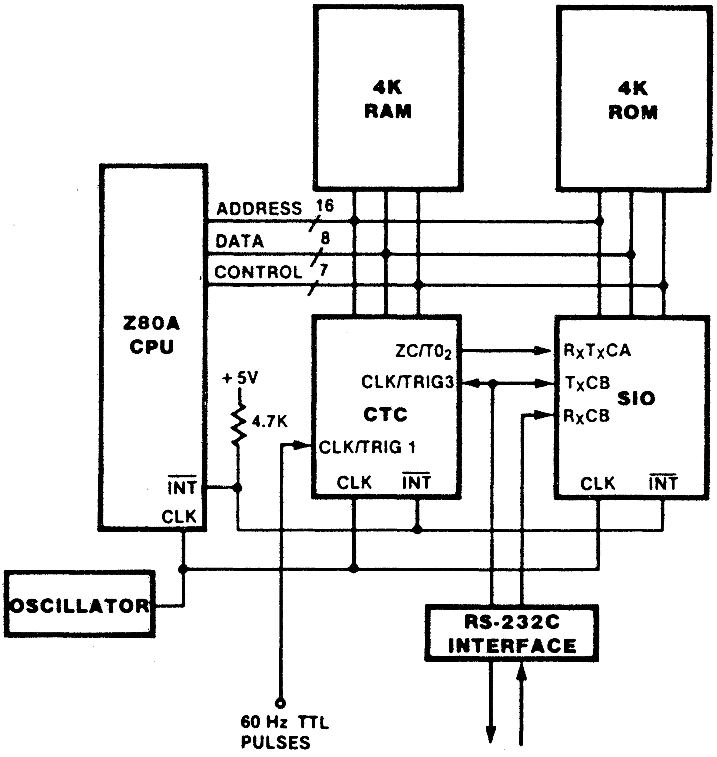


Figure 1. Z80A System Block Diagram

This application note refers to products as Z80 "A", "B" etc. to specifiy the speed grade. We are no longer using those characters for the speeds. For more details, please refer to the ordering information section.

The Z80 CTC is designed for easy interface to the Z80 CPU. An 8-bit bidirectional data bus is used to transfer information between the CTC and CPU. The control lines, $\overline{RD}$, $\overline{IORQ}$, $\overline{M1}$, and $\overline{CE}$, determine what data is being transferred and when. M1 and $\overline{IORQ}$ are used during the interrupt acknowledge cycle to allow the CTC to present its 8-bit interrupt vector to the CPU. $\overline{IORQ}$ is also used in conjunction with $\overline{CE}$ to enable transfers between the CTC and the CPU. $\overline{RD}$ is used to control the direction of data flow between the CTC and the CPU. The channel select lines (CS_0 and CS_1) are connected to the lowest two bits of the address bus and are used to access one of the four counter/timer channels. Table 1 shows the relationships between the CS pins and the counter/timer channels.

Table 1. Channel Select Values

CS_1	CS_0	C/T Channel
0	0	Channel 0
0	1	Channel 1
1	0	Channel 2
1	1	Channel 3

The CTC system clock input requirements are similar to those of the Z80 CPU. For both, the system clock input Low level should be no greater then 0.45 V, the High level should be no less than V_{cc}-0.6 V, and the clock rise and fall times should be less than 30 ns. A clock-driver device that meets these requirements, such as the HH-3006-A[1], works well with the CTC. Several devices can be connected to the driver, but the user should be careful not to overload the driver. The capacitance of the clock input to the CTC (20 pF) should be noted as this may affect the system clock rise and fall times.

Interrupt control logic within the CTC is used to initiate interrupts and to control the interrupt acknowledge cycle generated by the CPU. An interrupt is generated by the CTC when one of the counter/timer down counters reaches terminal count (0) and IEI is High. IEI and IEO allow the CTC to operate within the Z80 interrupt daisy chain and to connect to the next higher-priority and next lower-priority devices in the chain, respectively. If there is no higher-priority device, IEI is tied to +5 V.

The CTC internally prioritizes each counter/timer with respect to interrupt generation. This maximizes performance by resolving contention between channels should two or more interrupt conditions occur simultaneously. Table 2 shows the relative priority levels of each counter/timer within the CTC.

Table 2. CTC Channel Interrupt Priority

Priority	Channel
Highest	0
	1
	2
Lowest	3

CTC MODES

There are two basic modes under which the CTC can operate: Timer mode and Counter mode. Each mode has certain programmable characteristics that enable the CTC to be used in a wide variety of applications.

TIMER MODE

A typical use of the CTC in Timer mode is to provide regular, fixed-interval interrupts to the CPU used as a time-base reference to allocate the processor resources efficiently. For example, a multitasking system might have the processor execute a task for a given length of time and then interrupt execution of the program at one-second intervals to scan the task queue for higher-priority tasks. This system time interval can be provided by the CTC in Timer mode. In Timer mode, the CTC downcounter is decremented by the output of the prescaler, which is toggled by the system clock input. The prescaler has a programmable value of 16 or 256, depending on the condition of bit 5 in the channel control word (CCW). Thus, with a 4 MHz system clock fed into the CTC, a timer resolution of 4µs (prescaler count of 16) or 64µs (count of 256) is possible.

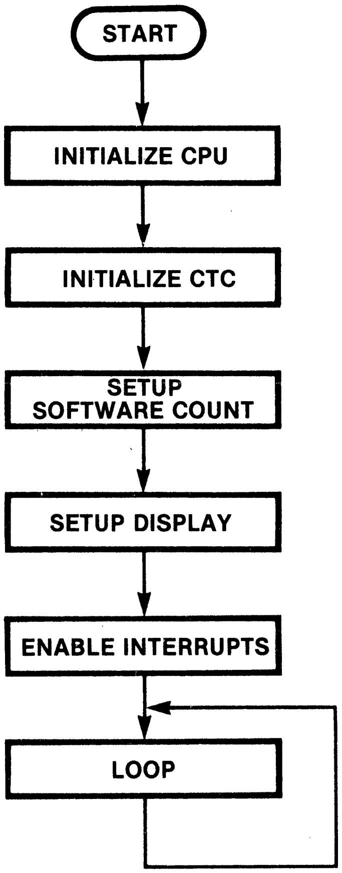
In the example shown, the interrupt interval is set to 8.33 ms, which is provided by the CTC with a 3.6864 MHz input clock, 256 prescaler value, and a time constant value of 120. The CTC interrupt service routine uses a software count of 120 to maintain a one-second system time interval. Each time the service routine is executed, the software count is decremented by 1. When the count reaches 0, a flag is set and the program pursues an appropriate course of action. Figure 2 shows the initialization and interrupt service routine coding for a CTC channel using the Timer mode.

Another use of CTC Timer mode operation is to implement a nonretriggerable one-shot using external circuitry. The digital approach to the one-shot provides a programmable time delay under CPU control and provides greater noise immunity than the more common analog delay circuits provide. Figure 3 shows a circuit that uses part of a 74LS02 package in addition to one CTC channel.

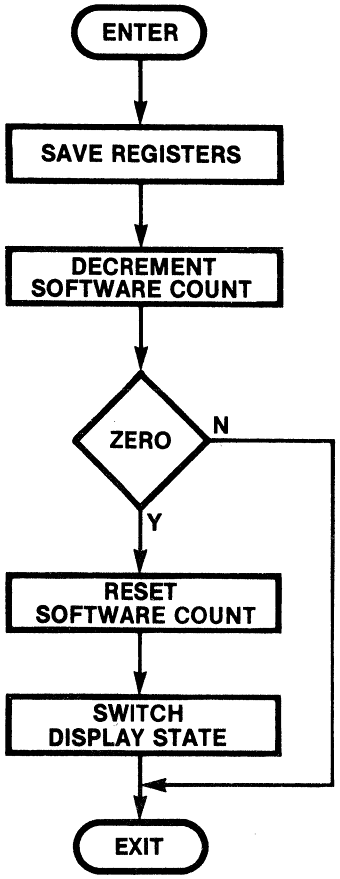
The trigger waveform should be positive-going and should meet the CTC setup time for the CLK/TRIG input. Also, the trigger High level time should be less than the CTC delay time in order to prevent the two 74LS02s from latching in the triggered state. An additional gate can be added to initialize the 74LS02 flip-flop to a defined state when the system is reset or else the software can pulse the timer output to set the flip-flop, as is done in this case. A third use of the Timer mode is to provide a bit rate clock for a serial transceiver device, such as the Z80 SIO. The SIO can accept a 1x, 16x, 32x, or 64x bit rate clock input from an external source, and with a 16x, 32x, or 64x multiplier, the SIO can accept a pulse waveform input for the bit rate clocks, as long as the pulses meet the rise, fall, and hold time requirements of the SIO. The CTC meets these requirements and can be connected directly to the SIO to provide the necessary bit rate clocks. Figure 4 shows the code needed to generate a bit rate clock for the SIO.

[1]A clock driver by Hybrid House, 1615 Remuda La., San Jose, CA 95112.

With a 1x bit rate clock programmed into the SIO, a square-wave input must be supplied. This can be done by adding a flip-flop between the CTC and the SIO. The time constant value should be set to half the baud rate value, since the CTC output is divided in half by the flip-flop.



a) Main Program



b) Interrupt Service Routine

Figure 2. Software for CTC Timer Mode Operation

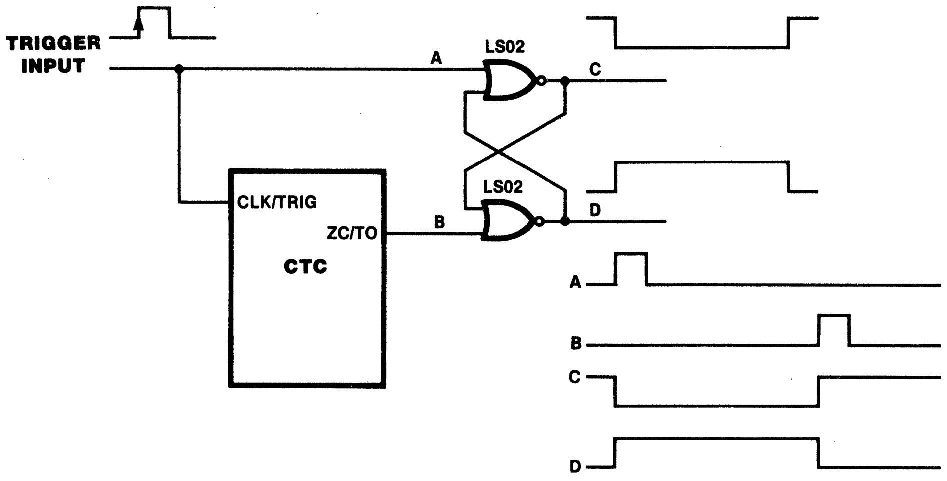


Figure 3. Monostable Multivibrator Using the Z80 CTC

```
                              TEST.CTCO
 LOC    OBJ CODE M STMT SOURCE STATEMENT

                         1  ;         CTC TEST PROGRAM
                         2
                         3  ;         THIS PROGRAM USES THE CTC IN CONTINUOUS
                         4  ;         TIMER MODE. THE CTC COUNTS SYSTEM CLOCK
                         5  ;         PULSES AND INTERRUPTS EVERY 120 PULSES,
                         6  ;         THEN DECREMENTS A COUNT, THEN SWITCHES
                         7  ;         THE LED STATE WHEN THE COUNT REACHES ZERO.
                         8
                         9  ;         PROGRAM EQUATES
                        10
                        11  CTCO:     EQU     12              ;CTC 0 PORT
                        12  CTC1:     EQU     CTCO+1          ;CTC 1 PORT
                        13  CTC2:     EQU     CTCO+2          ;CTC 2 PORT
                        14  CTC3:     EQU     CTCO+3          ;CTC 3 PORT
                        15  LITE:     EQU     OEOH            ;LIGHT PORT
                        16  RAM:      EQU     2000H           ;RAM START ADDR
                        17  RAMSIZ:   EQU     1000H
                        18  TIME:     EQU     120             ;COUNT VALUE
                        19
                        20
                        21  ;         CTC EQUATES
                        22
                        23  CCW:      EQU     1
                        24            INTEN:  EQU     80H
                        25            CTRMODE:        EQU     40H
                        26            P256:   EQU     20H
                        27            RISEDG: EQU     10H
                        28            PSTRT:  EQU     8
                        29            TCLOAD: EQU     4
                        30            RESET:  EQU     2
                        31  *E
                        32
                        33  ;;        *** MAIN PROGRAM ***
                        34
0000                    35            ORG     0
0000    C31800          36            JP      BEGIN
                        37
0010                    38            ORG     $.AND.OFFFOH.OR.10H
                        39  INTVEC:
0010    4000            40            DEFW    ICTCO
0012    3D00            41            DEFW    ICTC1
0014    3D00            42            DEFW    ICTC2
0016    3D00            43            DEFW    ICTC3
                        44
                        45  BEGIN:
0018    314020          46            LD      SP,STAK         ;INIT SP
001B    ED5E            47            IM      2               ;VECTOR INTERRUPT MODE
001D    3E00            48            LD      A,INTVEC/256    ;UPPER VECTOR BYTE
001F    ED47            49            LD      I,A
0021    CD2700          50            CALL    INIT            ;INIT DEVICES
0024    FB              51            EI                      ;ALLOW INTERRUPTS
                        52
0025    18FE            53            JR      $               ;LOOP FOREVER
                        54
                        55  INIT:
0027    3EA7            56            LD      A,INTEN+P256+TCLOAD+RESET+CCW
0029    D30C            57            OUT     (CTCO),A        ;SET CTC MODE
002B    3E78            58            LD      A,TIME
002D    D30C            59            OUT     (CTCO),A        ;SET TIME CONSTANT
002F    3E10            60            LD      A,INTVEC.AND.11111000B
0031    D30C            61            OUT     (CTCO),A        ;SET VECTOR VALUE
0033    AF              62            XOR     A
0034    324120          63            LD      (DISP),A        ;CLEAR DISPLAY BYTE
0037    3E78            64            LD      A,TIME          ;INIT TIMER VALUE
0039    324020          65            LD      (COUNT),A
003C    C9              66            RET
                        67  *E
                        68
                        69  ;         INTERRUPT SERVICE ROUTINE
                        70
```

```
                                      TEST.CTCO
 LOC   OBJ CODE M STMT SOURCE STATEMENT

                  71   ICTC1:
                  72   ICTC2:
                  73   ICTC3:
003D   FB         74            EI                        ;DUMMY ROUTINES
003E   ED4D       75            RETI
                  76
                  77   ICTCO:
0040   CD5A00     78            CALL    SAVE              ;SAVE REGISTERS
0043   3A4020     79            LD      A,(COUNT)         ;CHANGE TIMER COUNT
0046   3D         80            DEC     A
0047   324020     81            LD      (COUNT),A
004A   CO         82            RET     NZ                ;EXIT IF NOT DONE
004B   3E78       83            LD      A,TIME            ;ELSE, RESET TIMER VALUE
004D   324020     84            LD      (COUNT),A
0050   3A4120     85            LD      A,(DISP)          ;BLINK LITES
0053   2F         86            CPL
0054   324120     87            LD      (DISP),A
0057   D3EO       88            OUT     (LITE),A
0059   C9         89            RET
                  90
                  91   ;        SAVE REGISTER ROUTINE
                  92
                  93   SAVE:
005A   E3         94            EX      (SP),HL
005B   D5         95            PUSH    DE
005C   C5         96            PUSH    BC
005D   F5         97            PUSH    AF
005E   CD6800     98            CALL    GO
0061   F1         99            POP     AF
0062   C1        100            POP     BC
0063   D1        101            POP     DE
0064   E1        102            POP     HL
0065   FB        103            EI
0066   ED4D      104            RETI
                 105
                 106   GO:
0068   E9        107            JP      (HL)
                 108   *E
                 109
                 110   ;;       DATA AREA
                 111
2000             112            ORG     RAM
2000             113            DEFS    64                ;STACK AREA
                 114   STAK:    EQU     $
2040             115   COUNT:   DEFS    1                 ;TIMER COUNT VALUE
2041             116   DISP:    DEFS    1                 ;LITE DISPLAY BYTE
                 117
                 118            END
```

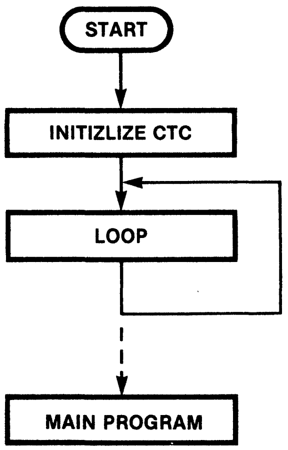


Figure 4. Software for CTC Bit Rate Generator

```
                              TEST.CTC2
LOC    OBJ CODE M STMT SOURCE STATEMENT
                     1  ;       CTC TEST PROGRAM
                     2
                     3  ;       THIS PROGRAM USES THE CTC IN CONTINUOUS
                     4  ;       TIMER MODE. THE CTC SUPPLIES A BIT RATE
                     5  ;       CLOCK TO THE SIO FROM THE SYSTEM CLOCK.
                     6  ;       THE SYSTEM CLOCK IS 3.6864 MHZ, WHICH IS
                     7  ;       DIVIDED BY 16 BY THE PRESCALER, AND DIVIDED
                     8  ;       BY A TIME CONSTANT VALUE OF 3 TO
                     9  ;       PROVIDE A 16X, 4800 BAUD CLOCK
                    10  ;       TO THE SIO. OTHER BAUD RATES CAN BE OBTAINED
                    11  ;       BY PROGRAMMING DIFFERENT TIME CONSTANT
                    12  ;       VALUES INTO THE CTC.
                    13
                    14  ;       PROGRAM EQUATES
                    15
                    16  CTC0:   EQU     12              ;CTC 0 PORT
                    17  CTC1:   EQU     CTC0+1          ;CTC 1 PORT
                    18  CTC2:   EQU     CTC0+2          ;CTC 2 PORT
                    19  CTC3:   EQU     CTC0+3          ;CTC 3 PORT
                    20  TIME:   EQU     3               ;TIME CONSTANT VALUE
                    21
                    22
                    23  ;       CTC EQUATES
                    24
                    25  CCW:    EQU     1
                    26          INTEN:  EQU     80H
                    27          CTRMODE:        EQU     40H
                    28          P256:   EQU     20H
                    29          RISEDG: EQU     10H
                    30          PSTRT:  EQU     8
                    31          TCLOAD: EQU     4
                    32          RESET:  EQU     2
                    33  *E
                    34
                    35  ;;      *** MAIN PROGRAM ***
                    36
0000                37          ORG     0
                    38  BEGIN:
0000   3E07         39          LD      A,TCLOAD+RESET+CCW
0002   D30E         40          OUT     (CTC2),A        ;SET CTC MODE
0004   3E03         41          LD      A,TIME
0006   D30E         42          OUT     (CTC2),A        ;SET TIME CONSTANT
                    43
                    44  ;       MAIN PROGRAM GOES HERE
                    45  *E
                    46
0008   18FE         47          JR      $               ;LOOP FOREVER
                    48
                    49          END
```

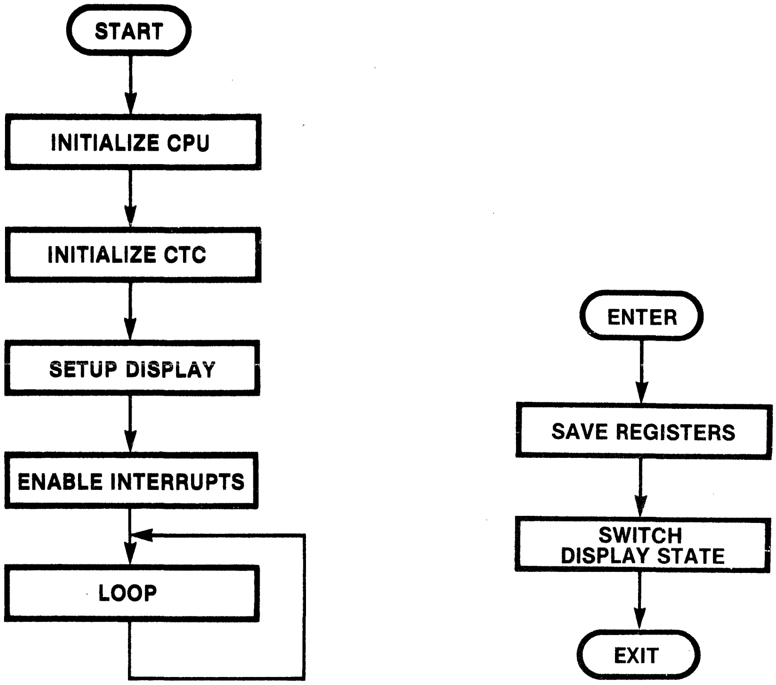
COUNTER MODE

A typical computer system often uses a time-of-day clock. In the United States, the 60 Hz power line provides an accurate time base for synchronous motor clocks. A computer system can take advantage of the 60 Hz accuracy by incorporating a circuit that feeds 60 Hz square waves into a CTC channel. With a time constant value of 60, the CTC generates an interrupt once every second, which can be used to update a time-of-day clock. The CTC is set to Counter mode and with a time constant value of 60, as shown in Figure 5.

The interrupt service routine does nothing more than update the time-of-day clock. A more sophisticated operating system kernel would use the CTC to check the task queue status. In synchronous data communications, it is often necessary to ensure that a flag or sync character separates two adjacent message packets. Since some serial controller devices have no way to determine the status of sync characters sent, the user must use time delays to separate messages with the appropriate number of sync characters. Typically, software or timer delays are used to provide the time necessary to allow the characters to shift out of the serial device. The disadvantage of using this method is that variable baud rates shift characters at variable times so a worst-case time must be allowed if the baud rate is not known. If the bit rate clock is supplied by the modem, as is normally the case, this problem becomes even more acute.

A solution to this problem is to use a counter to count the number of bits shifted out of the serial device. With the CTC tied to the transmit clock line of the serial device, the CTC can be programmed to delay a certain number of bits before the CPU sends another message. This solves all of the problems mentioned and simplifies the message-handling software. Figure 6 shows the program needed to achieve the counting function. Note that the interrupt service routine disables the CTC, because the CTC is used only once with each message. Otherwise, the CTC would generate an interrupt each time the counter reached terminal count.

Figure 1 shows the hardware implementation of the character delay counter using the CTC.



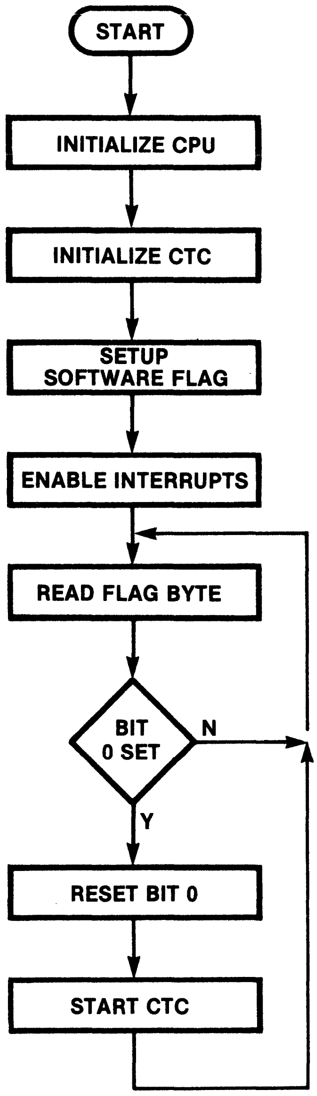
a) Main Program b) Interrupt Service Routine

Figure 5. Software for CTC Counter Mode

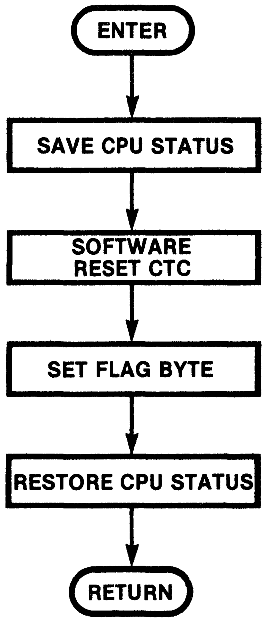
```
                                 TEST.CTC1
LOC    OBJ CODE M STMT SOURCE STATEMENT

                     1   ;        CTC TEST PROGRAM
                     2
                     3   ;        THIS PROGRAM COUNTS EXTERNAL PULSES AND
                     4   ;        CHANGES THE LED STATE EVERY 60 COUNTS
                     5
                     6   ;        PROGRAM EQUATES
                     7
                     8   CTC0:    EQU      12               ;CTC 0 PORT
                     9   CTC1:    EQU      CTC0+1           ;CTC 1 PORT
                    10   CTC2:    EQU      CTC0+2           ;CTC 2 PORT
                    11   CTC3:    EQU      CTC0+3           ;CTC 3 PORT
                    12   LITE:    EQU      0E0H             ;LIGHT PORT
                    13   RAM      EQU      2000H            ;RAM START ADDR
                    14   RAMSIZ   EQU      1000H
                    15   COUNT    EQU      60               ;COUNTER TIME CONSTANT
                    16
                    17
                    18            CTC EQUATES
                    19
                    20   CCW:     EQU      1
                    21            INTEN:   EQU      80H
                    22            CTRMODE:          EQU      40H
                    23            P256:    EQU      20H
                    24            RISEDG:  EQU      10H
                    25            PSTRT:   EQU      8
                    26            TCLOAD:  EQU      4
                    27            RESET:   EQU      2
```

```
                                      TEST. CTC1
 LOC    OBJ CODE M STMT SOURCE STATEMENT

                   28  *E
                   29
                   30  ;;        *** MAIN PROGRAM ***
                   31
0000               32            ORG      0
0000    C31800     33            JP       BEGIN
                   34
0010               35            ORG      $. AND. OFFFOH. OR. 10H
                   36  INTVEC:
0010    3800       37            DEFW     ICTCO
0012    3B00       38            DEFW     ICTC1
0014    3800       39            DEFW     ICTC2
0016    3800       40            DEFW     ICTC3
                   41
                   42  BEGIN:
0018    314020     43            LD       SP, STAK          ; INIT SP
001B    ED5E       44            IM       2                 ; VECTOR INTERRUPT MODE
001D    3E00       45            LD       A, INTVEC/256     ; UPPER VECTOR BYTE
001F    ED47       46            LD       I, A
0021    CD2700     47            CALL     INIT              ; INIT DEVICES
0024    FB         48            EI                         ; ALLOW INTERRUPTS
                   49
0025    18FE       50            JR       $                 ; LOOP FOREVER
                   51
                   52  INIT:
0027    3EC7       53            LD       A, INTEN+CTRMODE+TCLOAD+RESET+CCW
0029    D30D       54            OUT      (CTC1), A         ; SET CTC MODE
002B    3E3C       55            LD       A, COUNT
002D    D30D       56            OUT      (CTC1), A         ; SET TIME CONSTANT
002F    3E10       57            LD       A, INTVEC. AND. 11111000B
0031    D30C       58            OUT      (CTCO), A         ; SET VECTOR VALUE
0033    AF         59            XOR      A
0034    324020     60            LD       (DISP), A         ; CLEAR DISPLAY BYTE
0037    C9         61            RET
                   62  *E
                   63
                   64  ;         INTERRUPT SERVICE ROUTINE
                   65
                   66  ICTCO:
                   67  ICTC2:
                   68  ICTC3:
0038    FB         69            EI                         ; DUMMY ROUTINES
0039    ED4D       70            RETI
                   71
                   72  ICTC1:
003B    CD4800     73            CALL     SAVE              ; SAVE REGISTERS
003E    3A4020     74            LD       A, (DISP)         ; BLINK LITES
0041    2F         75            CPL
0042    324020     76            LD       (DISP), A
0045    D3E0       77            OUT      (LITE), A
0047    C9         78            RET
                   79
                   80  ;         SAVE REGISTER ROUTINE
                   81
                   82  SAVE:
0048    E3         83            EX       (SP), HL
0049    D5         84            PUSH     DE
004A    C5         85            PUSH     BC
004B    F5         86            PUSH     AF
004C    CD5600     87            CALL     GO
004F    F1         88            POP      AF
0050    C1         89            POP      BC
0051    D1         90            POP      DE
0052    E1         91            POP      HL
0053    FB         92            EI
0054    ED4D       93            RETI
                   94
                   95  GO:
0056    E9         96            JP       (HL)
                   97  *E
                   98
```

```
                                   TEST.CTC1
  LOC    OBJ CODE M STMT SOURCE STATEMENT

                       99  ;;       DATA AREA
                      100
 2000                 101           ORG     RAM
 2000                 102           DEFS    64            ;STACK AREA
                      103  STAK:    EQU     $
 2040                 104  DISP:    DEFS    1             ;LITE DISPLAY BYTE
                      105
                      106           END
```



a) Main Program



b) Interrupt Service Routine

Figure 6. Software for CTC Single-Cycle Use

```
                           TEST. CTC3
LOC    OBJ CODE M STMT SOURCE STATEMENT

                      1  ;        CTC TEST PROGRAM
                      2
                      3  ;        THIS PROGRAM INITIALIZES CTC INTERRUPT VECTOR,
                      4  ;        THEN STARTS CTC 3, THEN WAITS FOR CTC 3 TO
                      5  ;        TERMINATE. AFTER TERMINATING, THE CTC INTERRUPT
                      6  ;        THE CPU AND ENTERS A SERVICE ROUTINE THAT SETS
                      7  ;        A PROGRAM FLAG TO INDICATE ZERO COUNT, AND
                      8  ;        RESETS CTC 3.
                      9
                     10  ;        EQUATES
                     11
                     12  RAM:     EQU     2000H            ;RAM START ADDRESS
                     13  RAMSIZ:  EQU     1000H            ;RAM SIZE
                     14  CTC0:    EQU     12               ;CTC 0 PORT
                     15  CTC1:    EQU     CTC0+1           ;CTC 1 PORT
                     16  CTC2:    EQU     CTC0+2           ;CTC 2 PORT
                     17  CTC3:    EQU     CTC0+3           ;CTC 3 PORT
                     18  COUNT:   EQU     20               ;COUNT 20 PULSES
                     19
                     20  ;        CTC PARAMETERS
                     21
                     22  CCW:     EQU     1                ;CTRL BYTE
                     23           INTEN:  EQU     80H      ;INTERR. ENABLE
                     24           CTRMODE:        EQU      40H     ;COUNTER MODE
                     25           P256:   EQU     20H      ;PRESCALE BY 256
                     26           RISEDG: EQU     10H      ;START ON RISING EDGE
                     27           PSTRT:  EQU     8        ;PULSE STARTS TIMING
                     28           TCLOAD: EQU     4        ;TIME CONST. FOLLOWS
                     29           RESET:  EQU     2        ;SOFTWARE RESET
                     30  *E
                     31
0000                 32           ORG     0
0000   C31800        33           JP      BEGIN            ;GO MAIN PROGRAM
                     34
0010                 35           ORG     $.AND.0FFF0H.OR.10H
                     36  INTVEC:
                     37  CTCVEC:
0010   4100          38           DEFW    ICTC0
0012   4100          39           DEFW    ICTC1
0014   4100          40           DEFW    ICTC2
0016   4400          41           DEFW    ICTC3
                     42
                     43  ;;       MAIN PROGRAM
                     44
                     45  BEGIN:
0018   318120        46           LD      SP,STAK          ;INIT SP
001B   3E00          47           LD      A,INTVEC/256     ;INIT VECTOR REG.
001D   ED47          48           LD      I,A
001F   ED5E          49           IM      2                ;VECTORED INTERRUPT MC
0021   3E10          50           LD      A,CTCVEC.AND.11111000B
0023   D30C          51           OUT     (CTC0),A         ;SETUP CTC VECTOR
0025   3E01          52           LD      A,1              ;SET FLAG BYTE
0027   320020        53           LD      (FLAG),A
002A   FB            54           EI
                     55
                     56  LOOP:
002B   3A0020        57           LD      A,(FLAG)         ;READ FLAG BYTE
002E   CB47          58           BIT     0,A
0030   28F9          59           JR      Z,LOOP           ;BRANCH IF NOT SET
0032   CB87          60           RES     0,A              ;CLEAR FLAG BYTE
0034   320020        61           LD      (FLAG),A
0037   3ED5          62           LD      A,INTEN+CTRMODE+RISEDG+TCLOAD+1
0039   D30F          63           OUT     (CTC3),A         ;LOAD CTC 3
003B   3E14          64           LD      A,COUNT
003D   D30F          65           OUT     (CTC3),A
003F   18EA          66           JR      LOOP
                     67  *E
                     68
                     69  ;        INTERRUPT SERVICE ROUTINES FOR CTC
                     70
                     71  ICTC0:
                     72  ICTC1:
```

```
                                  TEST.CTC3
  LOC    OBJ CODE M STMT SOURCE STATEMENT
                      73  ICTC2:
 0041    FB           74          EI                          ;DUMMY INTERRUPT ROUTI
 0042    ED4D         75          RETI
                      76
                      77  ICTC3:
 0044    08           78          EX      AF,AF'
 0045    3E03         79          LD      A,00000011B         ;RESET CTC 3
 0047    D30F         80          OUT     (CTC3),A
 0049    3A0020       81          LD      A,(FLAG)            ;SET PROGRAM FLAG
 004C    CBC7         82          SET     0,A
 004E    320020       83          LD      (FLAG),A
 0051    08           84          EX      AF,AF'
 0052    FB           85          EI
 0053    ED4D         86          RETI
                      87  *E
                      88
                      89  ;;      DATA AREA
                      90
 2000                 91          ORG     RAM
 2000                 92  FLAG:   DEFS    1                   ;PROGRAM FLAG BYTE
 2001                 93          DEFS    128
                      94  STAK:   EQU     $
                      95
                      96          END
```

CONCLUSION The versatility of the Z80 CTC makes it useful in a myriad of applications. System efficiency and throughput can be improved through prudent use of the CTC with the Z80 CPU. Coupled with the powerful, vectored interrupt capabilities of the Z80 CPU, the CTC can be used to supply counter/timer functions to the CPU. This reduces software overhead on the CPU and significantly increases system throughput.

A Z80-Based System Using the DMA With the SIO

Zilog

Application Brief

January 1981

INTRODUCTION

In certain applications, serial data communications can be handled more efficiently by using a DMA device in conjunction with a serial controller. This application brief describes the use of the Z80A SIO and Z80A DMA hardware and software in a Z80-based system to transfer data to the SIO via the DMA.

Transfers through a serial data medium are usually done with a serial controller device, often a Universal Synchronous/Asynchronous Receiver/ Transmitter (USART), such as the Z80 SIO. Additionally, some sort of controlling device is required to manipulate the data on a character-by-character basis, (usually a CPU). Transferring characters can be accomplished either by polling the USART, which forces the CPU to take time away from other activities, or by initiating an interrupt mechanism, which requires CPU time only if there is data to be moved. However, when large blocks of data need to be moved, even the interrupt mechanism becomes awkward. In these cases, a Direct Memory Access (DMA) device is especially valuable.

With DMA transfer, data is moved directly between memory and I/O (or additional memory) without CPU intervention. Once initiated by the CPU, DMA operation continues transparently to CPU operation until completed. Then the DMA device can either interrupt the CPU or restart its cycle using the previously programmed parameters.

HARDWARE DESCRIPTION

The hardware used in the example for this brief consists of a Z80A CPU, a Z80A DMA controller, a Z80A SIO/2, some RAM and ROM, and some support circuitry (Figure 1).

The Z80A DMA contains a 16-bit address bus, an 8-bit data bus, and 13 control lines for external interfacing. The Z80 DMA can generate independent addresses for Port A and Port B. Each address can be variable or fixed. Variable addresses can be programmed to either increment or decrement from the programmed starting addresses, whereas fixed addressing eliminates the need for separate enabling lines for I/O ports.

Readable registers contain the current address of each port and a count of the number of bytes searched and/or transferred. Additional registers allow the DMA to perform bit-maskable data comparisons on the data that is being searched and/or transferred. The DMA has 21 writeable control registers and seven readable status registers, which together provide a high degree of programmability.

The DMA function described is for a simple test operation using memory-to-I/O transfer with no search options. The DMA is initialized to transfer data from a pattern in memory to the SIO when the SIO requests a byte via the $\overline{\text{WAIT}}/\overline{\text{RDY}}$ signal line. The SIO then sends the byte to a terminal, which displays it for visual inspection. After a block of bytes has been sent, the DMA restarts itself (Auto Restart mode) and the process repeats continuously. Since the data pattern in memory consists of displayable ASCII characters, data is easily verified by observing the characters displayed on the terminal.

One feature of the Z80 DMA is the ease with which it interfaces with the Z80 CPU. The DMA is designed to connect directly to the CPU, as illustrated in Figure 2. The 16 address lines, eight data lines, and seven control lines are connected directly to the corresponding lines on the Z80 CPU. These signals are then buffered by the 74LS241s and distributed to the rest of the system. The data bus is buffered by the 74LS245 bidirectional octal buffer. Other connections to the DMA include clock, $\overline{\text{CE}}/\overline{\text{WAIT}}$, $\overline{\text{INT}}$, RDY and IEI.

The clock input to the DMA is sensitive to both level and rise and fall times. The voltage should be no greater than +0.45V for a low level and no less than V_{CC}-0.6V for a

This application note refers to products as Z80 "A", "B" etc. to specifiy the speed grade. We are no longer using those characters for the speeds. For more details, please refer to the ordering information section.

high level. Additionally, the rise and fall times for the waveform should be no greater than 30ns, according to the device specifications. A clock driver device is used to deliver the proper voltage levels and rise/fall times.

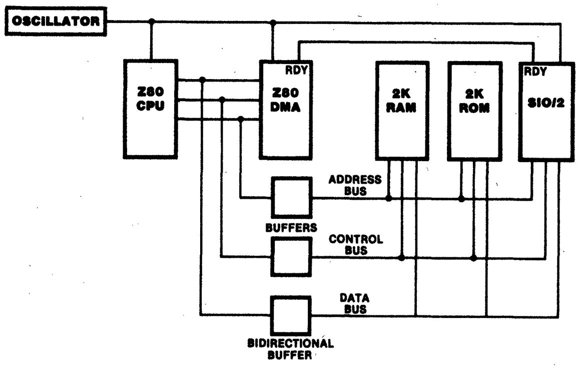


Figure 1. Block Diagram of a Z80 System with DMA and SIO.

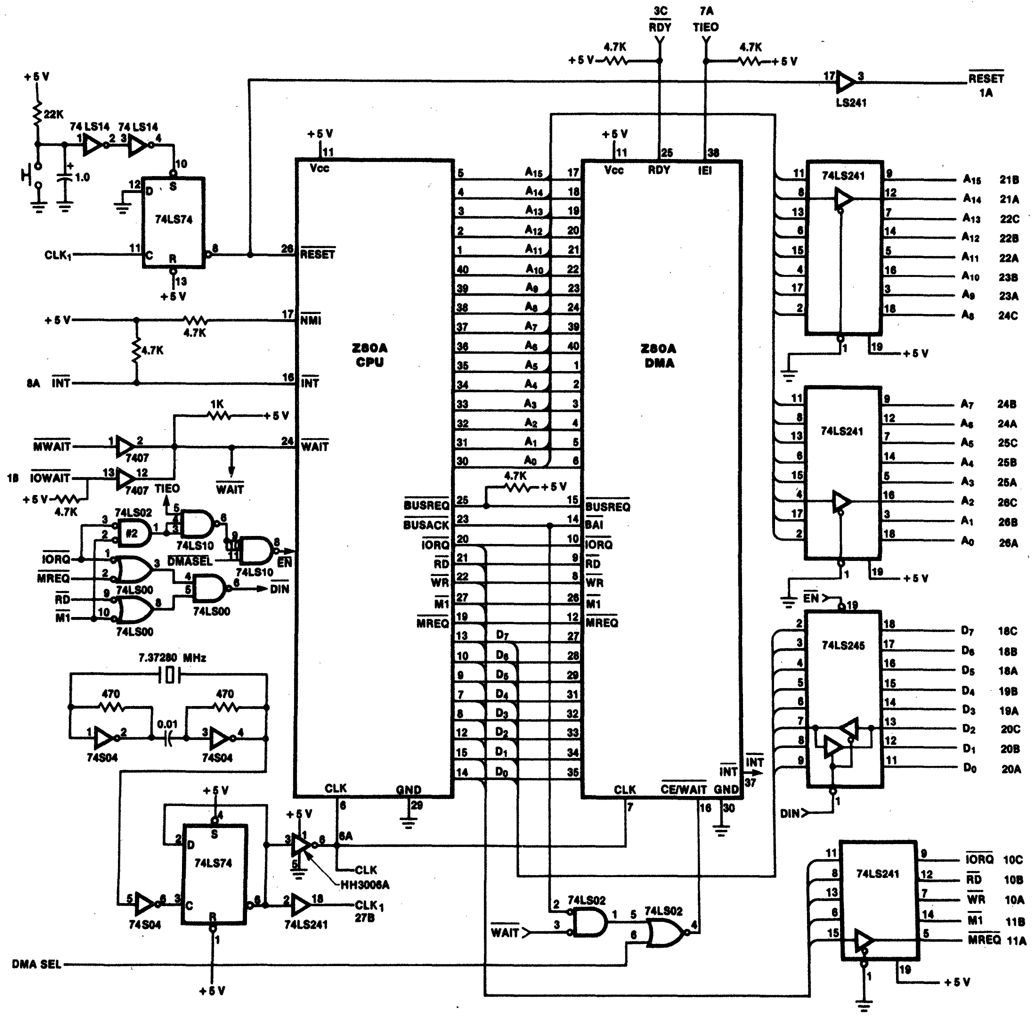
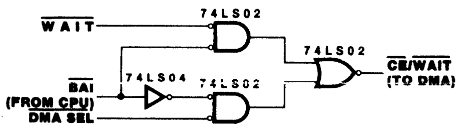


Figure 2. Schematic of CPU and DMA Interface

The $\overline{CE}/\overline{WAIT}$ input to the DMA serves a dual purpose. When the DMA is idle [Bus Acknowledge Input (BAI) inactive], the $\overline{CE}/\overline{WAIT}$ input is used to select the DMA during a CPU access cycle, allowing the DMA to be treated as a peripheral device by the CPU. However, when the DMA takes control of the system bus, the $\overline{CE}/\overline{WAIT}$ input can be programmed as a $\overline{WAIT}$ control line for the DMA, similar to the $\overline{WAIT}$ input on the Z80 CPU. Figure 3 shows the gating that determines the $\overline{CE}/\overline{WAIT}$ function.



NOTES:
$\overline{CE}/\overline{WAIT} = (\overline{DMA\ SEL} \bullet BAI) + (\overline{WAIT} \bullet \overline{BAI})$
Bus Acknowledge Input ($\overline{BAI}$) is active Low during the DMA cycle.

Figure 3. $\overline{CE}/\overline{WAIT}$ Control Logic.

With the SIO, the hardware interface is slightly more complex than the DMA hardware interface. The interface to the Z80 CPU is fairly straightforward, since the SIO is accessed as an I/O peripheral device. Still, the clock input has the same requirements as the DMA; so in order to provide this signal, some sort of clock driver is needed. In addition, if the SIO is used in an interrupt environment where its internally generated vector is placed onto the data bus, the data bus buffers must allow the interrupt vector to be presented to the CPU during the interrupt acknowledge cycle. Since the data bus is buffered at the CPU, this is not a problem with the example given here; the bus is controlled by the CPU circuitry. However, in larger systems, any buffers near the SIO need to be considered.

In addition, the system must supply some form of bit rate clock to the SIO for data communications. This is accomplished either by using an external clock source or by generating the clock with a device such as the CTC or CIO. Here the clock is supplied at a 1X rate for asynchronous communications from an external device such as a modem.

The $\overline{WAIT}/\overline{RDY}$ pin on the SIO is connected to the RDY input on the DMA. This provides character transfer control between the SIO and DMA. In this application, the ready function is used and the $\overline{WAIT}/\overline{RDY}$ pin is wired directly to the RDY input on the DMA with a pullup resistor. A low level initiates a DMA character transfer from memory to the SIO. The SIO drives the $\overline{WAIT}/\overline{RDY}$ line High or Low so that pullup is not strictly required. However, upon reset, the SIO $\overline{WAIT}/\overline{RDY}$ pin floats until the ready function is programmed in the SIO. Figure 4 shows the Z80 CPU-SIO interface.

Since the SIO has only one $\overline{WAIT}/\overline{RDY}$ pin per channel, it can be used with the DMA only during transmit or receive but not both simultaneously. Therefore, characters received by the SIO are transferred via interrupts with the CPU intervening. The interrupt system also handles errors detected either during reception or when the SIO notices an external or status change.

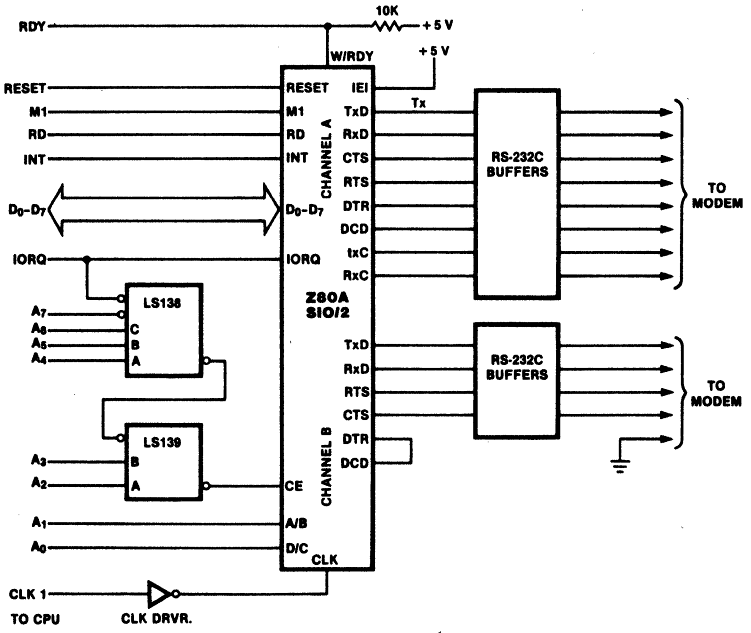


Figure 4. Z80 SIO Interface

PROGRAMMING

Before any action can occur, initialization must be performed on the Z80 CPU, the DMA, and the SIO devices. Since interrupts are used in processing special SIO conditions, the Z80 CPU must be initialized for the proper interrupt mode. In the example, the CPU is set to Interrupt Mode 2 using the IM instruction. The upper eight bits of the interrupt vector are loaded into the I register via the A register in the CPU. The Stack Pointer (SP) register must be loaded by the program upon reset, because it has an undefined value. The SP register is used when processing interrupts and when the Call instruction is executed during initialization. The appendix contains a source listing for a DMA test program using the SIO.

The DMA is initialized for memory-to-I/O, byte-at-a-time transfer with the search option disabled and operates continuously until stopped by a command from the CPU. The program uses Port A of the DMA for the memory source address (SRC) and Port B for the destination address (DST) and utilizes the auto restart option on the DMA so that data can be sent to the terminal as a stream of characters. Since Port B is a fixed destination address, it must be declared as the source when the DMA is given the Load command (WR6, CFH), as stated in the programming section of the DMA Technical Manual (document number 00-2013-A). Table 1 shows the initialization sequence for the example described here.

The SIO initialization sequence is straightforward. The example uses channel A in Asynchronous Communication mode with the DMA providing data characters to the SIO on a transmit buffer empty condition. The terminal requires async format, two stop bits, and even parity. An external 1X clock is used with the SIO for the bit rate clock. The lower eight bits of the SIO interrupt vector are loaded into WR2 through channel B, and the Status Affects Vector (SAV) bit in WR1 is also set. SAV provides eight separate interrupt vectors (four for each channel), allowing easy program operation. Table 2 shows the programming sequence and mode of the SIO for DMA operation. Note that when DMA transfers are used to move data, the transmit buffer empty interrupt should not be enabled (WR1, bit 1=0).

A data test pattern is generated in the memory buffer area used for transmission to the SIO so that intelligible information can be sent to the terminal for easy verification. This is done by a short routine that fills the memory block with an incremental pattern of ASCII characters in the range of from 20H to 7FH and appends a carriage return and a linefeed to the data block. Figure 5 contains a listing of the routine involved. The block length programmed into the DMA is one less than the actual block length transferred due to the counter characteristics of the Z80 DMA.

Table 1. DMA Initialization Sequence

1. Disable DMA
2. Issue six reset commands (insures a reset if DMA in undefined state)
3. WR0 - Port A (source) characteristics
4. Port A start address - low byte
5. Port A start address - high byte
6. Port A block length - low byte
7. Port A block length - high byte
8. WR1 - Port A increment address
9. WR2 - Port B is fixed address, I/O
10. WR4 - Byte mode, Port B address (low byte) follows
11. Port B (destination) address
12. WR5 - Auto Restart mode, $\overline{\text{CE}}/\overline{\text{WAIT}}$ is multiplexed
13. Insure Port A is standard timing
14. Insure Port B is standard timing
15. Load Port B
16. WR0 - Port A is source, Port B is destination
17. Load Port A

Table 2. SIO Initialization Sequence

Channel A

1. Channel Reset
2. WR1 - $\overline{\text{WAIT}}/\overline{\text{RDY}}$ enable for TX, ready function, RX interrupt on all characters; parity affects vector
3. WR4 - X1 clock, two stop bits, even parity
4. WR5 - DTR, RTS active, TX seven bits, enable TX
5. WR3 - RX seven bits

Channel B

1. Channel Reset
2. WR1 - status affects vector
3. WR2 - lower eight bits of vector

Once the CPU, DMA, and SIO are set up, the program enables the DMA device (WR6, 87H) and the data transfer process begins. The SIO brings the $\overline{\text{WAIT}}/\overline{\text{RDY}}$ output active as soon as the SIO has been initialized so that characters can be transmitted immediately. The user must insure that the DMA and data block have been set up properly before any data transfer actually occurs. DMA data transfer is different from the interrupt data transfer of the SIO, because with interrupts the SIO does not request data until it is activated by having a character sent to it.

Once operation of the DMA and SIO has begun, data transfers occur without CPU intervention unless the SIO encounters an error condition. An error causes the SIO to interrupt the CPU, thereby intervening in CPU processing. In this event, the CPU is interrupted by the device detecting the error and the DMA processing is terminated by the CPU. This termination is achieved by writing a command word to the DMA. The DMA remains disabled until given a command that enables it.

```
        LD      HL, SRC         ;%HL = start address
        LD      BC, BLKSIZ-2    ;%BC = length
        LD      D, 20H          ;%D  = data byte
LOOP:
        LD      (HL), D         ;store character
        INC     D               ;increment character code
        LD      A,D             ;mask upper bit
        AND     7FH
        OR      20H             ;keep displayable character
        LD      D,A             ;save in %D
        INC     HL              ;Bump memory ptr.
        DEC     BC              ;Bump byte count
        LD      A,B             ;see if through
        OR      C

        JR      NZ, LOOP        ;no-loop
        LD      (HL), 13        ;CR
        INC     HL
        LD      (HL), 10        ;LF
```

Figure 5. Data Test Pattern Generator Routine Listing.

CONCLUSION

This example shows only one aspect of using the DMA with the SIO. Use of the DMA with the SIO during receive deserves special consideration. Since the DMA operates without CPU processing, data received by the SIO does not normally indicate when the end of a message occurs. One solution to this problem is to send fixed-length data blocks so that the CPU can be interrupted when the DMA reaches terminal count. This is done by programming a fixed-length block count into the DMA and enabling it to interrupt the CPU upon End-Of-Block (EOB). As an alternative to the terminal count interrupt, the SIO can be programmed to interrupt the CPU when the closing flag is detected in SDLC mode. This allows the CPU to detect the end of a message using the SIO instead of the DMA.

Another method of detecting the end of a message is to dedicate a special EOB character used to terminate all message blocks. The DMA can then be programmed to search for this character during data transfers and to interrupt the CPU when the character is detected. This method allows for variable-length message blocks, up to the maximum byte count the DMA will accommodate. The disadvantage with this method is that the user must dedicate one character as the special EOB character.

The unique features of the DMA and SIO combine to form a powerful and flexible data communication mechanism. Due to the designed-in compatibility of the SIO and DMA, interfacing with both in hardware and software becomes a simplified task. Programming is easy because very little CPU intervention is necessary after initialization. Thus, the user is afforded a powerful tool for implementing an efficient, cost-effective data processing system.

APPENDIX

Following is a printout of the DMA/SIO test program. This program uses the DMA to transfer data from a pattern in memory to the SIO, which then sends the data, in async format at 9600 baud, to a terminal for display. The process continues until it is externally interrupted, such as by a reset.

Interrupts are used to process error conditions or to receive characters. However, no code is shown that handles the characters once they are received. Error conditions are reset by the interrupt service routine, although nothing is shown for these conditions either. The user normally sets a condition flag after resetting the error condition, so that the driver program can determine the appropriate course of action.

```
;          DMA/SIO TEST PROGRAM

;          BY M. PITCHER - 10/10/80

;          GENERATES BLOCK OF DATA IN RAM,
;          THEN OUTPUTS TO SIO VIA DMA,
;          THEN CONTINUES FOREVER.

RAM:       EQU       2000H               ;RAM START ADDR
RAMSIZ:    EQU       1000H               ;RAM SIZE
SIODA:     EQU       0                   ;SIO CH. A DATA PORT
SIOCA:     EQU       SIODA+1             ;SIO CH. A CTRL PORT
SIODB:     EQU       SIODA+2             ;SIO CH. B DATA PORT
SIOCB:     EQU       SIODB+1             ;SIO CH. B CTRL PORT
DMA:       EQU       OFOH                ;DMA PORT ADDR
DST:       EQU       SIODA               ;DESTINATION ADDR
BLKSIZ:    EQU       64                  ;XFER BLK SIZE
DMABLK:    EQU       BLKSIZ-1            ;DMA BLOCK SIZE VALUE


;          START DMA AFTER INITIALIZATION (WR6, 87H)
;          DMA PARAMETERS

DMAWRO:    EQU       0
           XFER:     EQU       1
           SRCH:     EQU       2
           XFRSCH:   EQU       3
           A_B:      EQU       4
           ALSTA:    EQU       8
           AHSTA:    EQU       10H
           ALBLEN:   EQU       20H
           AHBLEN:   EQU       40H

DMAWR1:    EQU       4
           AIO:      EQU       8
           AINCR:    EQU       10H
           ADECR:    EQU       0
           AFIXED:   EQU       20H
           AVTIM:    EQU       40H

DMAWR2:    EQU       0
           BIO:      EQU       8
           BINCR:    EQU       10H
           BDECR:    EQU       0
           BFIXED:   EQU       20H
           BVTIM:    EQU       40H

DMAWR3:    EQU       80H
           DMAEN:    EQU       40H
           INTEN:    EQU       20H
           MCHBYT:   EQU       10H
           MSKBYT:   EQU       8
           SOMCH:    EQU       4

DMAWR4:    EQU       81H
           BYTE:     EQU       0
           CONT:     EQU       20H
           BURST:    EQU       40H
           ICB:      EQU       10H
                     INTRDY:   EQU       40H
                     DMASAV:   EQU       20H
                     IV:       EQU       10H
                     PCB:      EQU       8
                     PULSE:    EQU       4
                     INTEOB:   EQU       2
                     INTMCH:   EQU       1

           BHSTA:    EQU       8
           BLSTA:    EQU       4

DMAWR5:    EQU       82H
```

```
                     72          AUTORS: EQU     20H
                     73          CEWAIT: EQU     10H
                     74          RDYHI:  EQU     8
                     75
                     76  ;       SETUP FOR ASYNC FORMAT AS FOLLOWS:
                     77  ;               9600 BAUD
                     78  ;               2 STOP BITS
                     79  ;               7 BIT CHARACTERS
                     80  ;               EVEN PARITY
                     81
                     82  ;       PROGRAM ASSUMES DMA XFER OF TX DATA
                     83  ;       THERE IS NO RECV DATA XFER
                     84  ;       STATUS IS REFLECTED IN "SIOFLG" LOC.
                     85  ;       EXTERNAL TX AND RX CLOCK ASSUMED
                     86
                     87  ;       SIOFLG -   X X 1 1 X X 1 1
                     88  ;                    /   !     !   `
                     89  ;               ERROR ASLEEP ERROR ASLEEP
                     90  ;                 CHANNEL B    CHANNEL A
                     91
                     92  SIOWR0: EQU     0
                     93          CHRES:  EQU     18H
                     94          ESCRES: EQU     10H
                     95          TBERES: EQU     28H
                     96          SRCRES: EQU     30H
                     97          RCRCRE: EQU     40H
                     98          TCRCRE: EQU     80H
                     99          EOMRES: EQU     0C0H
                    100
                    101  SIOWR1: EQU     1
                    102          WREN:   EQU     80H
                    103          RDY:    EQU     40H
                    104          WRONR:  EQU     20H
                    105          RXIFC:  EQU     8
                    106          RXIAP:  EQU     10H
                    107          RXIA:   EQU     18H
                    108          SIOSAV: EQU     4           ;CH. B ONLY
                    109          TXI:    EQU     2
                    110          EXTI:   EQU     1
                    111
                    112  SIOWR2: EQU     2                   ;CH. B ONLY
                    113
                    114  SIOWR3: EQU     3
                    115          RX8:    EQU     0C0H
                    116          RX6:    EQU     80H
                    117          RX7:    EQU     40H
                    118          RX5:    EQU     0
                    119          AUTOEN: EQU     20H
                    120          HUNT:   EQU     10H
                    121          RXCRC:  EQU     8
                    122          ADSRCH: EQU     4
                    123          SYNINH: EQU     2
                    124          RXEN:   EQU     1
                    125
                    126  SIOWR4: EQU     4
                    127          X64:    EQU     0C0H
                    128          X32:    EQU     80H
                    129          X16:    EQU     40H
                    130          X1:     EQU     0
                    131          EXTSYN: EQU     30H
                    132          SDLC:   EQU     20H
                    133          SYN16:  EQU     10H
                    134          SYN8:   EQU     0
                    135          STOP2:  EQU     0CH
                    136          STOP15: EQU     8
                    137          STOP1:  EQU     4
                    138          SYNCEN: EQU     0
                    139          EVEN:   EQU     2
                    140          PARITY: EQU     1
                    141
                    142  SIOWR5: EQU     5
```

```
LOC     OBJ CODE M STMT SOURCE STATEMENT                              ASM 5.9

                     143          DTR:     EQU     80H
                     144          TX8:     EQU     60H
                     145          TX6:     EQU     40H
                     146          TX7:     EQU     20H
                     147          TX5:     EQU     0
                     148          BREAK:   EQU     10H
                     149          TXEN:    EQU     8
                     150          CRC16:   EQU     4
                     151          RTS:     EQU     2
                     152          TXCRC:   EQU     1
                     153
                     154  SIOWR6: EQU      6
                     155
                     156  SIOWR7: EQU      7
                     157  *EJ
                     158
                     159  ;;      *** MAIN PROGRAM ***
                     160
0000                 161          ORG      0
0000    C32000       162          JP       BEGIN
                     163
0010                 164          ORG      $.AND.OFFFOH.OR.10H
                     165  INTVEC:
                     166  SIOVEC:
0010    6400         167          DEFW     CHBTBE
0012    7600         168          DEFW     CHBESC
0014    7000         169          DEFW     CHBRCA
0016    8A00         170          DEFW     CHBSRC
0018    9E00         171          DEFW     CHATBE
001A    B000         172          DEFW     CHAESC
001C    AA00         173          DEFW     CHARCA
001E    C400         174          DEFW     CHASRC
                     175
                     176  BEGIN:
0020    318120       177          LD       SP,STAK        ;INIT SP
0023    ED5E         178          IM       2              ;INTERRUPT MODE 2
0025    3E00         179          LD       A,INTVEC/256
0027    ED47         180          LD       I,A
0029    CD4D00       181          CALL     INIT           ;INIT DMA, SIO
002C    210120       182          LD       HL,SRC         ;GENERATE DATA PATTERN
002F    013E00       183          LD       BC,BLKSIZ-2
0032    1620         184          LD       D,20H
                     185  LOOP:
0034    72           186          LD       (HL),D
0035    14           187          INC      D
0036    7A           188          LD       A,D
0037    E67F         189          AND      7FH
0039    F620         190          OR       20H
003B    57           191          LD       D,A
003C    23           192          INC      HL
003D    0B           193          DEC      BC
003E    78           194          LD       A,B
003F    B1           195          OR       C
0040    20F2         196          JR       NZ,LOOP
0042    360D         197          LD       (HL),13          ;CR
0044    23           198          INC      HL
0045    360A         199          LD       (HL),10          ;LF
0047    3E87         200          LD       A,87H            ;ENABLE DMA
0049    D3F0         201          OUT      (DMA),A
                     202
004B    18FE         203          JR       $                ;LOOP FOREVER
                     204
                     205  INIT:
                     206  DMAINI:
004D    0EF0         207          LD       C,DMA            ;INIT DMA
004F    21EF00       208          LD       HL,DMATAB
0052    0616         209          LD       B,DMAEND-DMATAB
0054    EDB3         210          OTIR
                     211  SIOINI:
0056    210501       212          LD       HL,SIOTA         ;INIT SIO CH.A
0059    0E01         213          LD       C,SIOCA
005B    060A         214          LD       B,SIOEA-SIOTA
```

```
005D    EDB3           215             OTIR
005F    AF             216             XOR     A               ;CLEAR SIOFLG
0060    320020         217             LD      (SIOFLG),A
0063    C9             218             RET

                       219    *EJ
                       220
                       221    ;        INTERRUPT SERVICE ROUTINES
                       222
                       223    CHBTBE:
0064    CDD800         224             CALL    SAVE            ;CH. B TX BUFFER EMPTY
0067    3E00           225             LD      A,SIOWRO
0069    D303           226             OUT     (SIOCB),A
006B    3E28           227             LD      A,TBERES
006D    D303           228             OUT     (SIOCB),A
006F    C9             229             RET
                       230
                       231    CHBRCA:
0070    CDD800         232             CALL    SAVE            ;CH. B RX CHAR AVAIL.
0073    DB02           233             IN      A,(SIODB)
0075    C9             234             RET
                       235
                       236    CHBESC:
0076    CDD800         237             CALL    SAVE            ;EXTERNAL/STATUS CHG
0079    3E00           238             LD      A,SIOWRO
007B    D303           239             OUT     (SIOCB),A
007D    3E10           240             LD      A,ESCRES
007F    D303           241             OUT     (SIOCB),A
0081    3A0020         242             LD      A,(SIOFLG)
0084    CBE7           243             SET     4,A
0086    320020         244             LD      (SIOFLG),A
0089    C9             245             RET
                       246
                       247    CHBSRC:
008A    CDD800         248             CALL    SAVE            ;CH. B SPECIAL RX COND.
008D    3E00           249             LD      A,SIOWRO
008F    D303           250             OUT     (SIOCB),A
0091    3E30           251             LD      A,SRCRES
0093    D303           252             OUT     (SIOCB),A
0095    3A0020         253             LD      A,(SIOFLG)
0098    CBEF           254             SET     5,A
009A    320020         255             LD      (SIOFLG),A
009D    C9             256             RET
                       257
                       258    CHATBE:
009E    CDD800         259             CALL    SAVE            ;CH. A TX BUFFER EMPTY
00A1    3E00           260             LD      A,SIOWRO
00A3    D301           261             OUT     (SIOCA),A
00A5    3E28           262             LD      A,TBERES
00A7    D301           263             OUT     (SIOCA),A
00A9    C9             264             RET
                       265
                       266    CHARCA:
00AA    CDD800         267             CALL    SAVE            ;CH. A RX CHAR AVAIL.
00AD    DB00           268             IN      A,(SIODA)
00AF    C9             269             RET
                       270
                       271    CHAESC:
00B0    CDD800         272             CALL    SAVE            ;EXTERNAL/STATUS CHG
00B3    3E00           273             LD      A,SIOWRO
00B5    D301           274             OUT     (SIOCA),A
00B7    3E10           275             LD      A,ESCRES
00B9    D301           276             OUT     (SIOCA),A
00BB    3A0020         277             LD      A,(SIOFLG)
00BE    CBC7           278             SET     0,A
00C0    320020         279             LD      (SIOFLG),A
00C3    C9             280             RET
                       281
                       282    CHASRC:
00C4    CDD800         283             CALL    SAVE            ;CH. A SPECIAL RX COND.
00C7    3E00           284             LD      A,SIOWRO
00C9    D301           285             OUT     (SIOCA),A
```

```
00CB   3E30         286            LD      A,SRCRES
00CD   D301         287            OUT     (SIOCA),A
00CF   3A0020       288            LD      A,(SIOFLG)
00D2   CBCF         289            SET     1,A
00D4   320020       290            LD      (SIOFLG),A
00D7   C9           291            RET
                    292
                    293    ;       MATHEWS SAVE REGISTER ROUTINE
                    294
                    295    SAVE:
00D8   E3           296            EX      (SP),HL         ;SP =   HL
00D9   D5           297            PUSH    DE              ;       DE
00DA   C5           298            PUSH    BC              ;       BC
00DB   F5           299            PUSH    AF              ;       AF
00DC   DDE5         300            PUSH    IX              ;       IX
00DE   FDE5         301            PUSH    IY              ;       IY
00E0   CDEE00       302            CALL    GO              ;       SAVE PC
00E3   FDE1         303            POP     IY
00E5   DDE1         304            POP     IX
00E7   F1           305            POP     AF
00E8   C1           306            POP     BC
00E9   D1           307            POP     DE
00EA   E1           308            POP     HL
00EB   FB           309            EI
00EC   ED4D         310            RETI
                    311
                    312    GO:
00EE   E9           313            JP      (HL)

                    314    *EJ
                    315
                    316    ;;      CONSTANTS
                    317
                    318    DMATAB:
00EF   83           319            DEFB    83H             ;WR6, DISABLE DMA
00F0   C3           320            DEFB    0C3H            ;WR6, RESET
00F1   C3           321            DEFB    0C3H            ;WR6, RESET
00F2   C3           322            DEFB    0C3H            ;WR6, RESET
00F3   C3           323            DEFB    0C3H            ;WR6, RESET
00F4   C3           324            DEFB    0C3H            ;WR6, RESET
00F5   C3           325            DEFB    0C3H            ;WR6, RESET
00F6   79           326            DEFB    DMAWR0+XFER+ALSTA+AHSTA+ALBLEN+AHBLEN
00F7   01           327            DEFB    SRC.AND.255     ;PORT A ADDR (L)
00F8   20           328            DEFB    SRC/256         ;PORT A ADDR (H)
00F9   3F           329            DEFB    DMABLK.AND.255  ;PORT A COUNT (L)
00FA   00           330            DEFB    DMABLK/256      ;PORT A COUNT (H)
00FB   14           331            DEFB    DMAWR1+AINCR
00FC   28           332            DEFB    DMAWR2+BIO+BFIXED
00FD   85           333            DEFB    DMAWR4+BYTE+BLSTA
00FE   00           334            DEFB    DST.AND.255     ;PORT B ADDR (L)
00FF   B2           335            DEFB    DMAWR5+AUTORS+CEWAIT
0100   C7           336            DEFB    0C7H            ;WR6, RESET A TIMING
0101   CB           337            DEFB    0CBH            ;WR6, RESET B TIMING
0102   CF           338            DEFB    0CFH            ;WR6, LOAD PORT B
0103   05           339            DEFB    DMAWR0+XFER+A_B ;A -> B
0104   CF           340            DEFB    0CFH            ;WR6, LOAD COUNTERS
                    341    DMAEND: EQU     $
                    342
                    343    SIOTA:
0105   00           344            DEFB    SIOWR0          ;CH. RESET
0106   18           345            DEFB    CHRES
0107   01           346            DEFB    SIOWR1          ;RDY/WAIT, INT. MODE
0108   D0           347            DEFB    WREN+RDY+RXIAP
0109   04           348            DEFB    SIOWR4          ;MODE
010A   0F           349            DEFB    X1+STOP2+EVEN+PARITY
010B   05           350            DEFB    SIOWR5          ;TX PARAMS.
010C   AA           351            DEFB    DTR+TX7+TXEN+RTS
010D   03           352            DEFB    SIOWR3          ;RX PARAMS.
010E   40           353            DEFB    RX7
                    354    SIOEA:  EQU     $
                    355
                    356    SIOTB:
```

```
 LOC   OBJ CODE M STMT SOURCE STATEMENT                            ASM 5.9
010F   00            357          DEFB    SIOWRO          ;CH. RESET
0110   18            358          DEFB    CHRES
0111   01            359          DEFB    SIOWR1          ;STATUS AFFECTS VECTOR
0112   04            360          DEFB    SIOSAV
0113   02            361          DEFB    SIOWR2          ;VECTOR
0114   10            362          DEFB    SIOVEC.AND.255
                     363   SIOEB: EQU     $
                     364   *EJ
                     365
                     366   ;;     DATA AREA
                     367
2000                 368          ORG     RAM
2000                 369   SIOFLG: DEFS   1               ;SIO STATUS FLAG BYTE
2001                 370   SRC:   DEFS    BLKSIZ          ;DMA SOURCE ADDR
2041                 371          DEFS    64              ;STACK AREA
                     372   STAK:  EQU     $
                     373
                     374          END
```

Interfacing Z80 CPUs to the Z8500 Peripheral Family

Zilog

Application Note

May 1983

INTRODUCTION

The Z8500 Family consists of universal peripherals that can interface to a variety of microprocessor systems that use a non-multiplexed address and data bus. Though similar to Z80 peripherals, the Z8500 peripherals differ in the way they respond to I/O and Interrupt Acknowledge cycles. In addition, the advanced features of the Z8500 peripherals enhance system performance and reduce processor overhead.

To design an effective interface, the user needs an understanding of how the Z80 Family interrupt structure works, and how the Z8500 peripherals interact with this structure. This application note provides basic information on the interrupt structures, as well as a discussion of the hardware and software considerations involved in interfacing the Z8500 peripherals to the Z80 CPUs. Discussions center around each of the following situations:

- Z80A 4 MHz CPU to Z8500 4 MHz peripherals
- Z80B 6 MHz CPU to Z8500A 6 MHz peripherals
- Z80H 8 MHz CPU to Z8500 4 MHz peripherals
- Z80H 8 MHz CPU to Z8500A 6 MHz peripherals

This application note assumes the reader has a strong working knowledge of the Z8500 peripherals; it is not intended as a tutorial.

CPU HARDWARE INTERFACING

The hardware interface consists of three basic groups of signals: data bus, system control, and interrupt control, described below. For more detailed signal information, refer to Zilog's <u>Data Book</u>, Universal Peripherals.

Note: The timing specs. for the Z8530 have been improved. The numbers appearing in this application note are old timings.

Data Bus Signals

D_7-D_0	Data Bus (bidirectional, 3-state). This bus transfers data between the CPU and the peripherals.

System Control Signals

A_n-A_0	Address Select Lines (optional). These lines select the port and/or control registers.
$\overline{CE}$	Chip Enable (input, active Low). $\overline{CE}$ is used to select the proper peripheral for programming. $\overline{CE}$ should be gated with $\overline{IORQ}$ or $\overline{MREQ}$ to prevent spurious chip selects during other machine cycles.
$\overline{RD}$*	Read (input, active Low). $\overline{RD}$ activates the chip-read circuitry and gates data from the chip onto the data bus.
$\overline{WR}$*	Write (input, active Low). $\overline{WR}$ strobes data from the data bus into the peripheral.

*Chip reset occurs when $\overline{RD}$ and $\overline{WR}$ are active simultaneously.

Interrupt Control

$\overline{INTACK}$	Interrupt Acknowledge (input, active Low). This signal indicates an Interrupt Acknowledge cycle and is used with $\overline{RD}$ to gate the interrupt vector onto the data bus.
$\overline{INT}$	Interrupt Request (output, open-drain, active Low).

IEI Interrupt Enable In (input, active High).

IEO Interrupt Enable Out (output, active High).

These lines control the interrupt daisy chain for the peripheral interrupt response.

Z8500 I/O OPERATION

The Z8500 peripherals generate internal control signals from $\overline{RD}$ and $\overline{WR}$. Since PCLK has no required phase relationship to $\overline{RD}$ or $\overline{WR}$, the circuitry generating these signals provides time for metastable conditions to disappear.

The Z8500 peripherals are initialized for different operating modes by programming the internal registers. These internal registers are accessed during I/O Read and Write cycles, which are described below.

Read Cycle Timing

Figure 1 illustrates the Z8500 Read cycle timing. All register addresses and $\overline{INTACK}$ must remain stable throughout the cycle. If $\overline{CE}$ goes active after $\overline{RD}$ goes active, or if $\overline{CE}$ goes inactive before $\overline{RD}$ goes inactive, then the effective Read cycle is shortened.

Write Cycle Timing

Figure 2 illustrates the Z8500 Write cycle timing. All register addresses and $\overline{INTACK}$ must remain stable throughout the cycle. If $\overline{CE}$ goes active after $\overline{WR}$ goes active, or if $\overline{CE}$ goes inactive before $\overline{WR}$ goes inactive, then the effective Write cycle is shortened. Data must be available to the peripheral prior to the falling edge of $\overline{WR}$.

PERIPHERAL INTERRUPT OPERATION

Understanding peripheral interrupt operation requires a basic knowledge of the Interrupt Pending (IP) and Interrupt Under Service (IUS) bits in relation to the daisy chain. Both Z80 and Z8500 peripherals are designed in such a way that no additional interrupts can be requested during an Interrupt Acknowledge cycle. This allows the interrupt daisy chain to settle, and ensures proper response of the interrupting device.

The IP bit is set in the peripheral when CPU intervention is required (such conditions as buffer empty, character available, error detection, or status changes). The Interrupt Acknowledge cycle does not necessarily reset the IP bit. This bit is cleared by a software command to the peripheral, or when the action that generated the interrupt is completed (i.e., reading a character, writing data, resetting errors, or changing the status). When the interrupt has been serviced, other interrupts can occur.

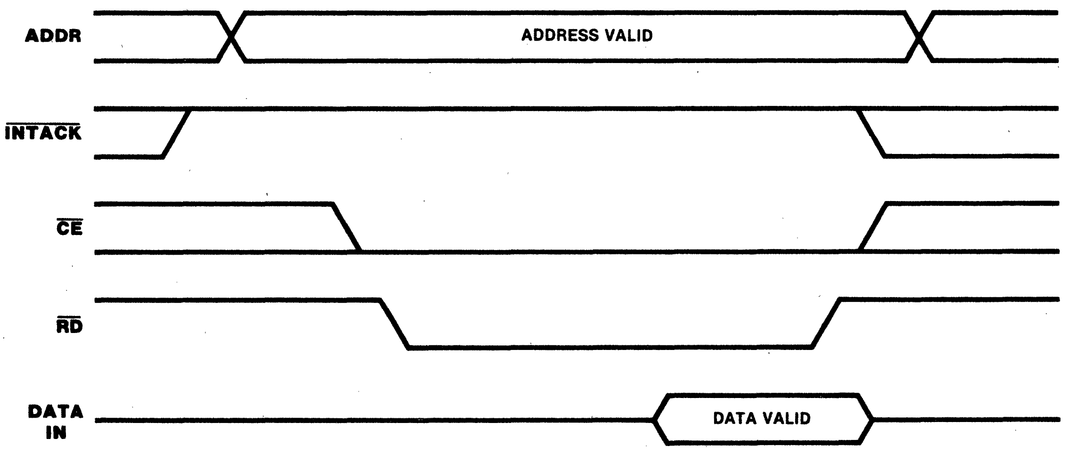


Figure 1. Z8500 Peripheral I/O Read Cycle Timing

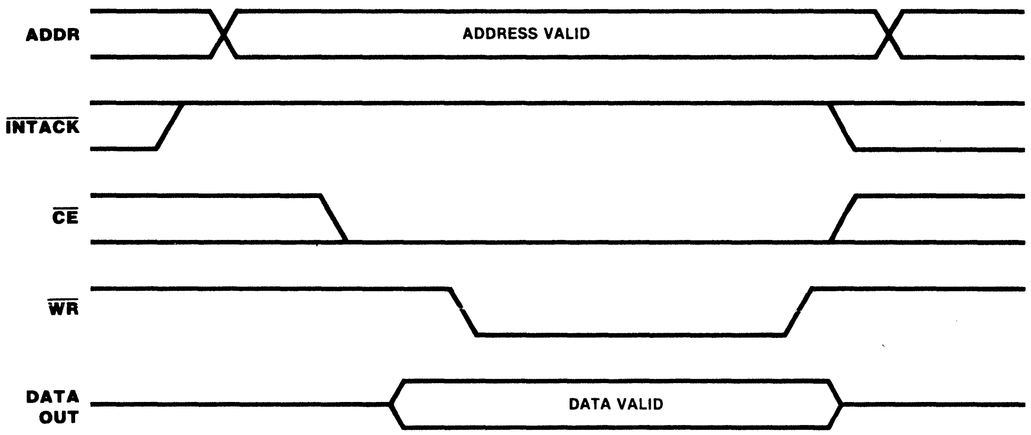


Figure 2. Z8500 Peripheral I/O Write Cycle Timing

The IUS bit indicates that an interrupt is currently being serviced by the CPU. The IUS bit is set during an Interrupt Acknowledge cycle if the IP bit is set and the IEI line is High. If the IEI line is Low, the IUS bit is not set, and the device is inhibited from placing its vector onto the data bus. In the Z80 peripherals, the IUS bit is normally cleared by decoding the RETI instruction, but can also be cleared by a software command (SIO). In the Z8500 peripherals, the IUS bit is cleared only by software commands.

Z80 Interrupt Daisy-Chain Operation

In the Z80 peripherals, both the IP and IUS bits control the IEO line and the lower portion of the daisy chain.

When a peripheral's IP bit is set, its IEO line is forced Low. This is true regardless of the state of the IEI line. Additionally, if the peripheral's IUS bit is clear and its IEI line High, the $\overline{\text{INT}}$ line is also forced Low.

The Z80 peripherals sample for both $\overline{\text{M1}}$ and $\overline{\text{IORQ}}$ active, and $\overline{\text{RD}}$ inactive to identify an Interrupt Acknowledge cycle. When $\overline{\text{M1}}$ goes active and $\overline{\text{RD}}$ is inactive, the peripheral detects an Interrupt Acknowledge cycle and allows its interrupt daisy chain to settle. When the $\overline{\text{IORQ}}$ line goes active with $\overline{\text{M1}}$ active, the highest priority interrupting peripheral places its interrupt vector onto the data bus. The IUS bit is also set to indicate that the peripheral is currently under service. As long as the IUS bit is set, the IEO line is forced Low. This inhibits any lower priority devices from requesting an interrupt.

When the Z80 CPU executes the RETI instruction, the peripherals monitor the data bus and the highest priority device under service resets its IUS bit.

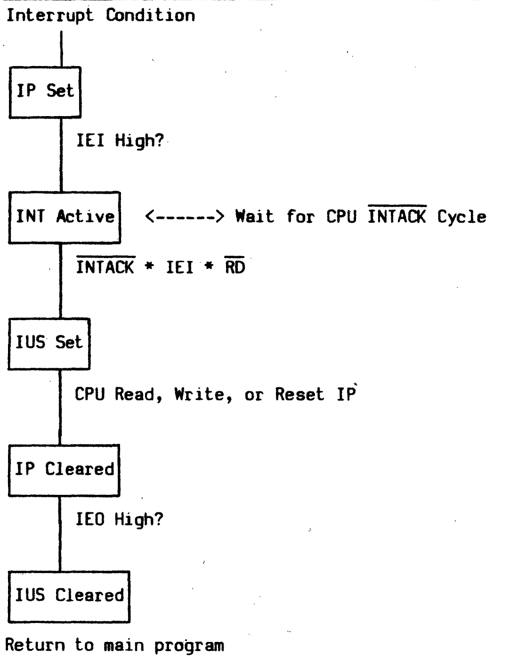
Z8500 Interrupt Daisy-Chain Operation

In the Z8500 peripherals, the IUS bit normally controls the state of the IEO line. The IP bit affects the daisy chain only during an Interrupt Acknowledge cycle. Since the IP bit is normally not part of the Z8500 peripheral interrupt daisy chain, there is no need to decode the RETI instruction. To allow for control over the daisy chain, Z8500 peripherals have a Disable Lower Chain (DLC) software command that pulls IEO Low. This can be used to selectively deactivate parts of the daisy chain regardless of the interrupt status. Table 1 shows the truth tables for the Z8500 interrupt daisy-chain control signals during certain cycles. Table 2 shows the interrupt state diagram for the Z8500 peripherals.

Table 1. Z8500 Daisy-Chain Control Signals

Truth Table for Daisy Chain Signals During Idle State				Truth Table for Daisy Chain Signals During $\overline{\text{INTACK}}$ Cycle			
IEI	IP	IUS	IEO	IEI	IP	IUS	IEO
0	X	X	0	0	X	X	0
1	X	0	1	1	1	X	0
1	X	1	0	1	X	1	0
				1	0	0	1

Table 2. Z8500 Interrupt State Diagram


Interrupt Condition
IP Set
IEI High?
INT Active
<------> Wait for CPU $\overline{\text{INTACK}}$ Cycle
$\overline{\text{INTACK}}$ * IEI * $\overline{\text{RD}}$
IUS Set
CPU Read, Write, or Reset IP
IP Cleared
IEO High?
IUS Cleared
Return to main program

The Z8500 peripherals use $\overline{\text{INTACK}}$ (Interrupt Acknowledge) for recognition of an Interrupt Acknowledge cycle. This pin, used in conjunction with $\overline{\text{RD}}$, allows the Z8500 peripheral to gate its interrupt vector onto the data bus. An active $\overline{\text{RD}}$ signal during an Interrupt Acknowledge cycle performs two functions. First, it allows the highest priority device requesting an interrupt to place its interrupt vector on the data bus. Secondly, it sets the IUS bit in the highest priority device to indicate that the device is currently under service.

INPUT/OUTPUT CYCLES

Although Z8500 peripherals are designed to be as universal as possible, certain timing parameters differ from the standard Z80 timing. The following sections discuss the I/O interface for each of the Z80 CPUs and the Z8500 peripherals. Figure 5 depicts logic for the Z80A CPU to Z8500 peripherals (and Z80B CPU to Z8500A peripherals) I/O interface as well as the Interrupt Acknowledge interface. Figures 4 and 7 depict some of the logic used to interface the Z80H CPU to the Z8500 and Z8500A peripherals for the I/O and Interrupt Acknowledge interfaces. The logic required for adding additional Wait states into the timing flow is not discussed in the folowing sections.

Z80A CPU to Z8500 Peripherals

No additional Wait states are necessary during the I/O cycles, although additional Wait states can be inserted to compensate for timing delays that are inherent in a system. Although the Z80A timing parameters indicate a negative value for data valid prior to $\overline{\text{WR}}$, this is a worse than "worst case" value. This parameter is based upon the longest (worst case) delay for data available from the falling edge of the CPU clock minus the shortest (best case) delay for CPU clock High to $\overline{\text{WR}}$ Low. The negative value resulting from these two parameters does not occur because the worst case of one parameter and the best case of the other do not occur within the same device. This indicates that the value for data available prior to $\overline{\text{WR}}$ will always be greater than zero.

All setup and pulse width times for the Z8500 peripherals are met by the standard Z80A timing. In determining the interface necessary, the $\overline{\text{CE}}$ signal to the Z8500 peripherals is assumed to be the decoded address qualified with the $\overline{\text{IORQ}}$ signal.

Figure 3a shows the minimum Z80A CPU to Z8500 peripheral interface timing for I/O cycles. If additional Wait states are needed, the same number of Wait states can be inserted for both I/O Read and Write cycles to simplify interface logic. There are several ways to place the Z80A CPU into a Wait condition (such as counters or shift registers to count system clock pulses), depending upon whether or not the user wants to place Wait states in all I/O cycles, or only during Z8500 I/O cycles. Tables 3 and 4 list the Z8500 peripheral and the Z80A CPU timing parameters (respectively) of concern during the I/O cycles. Tables 5 and 6 list the equations used in determining if these parameters are satisfied. In generating these equations and the values obtained from them, the required number of Wait states was taken into account. The reference numbers in Tables 3 and 4 refer to the timing diagram in Figure 3a.

Table 3. Z8500 Timing Parameters I/O Cycles

Worst Case			Min	Max	Units
6.	TsA(WR)	Address to $\overline{WR}$ Low Setup	80		ns
1.	TsA(RD)	Address to $\overline{RD}$ Low Setup	80		ns
2.	TdA(DR)	Address to Read Data Valid		590	ns
	TsCE1(WR)	$\overline{CE}$ Low to $\overline{WR}$ Low Setup	0		ns
	TsCE1(RD)	$\overline{CE}$ Low to $\overline{RD}$ Low Setup	0		ns
4.	TwRD1	$\overline{RD}$ Low Width	390		ns
8.	TwWR1	$\overline{WR}$ Low Width	390		ns
3.	TdRDf(DR)	$\overline{RD}$ Low to Read Data Valid		255	ns
7.	TsDW(WR)	Write Data to $\overline{WR}$ Low Setup	0		ns

Table 4. Z80A Timing Parameters I/O Cycles

Worst Case			Min	Max	Units
	TcC	Clock Cycle Period	250		ns
	TwCh	Clock Cycle High Width	110		ns
	TfC	Clock Cycle Fall Time		30	ns
	TdCr(A)	Clock High to Address Valid		110	ns
	TdCr(RDf)	Clock High to $\overline{RD}$ Low		85	ns
	TdCr(IORQf)	Clock High to $\overline{IORQ}$ Low		75	ns
	TdCr(WRf)	Clock High to $\overline{WR}$ Low		65	ns
5.	TsD(Cf)	Data to Clock Low Setup	50		ns

Table 5. Parameter Equations

Z8500 Parameter	Z80A Equation		Value	Units
TsA(RD)		TcC-TdCr(A)	140 min	ns
TdA(DR)		3TcC+TwCh-TdCr(A)-TsD(Cf)	800 min	ns
TdRDf(DR)		2TcC+TwCh-TsD(Cf)	460 min	ns
TwRD1		2TcC+TwCh+TfC-TdCr(RDf)	525 min	ns
TsA(WR)		TcC-TdCr(A)	140 min	ns
TsDW(WR)			> 0 min	ns
TwWR1		2TcC+TwCh+TfC-TdCr(WRf)	560 min	ns

Table 6. Parameter Equations

Z80A Parameter	Z8500 Equation		Value	Units
TsD(Cf)		Address		
		3TcC+TwCh-TdCr(A)-TdA(DR)	160 min	ns
		$\overline{RD}$		
		2TcC+TwCh-TdCr(RDf)-TdRD(DR)	135 min	ns

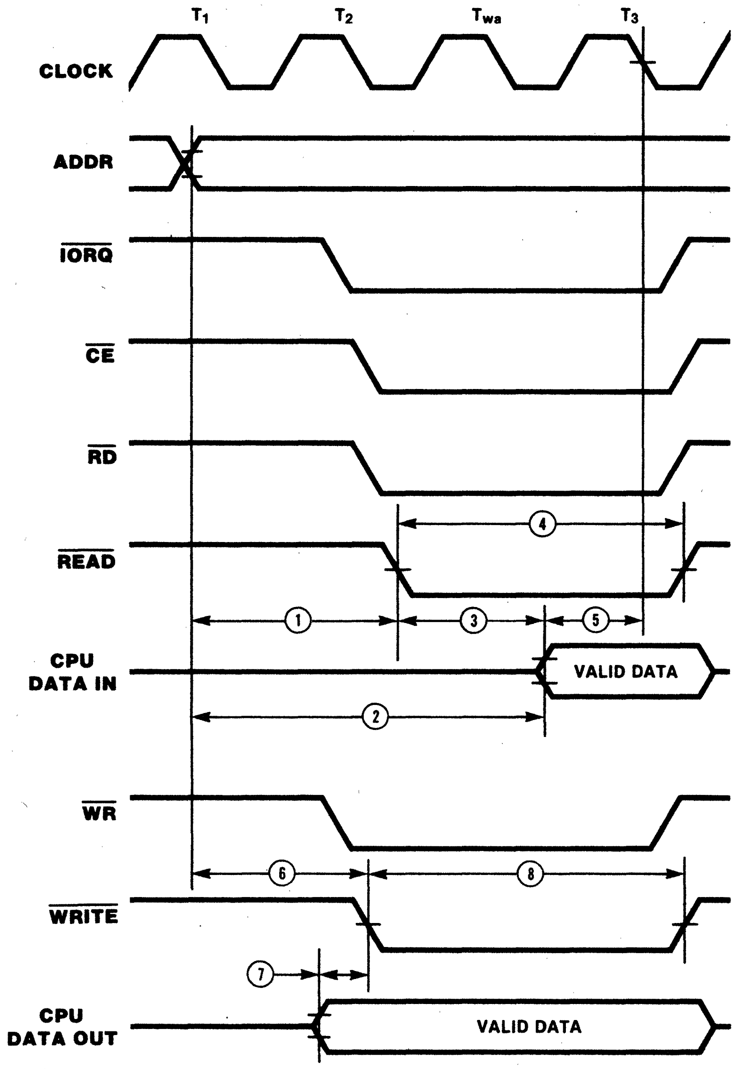


Figure 3a. Z80A CPU to Z8500 Peripheral Minimum I/O Cycle Timing

Z80B CPU to Z8500A Peripherals

No additional Wait states are necessary during I/O cycles, although Wait states can be inserted to compensate for any system delays. Although the Z80B timing parameters indicate a negative value for data valid prior to $\overline{WR}$, this is a worse than "worst case" value. This parameter is based upon the longest (worst case) delay for data available from the falling edge of the CPU clock minus the shortest (best case) delay for CPU clock High to $\overline{WR}$ Low. The negative value resulting from these two parameters does not occur because the worst case of one parameter and the best case of the other do not occur within the same device. This indicates that the value for data available prior to $\overline{WR}$ will always be greater than zero.

All setup and pulse width times for the Z8500A peripherals are met by the standard Z80B timing. In determining the interface necessary, the $\overline{CE}$ signal to the Z8500A peripherals is assumed to be the decoded address qualified with the $\overline{IORQ}$ signal.

Figure 3b shows the minimum Z80B CPU to Z8500A peripheral interface timing for I/O cycles. If additional Wait states are needed, the same number of Wait states can be inserted for both I/O Read and I/O Write cycles in order to simplify interface logic. There are several ways to place the Z80B CPU into a Wait condition (such as counters or shift registers to count system clock pulses), depending upon whether or not the user wants to place Wait states in all I/O cycles, or only during Z8500A I/O cycles. Tables 7 and 8 list the Z8500A peripheral and the Z80B CPU timing parameters (respectively) of concern during the I/O cycles. Tables 9 and 10 list the equations used in determining if these parameters are satisfied. In generating these equations and the values obtained from them, the required number of Wait states was taken into account. The reference numbers in Tables 7 and 8 refer to the timing diagram of Figure 3b.

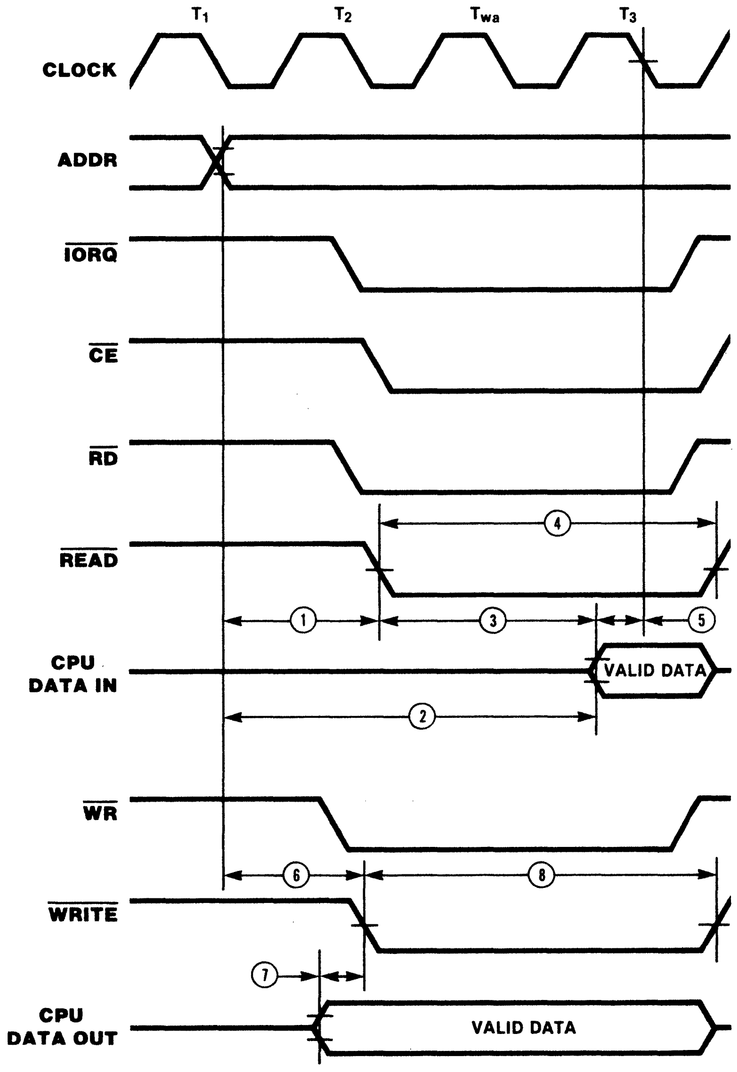


Figure 3b. Z80B CPU to Z8500A Peripheral Minimum I/O Cycle Timing

Table 7. Z8500A Timing Parameters I/O Cycles

	Worst Case		Min	Max	Units
6.	TsA(WR)	Address to $\overline{WR}$ Low Setup	80		ns
1.	TsA(RD)	Address to $\overline{RD}$ Low Setup	80		ns
2.	TdA(DR)	Address to Read Data Valid		420	ns
	TsCE1(WR)	$\overline{CE}$ Low to $\overline{WR}$ Low Setup	0		ns
	TsCE1(RD)	$\overline{CE}$ Low to $\overline{RD}$ Low Setup	0		ns
4.	TwRD1	$\overline{RD}$ Low Width	250		ns
8.	TwWR1	$\overline{WR}$ Low Width	250		ns
3.	TdRDf(DR)	$\overline{RD}$ Low to Read Data Valid		180	ns
7.	TsDW(WR)	Write Data to $\overline{WR}$ Low Setup	0		ns

Table 8. Z80B Timing Parameters I/O Cycles

	Worst Case		Min	Max	Units
	TcC	Clock Cycle Period	165		ns
	TwCh	Clock Cycle High Width	65		ns
	TfC	Clock Cycle Fall Time		20	ns
	TdCr(A)	Clock High to Address Valid		90	ns
	TdCr(RDf)	Clock High to $\overline{RD}$ Low		70	ns
	TdCr(IORQf)	Clock High to $\overline{IORQ}$ Low		65	ns
	TdCr(WRf)	Clock High to $\overline{WR}$ Low		60	ns
5.	TsD(Cf)	Data to Clock Low Setup	40		ns

Table 9. Parameter Equations

Z8500A Parameter	Z80B Equation	Value	Units
TsA(RD)	TcC-TdCr(A)	>75 min	ns
TdA(DR)	3TcC+TwCh-TdCr(A)-TsD(Cf)	430 min	ns
TdRDf(DR)	2TcC+TwCh-TsD(Cf)	345 min	ns
TwRD1	2TcC+TwCh+TfC-TdCr(RDf)	325 min	ns
TsA(WR)	TcC-TdCr(A)	75 min	ns
TsDW(WR)		> 0 min	ns
TwWR1	2TcC+TwCh+TfC-TdCr(WRf)	352 min	ns

Table 10. Parameter Equations

Z80B Parameter	Z8500A Equation	Value	Units
TsD(Cf)	Address		
	3TcC+TwCh-TdCr(A)-TdA(DR)	50 min	ns
	$\overline{RD}$		
	2TcC+TwCh-TdCr(RDf)-TdRD(DR)	75 min	ns

Z80H CPU to Z8500 Peripherals

During an I/O Read cycle, there are three Z8500 parameters that must be satisfied. Depending upon the loading characteristics of the $\overline{RD}$ signal, the designer may need to delay the leading (falling) edge of $\overline{RD}$ to satisfy the Z8500 timing parameter TsA(RD) (Address Valid to $\overline{RD}$ Setup). Since Z80H timing parameters indicate that the $\overline{RD}$ signal may go Low after the falling edge of T_2, it is recommended that the rising edge of the system clock be used to delay $\overline{RD}$ (if necessary). The CPU must also be placed into a Wait condition long enough to satisfy TdA(DR) (Address Valid to Read Data Valid Delay) and TdRDf(DR) ($\overline{RD}$ Low to Read Data Valid Delay).

During an I/O Write cycle, there are three other Z8500 parameters that must be satisfied. Depending upon the loading characteristics of the $\overline{WR}$ signal and the data bus, the designer may need to delay the leading (falling) edge of $\overline{WR}$ to satisfy the Z8500 timing parameters TsA(WR) (Address Valid to $\overline{WR}$ Setup) and TsDW(WR) (Data Valid Prior to $\overline{WR}$ setup). Since Z80H timing parameters indicate that the $\overline{WR}$ signal may go Low after the falling edge of T_2, it is recommended that the rising edge of the system clock be used to delay $\overline{WR}$ (if necessary). This delay will ensure that both parameters are satisfied. The CPU must also be placed into a Wait condition long enough to satisfy TwWRl ($\overline{WR}$ Low Pulse Width). Assuming that the $\overline{WR}$ signal is delayed, only two additional Wait states are needed during an I/O Write cycle when interfacing the Z80H CPU to the Z8500 peripherals.

To simplify the I/O interface, the designer can use the same number of Wait states for both I/O Read and I/O Write cycles. Figure 3c shows the minimum Z80H CPU to Z8500 peripheral interface timing for the I/O cycles (assuming that the same number of Wait states are used for both cycles and that both $\overline{RD}$ and $\overline{WR}$ need to be delayed). Figure 4 shows two circuits that can be used to delay the leading (falling) edge of either the $\overline{RD}$ or the $\overline{WR}$ signals. There are several ways to place the Z80A CPU into a Wait condition (such as counters or shift registers to count system clock pulses), depending upon whether or not the user wants to place Wait states in all I/O cycles, or only during Z8500 I/O cycles. Tables 4 and 11 list the Z8500 peripheral and the Z80H CPU timing parameters (respectively) of concern during the I/O cycles. Tables 14 and 15 list the equations used in determining if these parameters are satisfied. In generating these equations and the values obtained from them, the required number of Wait states was taken into account. The reference numbers in Tables 4 and 11 refer to the timing diagram of Figure 3c.

Table 11. Z80H Timing Parameter I/O Cycles

		Equation	Min	Max	Units
	TcC	Clock Cycle Period	125		ns
	TwCh	Clock Cycle High Width	55		ns
	TfC	Clock Cycle Fall Time		10	ns
	TdCr(A)	Clock High to Address Valid		80	ns
	TdCr(RDf)	Clock High to $\overline{RD}$ Low		60	ns
	TdCr(IORQf)	Clock High to $\overline{IORQ}$ Low		55	ns
	TdCr(WRf)	Clock High to $\overline{WR}$ Low		55	ns
5.	TsD(Cf)	Data to Clock Low Setup	30		ns

Table 12. Parameter Equations

Z8500 Parameter	Z80H Equation	Value	Units
TsA(RD)	2TcC-TdCr(A)	170 min	ns
TdA(DR)	6TcC+TwCh-TdCr(A)-TsD(Cf)	695 min	ns
TdRDf(DR)	4TcC+TwCh-TsD(Cf)	523 min	ns
TwRDl	4TcC+TwCh+TfC-TdCr(RDf)	503 min	ns
TsA(WR)	$\overline{WR}$ - delayed		
	2TcC-TdCr(A)	170 min	ns
TsDW(WR)		> 0 min	ns
TwWRl	4TcC+TwCh+TfC	563 min	ns

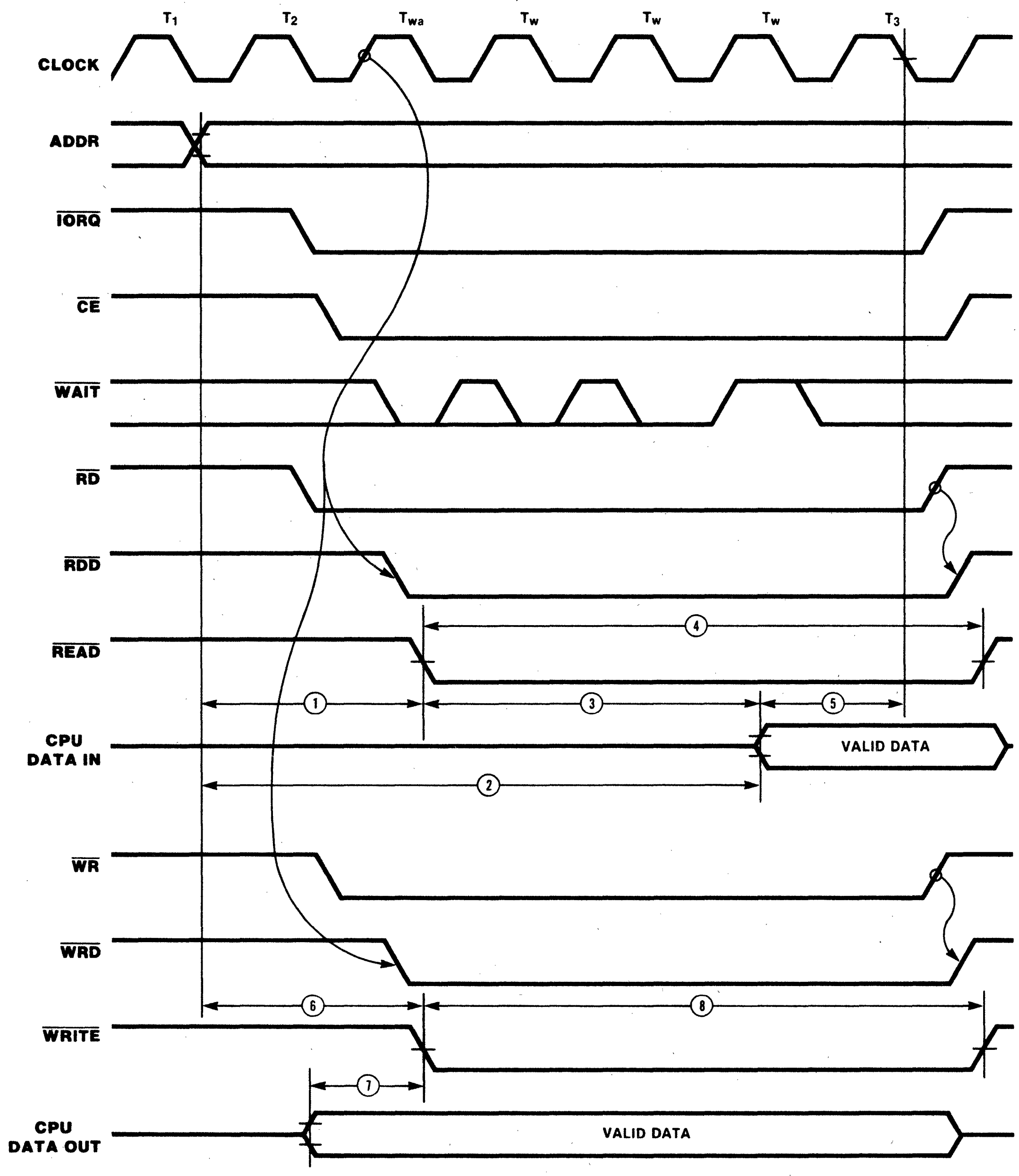


Figure 3c. Z80H CPU to Z8500 Peripheral Minimum I/O Cycle Timing

Z80H CPU to Z8500A Peripherals

During an I/O Read cycle, there are three Z8500A parameters that must be satisfied. Depending upon the loading characteristics of the $\overline{RD}$ signal, the designer may need to delay the leading (falling) edge of $\overline{RD}$ to satisfy the Z8500A timing parameter TsA(RD) (Address Valid to $\overline{RD}$ Setup). Since Z80H timing parameters indicate that the $\overline{RD}$ signal may go Low after the falling edge of T_2, it is recommended that the rising edge of the system clock be used to delay $\overline{RD}$ (if necessary). The CPU must also be placed into a Wait condition long enough to satisfy TdA(DR) (Address Valid to Read Data Valid Delay) and TdRDf(DR) ($\overline{RD}$ Low to Read Data Valid Delay). Assuming that the $\overline{RD}$ signal is delayed, then only one additional Wait state is needed during an I/O Read cycle when interfacing the Z80H CPU to the Z8500A peripherals.

During an I/O Write cycle, there are three other Z8500A parameters that have to be satisfied. Depending upon the loading characteristics of the $\overline{WR}$ signal and the data bus, the designer may need to delay the leading (falling) edge of $\overline{WR}$ to satisfy the Z8500A timing parameters TsA(WR) (Address Valid to $\overline{WR}$ Setup) and TsDW(WR) (Data Valid Prior to $\overline{WR}$ Setup). Since Z80H timing parameters indicate that the $\overline{WR}$ signal may go Low after the falling edge of T_2, it is recommended that the rising edge of the system clock be used to delay $\overline{WR}$ (if necessary). This delay will ensure that both parameters are satisfied. The CPU must also be placed into a Wait condition long enough to satisfy TwWRl ($\overline{WR}$ Low Pulse Width). Assuming that the $\overline{WR}$ signal is delayed, then only one additional Wait state is needed during an I/O Write cycle when interfacing the Z80H CPU to the Z8500A peripherals.

Figure 3d shows the minimum Z80H CPU to Z8500A peripheral interface timing for the I/O cycles (assuming that the same number of Wait states are used for both cycles and that both $\overline{RD}$ and $\overline{WR}$ need to be delayed). Figure 4 shows two circuits that may be used to delay the leading (falling) edge of either the $\overline{RD}$ or the $\overline{WR}$ signals. There are several methods used to place the Z80A CPU into a Wait condition (such as counters or shift registers to count system clock pulses), depending upon whether or not the user wants to place Wait states in all I/O cycles, or only during Z8500A I/O cycles. Tables 7 and 11 list the Z8500A peripheral and the Z80H CPU timing parameters (respectively) of concern during the I/O cycles. Tables 14 and 15 list the equations used in determining if these parameters are satisfied. In generating these equations and the values obtained from them, the required number of Wait states was taken into account. The reference numbers in Tables 4 and 11 refer to the timing diagram of Figure 3d.

Table 13. Parameter Equations

Z80H Parameter	Z8500 Equation	Value	Units
TsD(Cf)	Address		
	6TcC+TwCh-TdCr(A)-TdA(DR)	135 min	ns
	$\overline{RD}$ - delayed		
	4TcC+TwCh+TfC-TdRD(DR)	300 min	ns

Table 14. Parameter Equations

Z8500A Parameter	Z80H Equation	Value	Units
TsA(RD)	2TcC-TdCr(A)	170 min	ns
TdA(DR)	6TcC+TwCh-TdCr(A)-TsD(Cf)	695 min	ns
TdRDf(DR)	4TcC+TwCh-TsD(Cf)	525 min	ns
TwRDl	4TcC+TwCh+TfC-TdCr(RDf)	503 min	ns
TsA(WR)	$\overline{WR}$ - delayed		
	2TcC-TdCr(A)	170 min	ns
TsDW(WR)		> 0 min	ns
TwWRl	2TcC+TwCh+TfC	313 min	ns

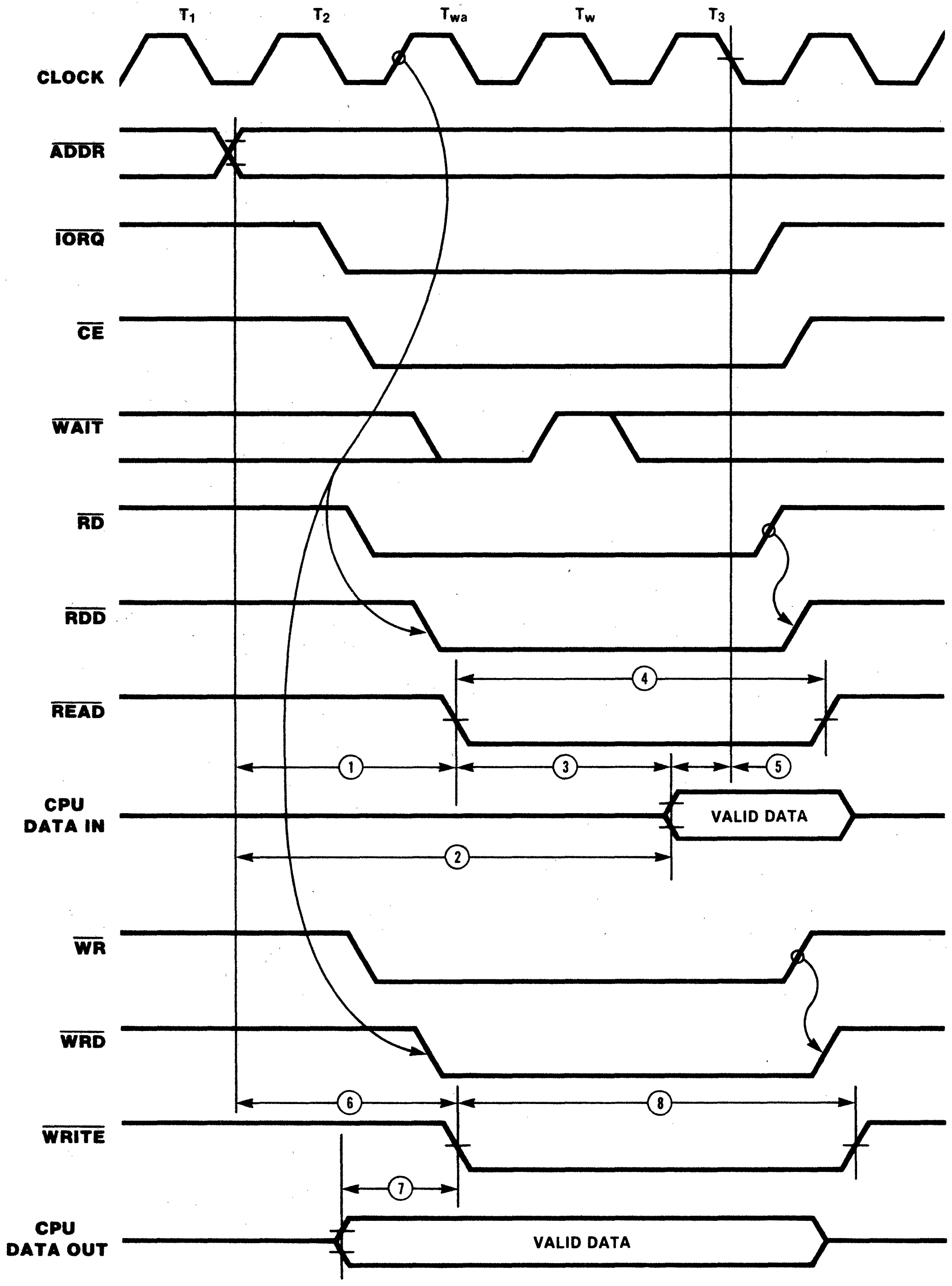
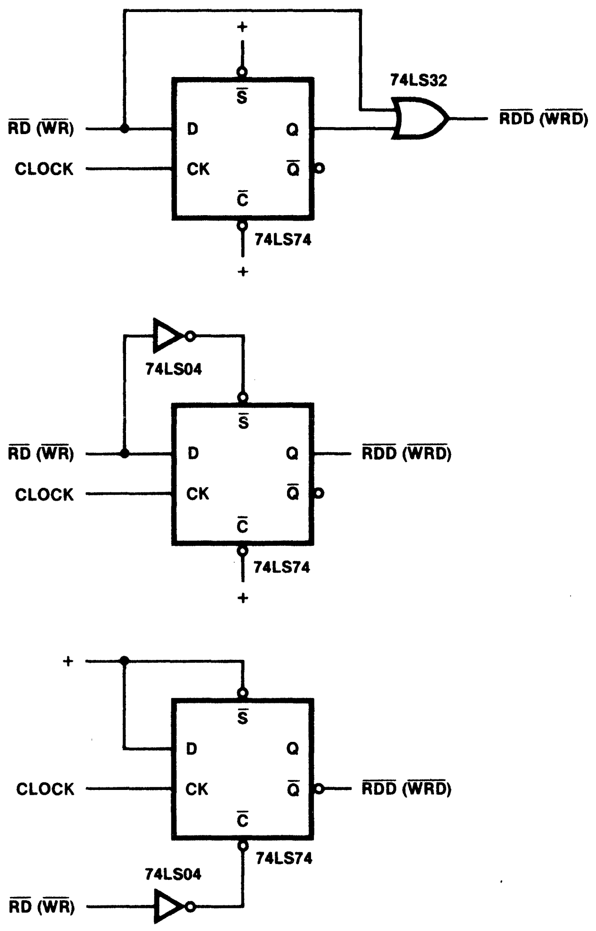


Figure 3d. Z80H CPU to Z8500A Peripheral Minimum I/O Cycle Timing



74LS32
$\overline{S}$
$\overline{RD}$ ($\overline{WR}$)
D
Q
$\overline{RDD}$ ($\overline{WRD}$)
CLOCK
CK
$\overline{Q}$
$\overline{C}$
74LS74
74LS04
$\overline{S}$
$\overline{RD}$ ($\overline{WR}$)
D
Q
$\overline{RDD}$ ($\overline{WRD}$)
CLOCK
CK
$\overline{Q}$
$\overline{C}$
74LS74
$\overline{S}$
D
Q
CLOCK
CK
$\overline{Q}$
$\overline{RDD}$ ($\overline{WRD}$)
$\overline{C}$
74LS74
74LS04
$\overline{RD}$ ($\overline{WR}$)

Figure 4. Delaying $\overline{RD}$ or $\overline{WR}$

Table 15. Parameter Equations

Z80H Parameter	Z8500A Equation	Value	Units
TsD(Cf)	Address		
	4TcC+TwCh-TdCr(A)-TdA(DR)	55 min	ns
	$\overline{RD}$ - delayed		
	2TcC+TwCh-TdRD(DR)	125 min	ns

INTERRUPT ACKNOWLEDGE CYCLES

The primary timing differences between the Z80 CPUs and Z8500 peripherals occur in the Interrupt Acknowledge cycle. The Z8500 timing parameters that are significant during Interrupt Acknowledge cycles are listed in Table 16, while the Z80 parameters are listed in Table 17. The reference numbers in Tables 16 and 17 refer to Figures 6, 8a, and 8b.

If the CPU and the peripherals are running at different speeds (as with the Z80H interface), the $\overline{\text{INTACK}}$ signal must be synchronized to the peripheral clock. Synchronization is discussed in detail under Interrupt Acknowledge for Z80H CPU to Z8500/8500A Peripherals.

During an Interrupt Acknowledge cycle, Z8500 peripherals require both $\overline{\text{INTACK}}$ and $\overline{\text{RD}}$ to be active at certain times. Since the Z80 CPUs do not issue either $\overline{\text{INTACK}}$ or $\overline{\text{RD}}$, external logic must generate these signals.

Generating these two signals is easily accomplished, but the Z80 CPU must be placed into a Wait condition until the peripheral interrupt vector is valid. If more peripherals are added to the daisy chain, additional Wait states may be necessary to give the daisy chain time to settle. Sufficient time between $\overline{\text{INTACK}}$ active and $\overline{\text{RD}}$ active should be allowed for the entire daisy chain to settle.

Since the Z8500 peripheral daisy chain does not use the IP flag except during interrupt acknowledge, there is no need for decoding the RETI instruction used by the Z80 peripherals. In each of the Z8500 peripherals, there are commands that reset the individual IUS flags.

EXTERNAL INTERFACE LOGIC

The following sections discuss external interface logic required during Interrupt Acknowledge cycles for each interface type.

CPU/Peripheral Same Speed

Figure 5 shows the logic used to interface the Z80A CPU to the Z8500 peripherals and the Z80B CPU to Z8500A peripherals during an Interrupt Acknowledge cycle. The primary component in this logic is the Shift register (74LS164), which generates $\overline{\text{INTACK}}$, $\overline{\text{READ}}$, and $\overline{\text{WAIT}}$.

Table 16. Z8500 Timing Parameters Interrupt Acknowledge Cycles

	Worst Case		**4 MHz Min**	**4 MHz Max**	**6 MHz Min**	**6 MHz Max**	**Units**
1.	TsIA(PC)	$\overline{\text{INTACK}}$ Low to PCLK High Setup	100		100		ns
	ThIA(PC)	$\overline{\text{INTACK}}$ Low to PCLK High Hold	100		100		ns
2.	TdIAi(RD)	$\overline{\text{INTACK}}$ Low to RD (Acknowledge) Low	350		250		ns
5.	TwRDA	$\overline{\text{RD}}$ (Acknowledge) Width	350		250		ns
3.	TdRDA(DR)	$\overline{\text{RD}}$ (Acknowledge) to Data Valid		250		180	ns
	TsIEI(RDA)	IEI to $\overline{\text{RD}}$ (Acknowledge) Setup	120		100		ns
	ThIEI(RDA)	IEI to $\overline{\text{RD}}$ (Acknowledge) Hold	100		70		ns
	TdIEI(IE)	IEI to IEO Delay		150		100	ns

Table 17. Z80 CPU Timing Parameters Interrupt Acknowledge Cycles

	Worst Case		**4 MHz Min**	**4 MHz Max**	**6 MHz Min**	**6 MHz Max**	**8 MHz Min**	**8 MHz Max**	**Units**
	TdC(M1f)	Clock High to $\overline{\text{M1}}$ Low Delay		100		80		70	ns
	TdM1f(IORQf)	$\overline{\text{M1}}$ Low to $\overline{\text{IORQ}}$ Low Delay	575*		345*		275*		ns
4.	TsD(Cr)	Data to Clock High Setup	35		30		25		ns

*Z80A: 2TcC + TwCh + TfC - 65
Z80B: 2TcC + TwCh + TfC - 50
Z80H: 2TcC + TwCh + TfC - 45

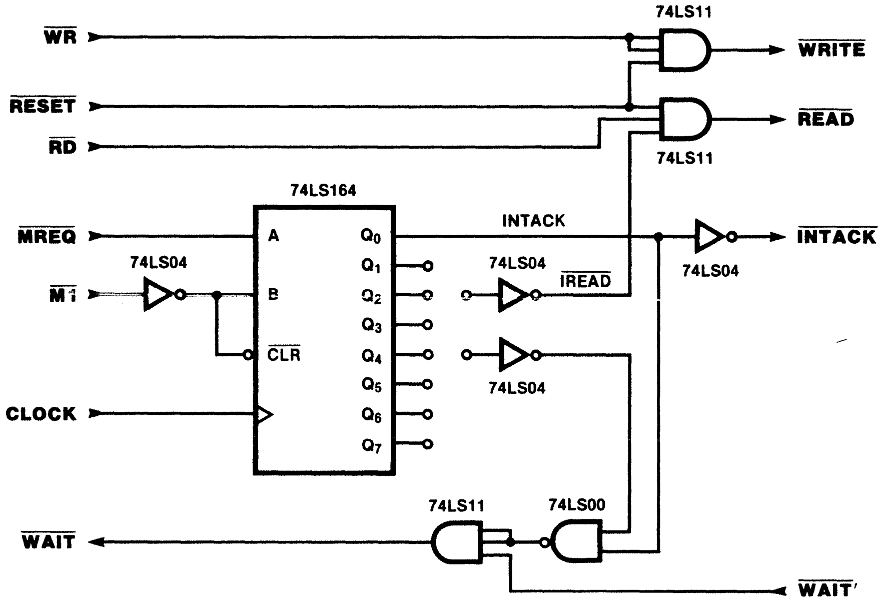


Figure 5. Z80A/Z80B CPU to Z8500/Z8500A Peripheral Interrupt Acknowledge Interface Logic

During I/O and normal memory access cycles, the Shift register remains cleared because the $\overline{M1}$ signal is inactive. During opcode fetch cycles, also, the Shift register remains cleared, because only 0s can be clocked through the register. Since Shift register outputs are Low, $\overline{READ}$, $\overline{WRITE}$, and $\overline{WAIT}$ are controlled by other system logic and gated through the AND gates (74LS11). During I/O and normal memory access cycles, $\overline{READ}$ and $\overline{WRITE}$ are active as a result of the system $\overline{RD}$ and $\overline{WR}$ signals (respectively) becoming active. If system logic requires that the CPU be placed into a Wait condition, the $\overline{WAIT}$' signal controls the CPU. Should it be necessary to reset the system, $\overline{RESET}$ causes the interface logic to generate both $\overline{READ}$ and $\overline{WRITE}$ (the Z8500 peripheral Reset condition).

Normally an Interrupt Acknowledge cycle is indicated by the Z80 CPU when $\overline{M1}$ and $\overline{IORQ}$ are both active (which can be detected on the third rising clock edge after T_1). To obtain an early indication of an Interrupt Acknowledge cycle, the Shift register decodes an active $\overline{M1}$ in the presence of an inactive $\overline{MREQ}$ on the rising edge of T_2.

During an Interrupt Acknowledge cycle, the $\overline{INTACK}$ signal is generated on the rising edge of T_2. Since it is the presence of $\overline{INTACK}$ and an active $\overline{READ}$ that gates the interrupt vector onto the data bus, the logic must also generate $\overline{READ}$ at the proper time. The timing parameter of concern here is TdIAi(RD) [$\overline{INTACK}$ to $\overline{RD}$ (Acknowledge) Low Delay]. This time delay allows the interrupt daisy chain to settle so that the device requesting the interrupt can place its interrupt vector onto the data bus. The Shift register allows a sufficient time delay from the generation of $\overline{INTACK}$ before it generates $\overline{READ}$. During this delay, it places the CPU into a Wait state until the valid interrupt vector can be placed onto the data bus. If the time between these two signals is insufficient for daisy chain settling, more time can be added by taking $\overline{READ}$ and $\overline{WAIT}$ from a later position on the Shift register.

Figure 6 illustrates Interrupt Acknowledge cycle timing resulting from the Z80A CPU to Z8500 peripheral and the Z80B CPU to Z8500A peripheral interface. This timing comes from the logic illustrated in Figure 5, which can be used for both interfaces. Should more Wait states be required, the additional time can be calculated in terms of system clocks, since the CPU clock and PCLK are the same.

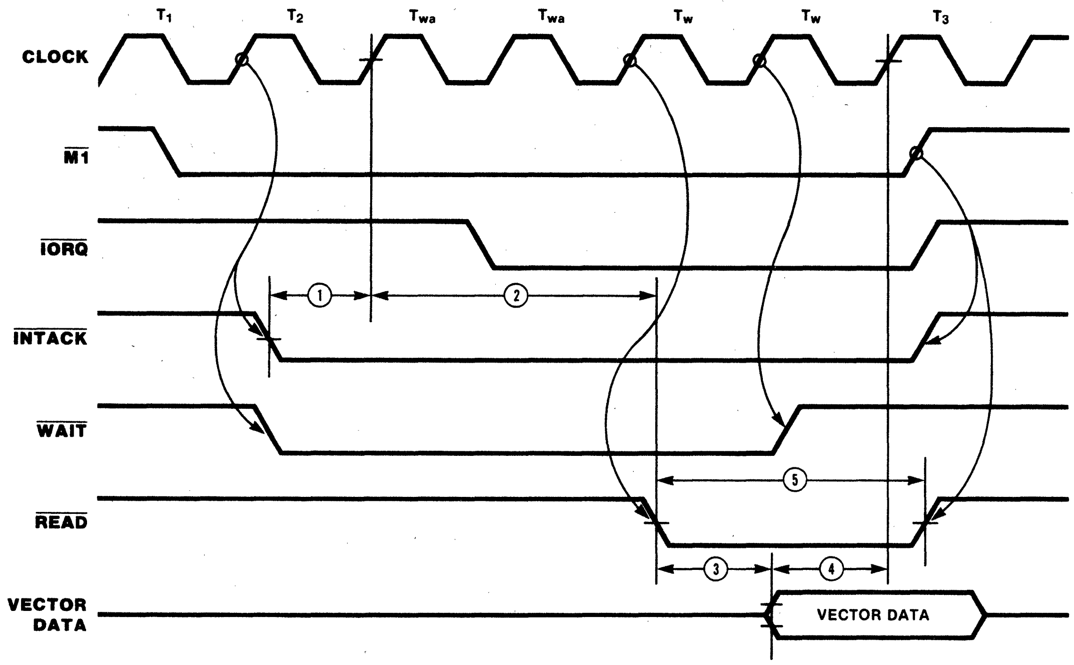


Figure 6. Z80A/Z80B CPU to Z8500/Z8500A Peripheral Interrupt Acknowledge Interface Timing

Z80H CPU to Z8500/Z8500A Peripherals

Figure 7 depicts logic that can be used in interfacing the Z80H CPU to the Z8500/Z8500A peripherals. This logic is the same as that shown in Figure 5, except that a synchronizing flip-flop is used to recognize an Interrupt Acknowledge cycle. Since Z8500 peripherals do not rely upon PCLK except during Interrupt Acknowledge cycles, synchronization need occur only at that time. Since the CPU and the peripherals are running at different speeds, $\overline{\text{INTACK}}$ and $\overline{\text{RD}}$ must be synchronized to the Z8500 peripherals clock.

During I/O and normal memory access cycles, the synchronizing flip-flop and the Shift register remain cleared because the $\overline{\text{M1}}$ signal is inactive. During opcode fetch cycles, the flip-flop and the Shift register again remain cleared, but this time because the $\overline{\text{MREQ}}$ signal is active. The synchronizing flip-flop allows an Interrupt Acknowledge cycle to be recognized on the rising edge of T_2 when $\overline{\text{M1}}$ is active and $\overline{\text{MREQ}}$ is inactive, generating the INTA signal. When INTA is active, the Shift register can clock and generate $\overline{\text{INTACK}}$ to the peripheral and $\overline{\text{WAIT}}$ to the CPU. The Shift register delays the generation of $\overline{\text{READ}}$ to the peripheral until the daisy chain settles. The $\overline{\text{WAIT}}$ signal is removed when sufficient time has been allowed for the interrupt vector data to be valid.

Figure 8a illustrates Interrupt Acknowledge cycle timing for the Z80H CPU to Z8500 peripheral interface. Figure 8b illustrates Interrupt Acknowledge cycle timing for the Z80H CPU to Z8500A peripheral interface. These timings result from the logic in Figure 7. Should more Wait states be required, the needed time should be calculated in terms of PCLKs, not CPU clocks.

Z80 CPU to Z80 and Z8500 Peripherals

In a Z80 system, a combination of Z80 peripherals and Z8500 peripherals can be used compatibly. While there is no restriction on the placement of the Z8500 peripherals in the daisy chain, it is recommended that they be placed early in the chain to minimize propagation delays during RETI cycles.

During an Interrupt Acknowledge cycle, the IEO line from the Z8500 peripherals changes to reflect the interrupt status. Time should be allowed for this change to ripple through the remainder of the daisy chain before activating $\overline{\text{IORQ}}$' to the Z80 peripherals, or $\overline{\text{READ}}$ to the Z8500 peripherals.

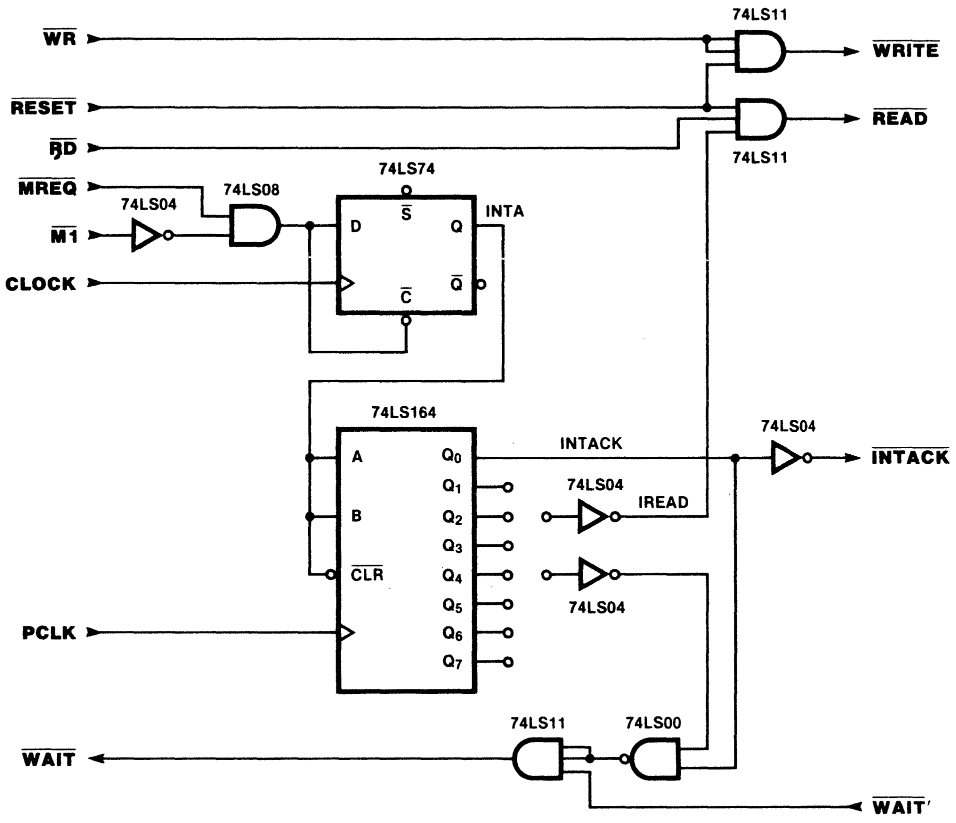


Figure 7. Z80H to Z8500/Z8500A Peripheral Interrupt Acknowledge Interface Logic

During the RETI cycles, the IEO line from the Z8500 peripherals does not change state as in the Z80 peripherals. As long as the peripherals are at the top of the daisy chain, propagation delays are minimized.

The logic necessary to create the control signals for both Z80 and Z8500 peripherals is shown in Figure 9. This logic delays the generation of $\overline{\text{IORQ}}$' to the Z80 peripherals by the same amount of time necessary to generate $\overline{\text{READ}}$ for the Z8500 peripherals. Timing for this logic during an Interrupt Acknowledge cycle is depicted in Figure 10.

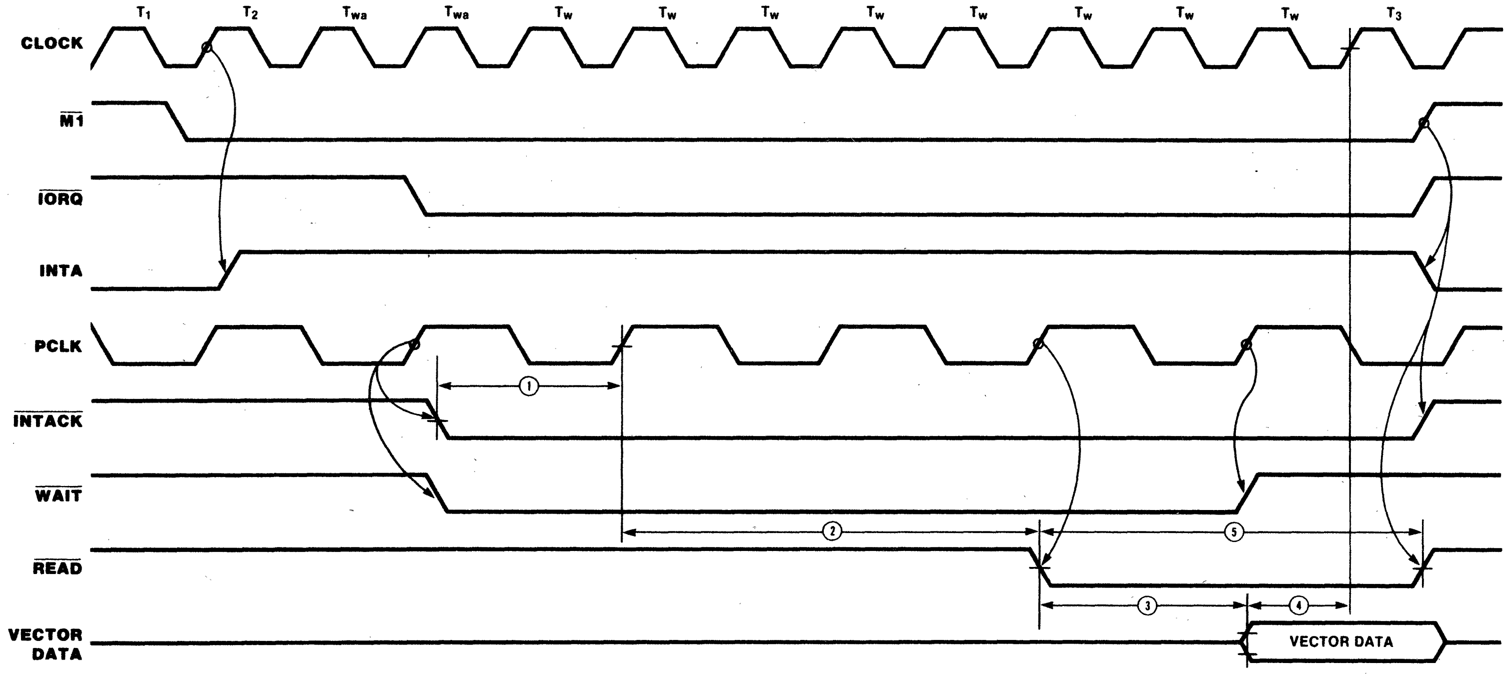


Figure 8a. Z80H CPU to Z8500 Peripheral Interrupt Acknowledge Interface Timing

2296-011

2296-012

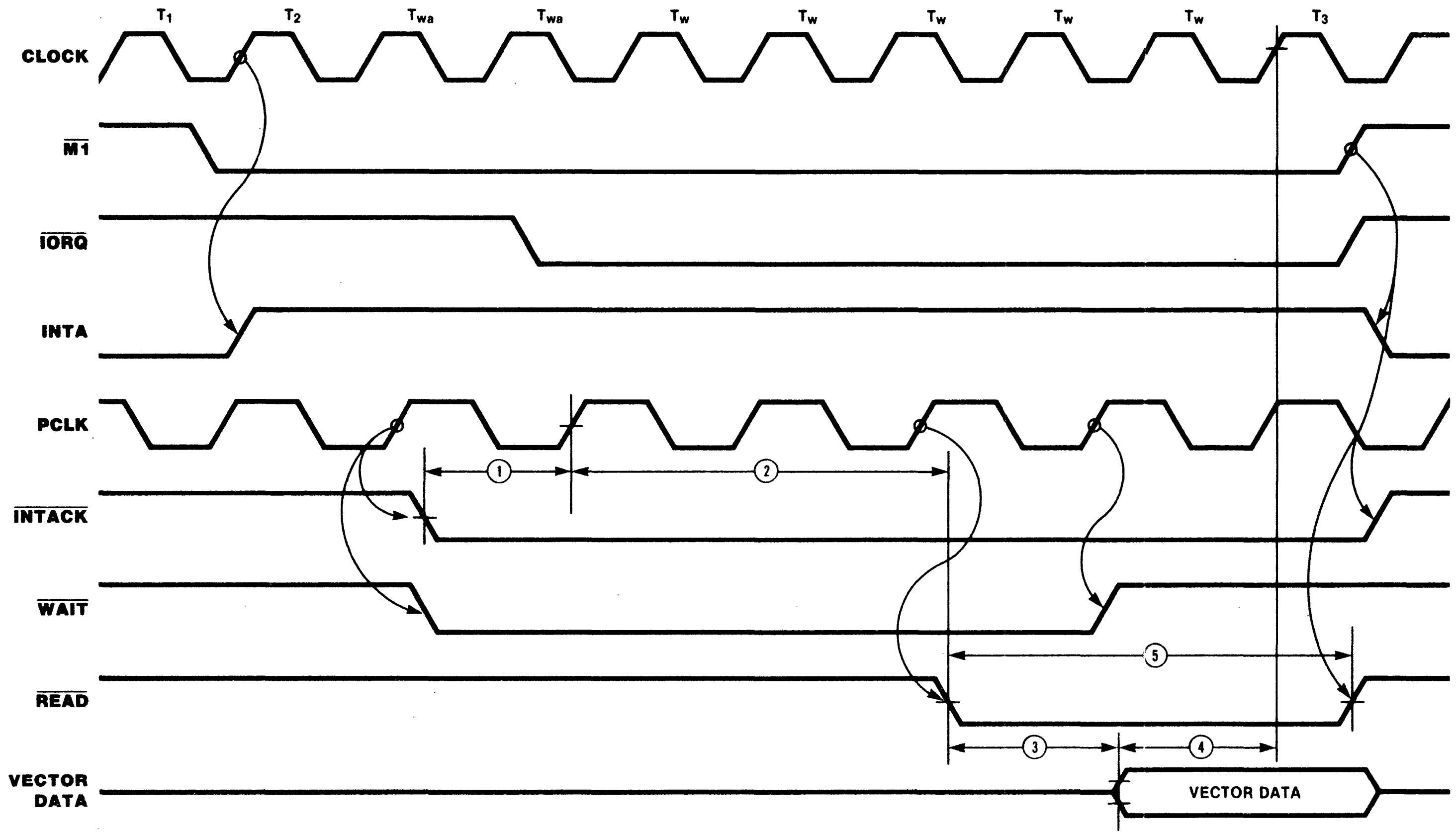


Figure 8b. Z80H CPU to Z8500A Peripheral Interrupt Acknowledge Interface Timing

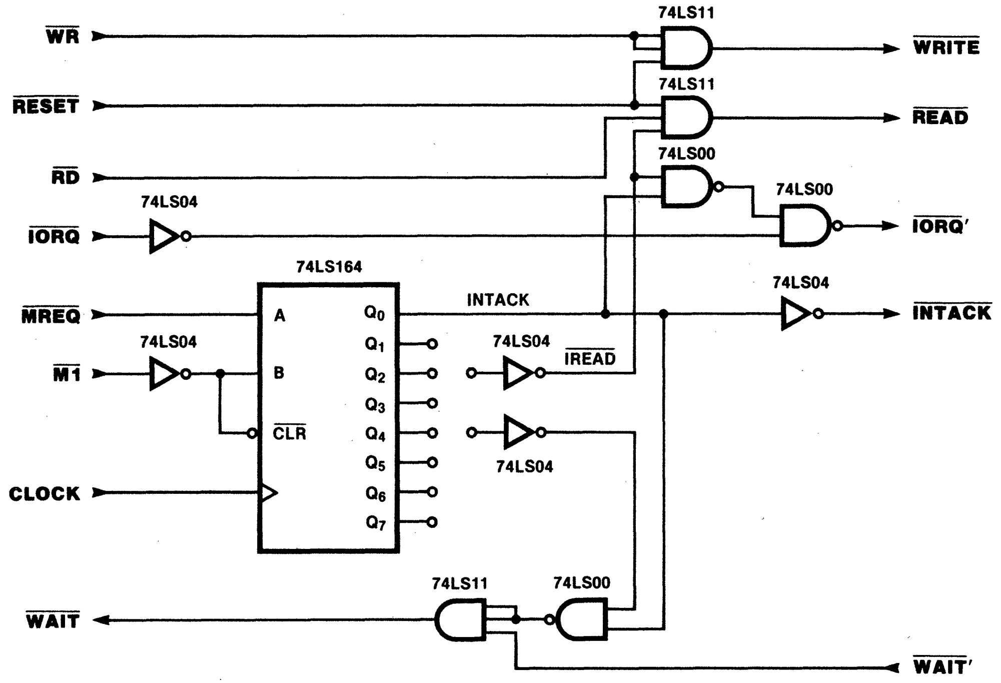


Figure 9. Z80 and Z8500 Peripheral Interrupt Acknowledge Interface Logic

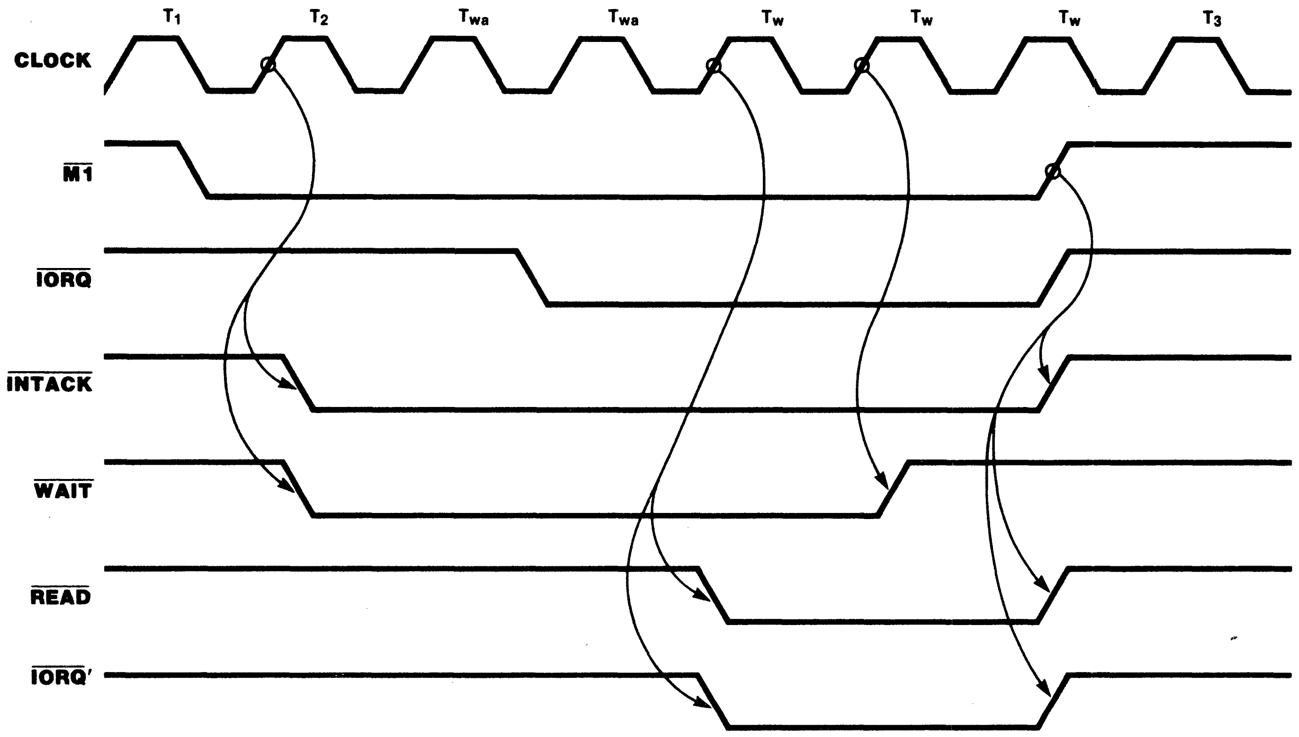


Figure 10. Z80 and Z8500 Peripheral Interrupt Acknowledge Interface Timing

SOFTWARE CONSIDERATIONS — POLLED OPERATION

There are several options available for servicing interrupts on the Z8500 peripherals. Since the vector or IP registers can be read at any time, software can be used to emulate the Z80 interrupt response. The interrupt vector read reflects the interrupt status condition even if the device is programmed to return a vector that does not reflect the status change (SAV or VIS is not set). The code below is a simple software routine that emulates the Z80 vector response operation.

Z80 Vector Interrupt Response, Emulation by Software

```
          ;This code emulates the Z80 vector interrupt
          ;operation by reading the device interrupt
          ;vector and forming an address from a vector
          ;table.  It then executes an indirect jump to
          ;the interrupt service routine.

INDX:     LD        A,CIVREG        ;CURRENT INT. VECT. REG.
          OUT       (CTRL),A        ;WRITE REG. PTR.
          IN        A,(CTRL)        ;READ VECT. REG.
          INC       A               ;VALID VECTOR?
          RET       Z               ;NO INT - RETURN
          AND       00001110B       ;MASK OTHER BITS
          LD        E,A
          LD        D,0             ;FORM INDEX VALUE
          LD        HL,VECTAB
          ADD       HL,DE           ;ADD VECT. TABLE ADDR.
          LD        A,(HL)          ;GET LOW BYTE
          INC       HL
          LD        H,(HL)          ;GET HIGH BYTE
          LD        L,A             ;FORM ROUTINE ADDR.
          JP        (HL)            ;JUMP TO IT

VECTAB:   DEFW      INT1
          DEFW      INT2
          DEFW      INT3
          DEFW      INT4
          DEFW      INT5
          DEFW      INT6
          DEFW      INT7
          DEFW      INT8
```

A SIMPLE Z80-Z8500 SYSTEM

The Z8500 devices interface easily to the Z80 CPU, thus providing a system of considerable flexibility. Figure 11 illustrates a simple system using the Z80A CPU and the Z8536 Counter/Timer and Parallel I/O Unit (CIO) in a mode 1 or non-interrupt environment. Since interrupt vectors are not used, the $\overline{\text{INTACK}}$ line is tied High and no additional logic is needed. Because the CIO can be used in a polled interrupt environment, the $\overline{\text{INT}}$ pin is connected to the CPU. The Z80 should not be set for mode 2 interrupts since the CIO will never place a vector onto the data bus. Instead, the CPU should be placed into mode 1 interrupt mode and a global interrupt service routine can poll the CIO to determine what caused the interrupt to occur. In this system, the software emulation procedure described above is effective.

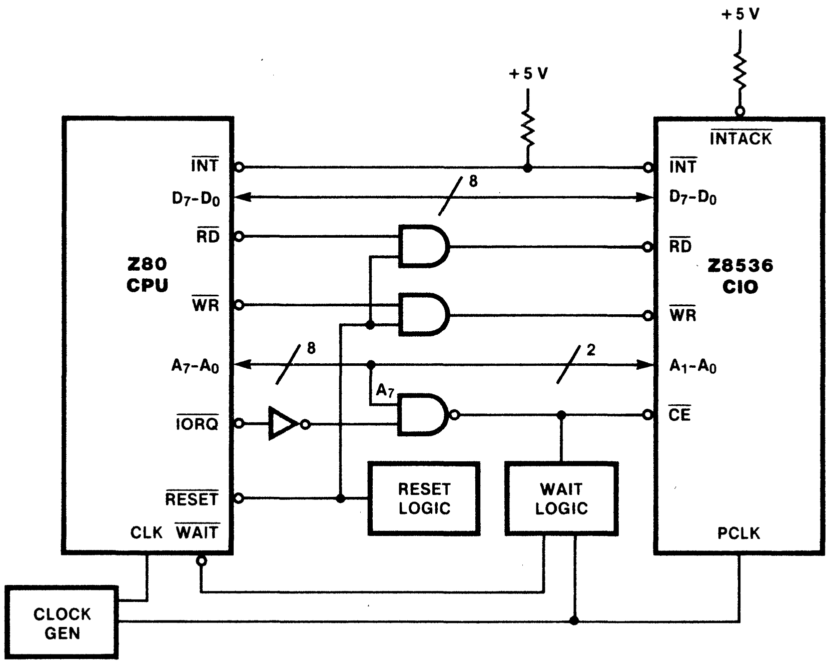


Figure 11. Z80 to Z8500 Simple System Mode 1 Interrupt or Non-Interrupt Structure

Additional Information - Zilog Publications

1. Z80 CPU Technical Manual	(03-0029-01)	7. Z80 Family Interrupt Structure Tutorial	(611-1809-0003)
2. Z80 DMA Technical Manual	(00-2013-A0)	8. Z8530 SCC Technical Manual	(00-2057-01)
3. Z80 PIO Technical Manual	(03-0008-01)	9. Z8536 CIO Technical Manual	(00-2091-01)
4. Z80 CTC Technical Manual	(03-0036-02)	10. Z8038 FIO Technical Manual	(00-2051-01)
5. Z80 SIO Technical Manual	(03-3033-01)	11. Zilog 1982/83 Data Book	(00-2034-02)
6. Z80H CPU AC Characteristics	(00-2293-01)		

Zilog

January 1989

Z80 Family Questions & Answers

This application note contains the most commonly asked questions about the Zilog Z80 Family. They are divided into following sections:

- Z80 CPU
- Z80 DMA
- Z80 PIO
- Z80 CTC
- Z80 SIO, Z80 DART

Obviously, not every questions on Z80 Family components are answered. However, this application note should give you a good feel for the Z80 Family devices. Along with the technical Manual, Product Specification and some other application notes, it should help make your Z80 design family a little easier. Also, this Application Note is applicable to the Z80 KIO and other Z80 family based Super Integration Devices.

Z80 CPU

Q: Are the Z80 CPU 6 and 8MHz clocks sensitive like their predecessors?
A: Yes, specifications for rise and fall times and clock voltage levels must be met.

Q: Can the rising edge on the CLK input affect the operation of the CPU?
A: Very much so. For NMOS devices, a negative voltage spike on any pin without back bias will forward-bias the diode that exists between the N+ material connected to the pad and p-type substrate. This action causes the injection of many electrons into the substrate. Once in the substrate, they are free to drift into any region of higher potential, which is the N+ region at Vcc of storage nodes storing a "1". Since storage holds don't store much charge (in order to minimize capacitance), these electrons in the substrate can be swept across the junction and destroy the "1" stored there. This reaction obviously affects the operation of the part.

Also, on CMOS devices, positive spikes on any pin exceeding Vcc voltage could cause "Latch-up"!

Q: What is the clock input impeadance (load)?
A: Capacitive load only (35pF max).

Q: Will Non-maskable interrupts continue occurring and executing if the NMI line pulses prior to the finish of the service routine?
A: Yes. Non-Maskable interrupts can not be disabled by user. Even though Non-Maskable Interrupts are negative edge triggered, if the input to the CPU pulses before termination of the service routine, then the service routine will begin again.

Q: How does the Non-Maskable Interrupt acknowledge cycle and RETN instruction actually work?
A: When a Non-Maskable Interrupt is acknowledged, interrupt flip-flop #1 (IFF1) is actually cleared to inhibit the acknowledgement of maskable interrupts. The state of interrupt flip-flop #2 (IFF2) is not altered. This is the only time that the contents of IFF1 and IFF2 can disagree. When the RETN instruction executes, the state of IFF2 is copied back into IFF1. This allows the state of maskable interrupts, before a Non-Maskable Interrupt, to be restored after service routine execution.

Q: How are subtraction operations performed?
A: Although the actual operation is probably a 2's complement addition, the flags are affected as if it were a logical subtraction operation.

Q: What is the setup time to recognize an NMI?
A: Through characterization of the CPU, Zilog has found that a setup time of 120nS (@ 4MHz) is required in order to assure that NMI is recognized before INT is recognized.

Q: What do the EI and DI instruction actually do?
A: Only the interrupt control flip-flops (IFF1 and IFF2) are affected by those instructions. The DI instruction will clear both IFF1 and IFF2 and prevent any further maskable interrupt from being recognized from that point on. The EI instruction will set both IFF1 and IFF2, but maskable interrupts will not be recognized till the completion of the next instruction.

Q: What is the status of the output drivers when the CPU is in a power-down situation?

A: When the CPU is without power, the output drivers appear to be in a high impedance state.

Q: How can I use the on chip refresh mechanism of the Z80 CPU to handle refreshing of 64K D-RAMs?

A: Here are some suggestions (assuming 256 cycle refresh):

1. Use an external counter to count 128 M1 cycles and toggle refresh address line A7.
2. Use external hardware to generate an NMI every 2mS and change the state of bit D7 in the R Register via software.
3. Use refresh address bit A6 to toggle the state of refresh address bit A7.

Q: Is there a method for testing hardware without removing the Z80 CPU from the socket?

A: Two methods are available:

1. Use BUSREQ to tri-state all control signals and then use external hardware to simulate the logic;
2. Remove power and ground from the CPU, all signals should go to a high-impedance.

Q: Does the CPU tristate M1 during reset?

A: No.

Q: Is Zilog going to add a 3.15 Volt current drive spec for designers using 74HCxx series of components?

A: No. There is a choice of either using 74HCTxx logic or using our CMOS Z80 CPU.

Q: If NMI is activated DURING reset, will the processor execute the NMI or address 0000h after reset goes high?

A: Since NMI input is "edge-triggered" input, if the CPU has active NMI "during" reset, CPU won't detect NMI and will execute the instruction at 0000h. If NMI goes low after RESET goes inactive, then CPU will process NMI.

Q: I've heard the CPU is a static device. Can I use the clock to single step it?

A: It's different for NMOS and CMOS.

NMOS: No, it violates the clock specs.
CMOS: Yes. You can do that.

Q: I don't seem to get the correct state of the interrupts when using the LD A,I and LD A,R instructions to read the state of IFF2. Why is this? How can I get around this?

A: On CMOS Z80 CPU, we've fixed this problem. On NMOS Z80 CPU, in certain narrowly defined circumstances, the Z80 CPU interrupt enable latch, IFF2, does not necessarily reflect the true interrupt status. The two instructions LD A,R and LD A,I copy the state of interrupt enable latch (IFF2) into the parity flag and modifies the accumulator contents (See table 7.0.1 in the Z80 CPU technical manual for details). Thus, it is possible to determine whether interrupts are enabled or disabled at the time that the instruction is executed. This facility is necessary to save the complete state of the machine. However, if an interrupt is accepted by the CPU during the execution of the instruction -- implying that the interrupts must be enabled -- the P/V flag is cleared. This incorrectly asserts that interrupts were disabled at the time the instruction was executed.

This paradox can be traced to the internal timing of the CPU. The problem is that the interrupt flip-flop (IFF2) is cleared before it is actually transferred to the P/V flag. The state of the interrupt enable latch is not copied into the parity flag until after the interrupt time, occurring during the execution of the instruction, has been accepted. Since the acceptance of the interrupt automatically clears the interrupt enable latch, the parity flag is also cleared, despite the fact that interrupts were enabled when the instruction started executing.

A neat solution to this anomaly relies on the fact that at least one item -- the old PC value -- is saved on the stack when an interrupt is accepted. The "next entry" position on the stack (the word below the address currently held in the stack pointer) may be cleared before execution of LD A,I (or LD A,R). If that zero value has changed by the time that the next instruction in the routine is executed, then an interrupt must have been accepted. This implies that interrupts were enabled, even if the state of the parity flag suggests that they were not. Of course, if the parity flag is found to be set after LD A,R (LD A,I) has been executed, there is no need to check the stack top. Interrupts are definitely enabled if the parity flag is in this state.

Two routines are listed here. Both return carry clear if interrupts are enabled, set otherwise. Both corrupt the A register; it does not contain the value in the I (or R) register on exit. The status of all flags except the carry flag are undefined on exit.

The first routine may be loaded anywhere in memory except "page zero" -- 0000h to 00FFh. This small restriction comes about because the routine checks only the most significant byte of the "next" stack entry. This byte will be non-zero after an interrupt has occurred if and only if the routine itself is not on page zero. The second routine tests both bytes of the "next" entry and, therefore, overcomes this restriction.

Caution, these routines presume that the service routine for any acceptable interrupt will re-enable interrupts before it terminates. This is almost always the case. They may not return the correct result if an interrupt service routine, which does not re-enable interrupts, is entered after the execution of LD A,I (or LD A,R).

Listing 1: This routine may not be loaded in page zero (0000h to 00FFh).

```
GETIFF:
        XOR    A      ;C flag, acc. := 0
        PUSH   AF     ;stack bottom := 00xxh
        POP    AF     ;Restore SP
        LD     A,I    ;P flag := IFF2
        RET    PE     ;Exit if enabled
        DEC    SP     ;May be disabled.
        DEC    SP     ;Has stack bottom been
        POP    AF     ;overwritten ?
        AND    A      ;If not 00xxh, INTs were
        RET    NZ     ;actually enabled.
        SCF           ;Otherwise, they really are
        RET           ;disabled.
        END
```

Listing 2: This routine may be loaded anywhere in memory.

```
GETIFF:
        PUSH HL       ;Save HL contents
        XOR    A      ;C flag, acc. := 0
        LD     H,A    ;HL := 0000h
        LD     L,A
        PUSH   HL     ;Stack bottom := 0000h
        POP    HL     ;Restore SP
        LD     A,I    ;P flag := IFF2
        JP     PE,
               POPHL ;Exit if isn't enabled
        DEC    SP     ;May be disabled.
        DEC    SP     ;Let's see if stack bottom
        POP    HL     ;is still 0000h.
        LD     A,H    ;Are any bits set in H
        OR     L      ;or in L ?
        POP    HL     ;Restore old contents.
        RET    NZ     ;HL <> 0 : isn't enabled.
        SCF           ;Otherwise, they really are
        RET           ;disabled.
POPHL:
        POP    HL     ;Exit when P flag is
        RET           ;set by LD A,I
        END
```

Q: Are all of the Z80 control lines internally synchronized?

A: The inputs in question are INT, NMI, BUSREQ, WAIT, and RESET. In the past, it seems that some of our customers have assumed that those inputs are totally asynchronous with respect to the system clock (i.e. no setup time required). Zilog's official position on this topic is as follows.

All asynchronous inputs to the Z80 family CPUs should be externally synchronized with the CPU clock. The required synchronization is specified by the setup and hold times for asynchronous inputs to the CPU. The synchronization is automatically provided for by the Z80 Family peripherals that are capable of driving the asynchronous inputs to the CPU.

In the Z80 CPU Technical Manual (Pages 70 and 72, footnote B), it is stated that "All control signals are internally synchronized so that they are totally asynchronous with respect to the clock." This statement should be amended to say "When interfacing the Z80 CPU to the Z80 family peripherals, the interface control signals are internally synchronized with the system clock by the peripherals themselves. When interfacing to the Z80 CPU with other devices, these control signals should be synchronized with respect to the system clock." Note that the former statement has been removed from the data book and the CPU product specification, but has not been removed from the technical manual yet.

The basis for the synchronization of the input control signals is the potential for the occurrence of a phenomenon called a "meta-stable state". The details of the meta-stable state are complex, but the concept is fairly simple. A meta-stable state occurs in bi-stable logic devices at the interface between an asynchronous and synchronous environment. All two-state logic devices spend some finite amount of time in the "linear region" (between the logic state of one and zero). The length of time spent in the linear region depends upon the switching speed of the device. If a synchronous system samples asynchronous inputs at the precise point in time that it passes through the linear region, the output of the sampling logic may spend time in an undefined logic state (the meta-stable state). The settling time to a valid logic state is proportional to the inverse exponential of the speed of the switching devices. More importantly, if the device in the meta-stable state is connected to several other bi-state devices in the system, the possibility exists for each of these bi-state devices to interpret the non-binary (or meta-stable) input differently. The final result can be an undefined or unpredictable state for a sequential state machine such as the Z80 CPU.

There are several points that should be remembered concerning these asynchronous inputs:

1. All interfaces between synchronous and asynchronous system that use clocked bi-stable devices are subject to the "meta-stable" phenomenon.
2. The probability of occurrence of a meta-stable state is directly proportional of the frequency of changes in the state at the interface and inversely proportional to the exponential of the switching speed of the devices used.

Q: How to interface the Z80 CPU to a 8259 using Mode 0 interrupt?

A: The Z80 CPU's interrupt mode "Mode 0" is the mode which maintains the "software compatibility" with the 8080, it is NOT fully compatible.
In this interrupt mode, during INTACK cycle, the Z80 CPU fetches the data on the bus as an "instruction" and executes it, like the 8080. However, from the hardware stand point, it's not true.
The 8080 generates three INTA pulses during the interrupt acknowledge cycle while the Z80 CPU generates only one INTACK signal (which can be decoded from M1 and RD).
This system works fine if you are not using the 8259 and put "RST" (restart) instruction onto the bus during the Interrupt Acknowledge cycle, which is a one byte instruction.
However, if you want to use the 8259 with the Z80 CPU, you'll have a problem. That is:

The 8259 expects three INTA pulses but the Z80 CPU generates only one INTACK cycle.

The best way to solve the problem is "simulating an 8080 interrupt acknowledge cycle" - which means generating a total of three "INTA" pulse for the 8259 from the Z80 CPU's interrupt acknowledge cycle by external logic. Following figure (Figure 1.) is the one example of the implementation.

This circuit works as follows (Assume that the instruction sent by the 8259 is "CALL" instruction):

On interrupt acknowledge cycle, the decoded INTA signal is sent as an INTA pulse for the 8259 and at the same time sets the LS74 to indicate that an interrupt acknowledge cycle has started.

On the following memory read cycle for the jump address on the call instruction, this circuit generates two additional INTA pulses for the 8259 and also masks off the read signal for the memory to avoid bus contention problems.

On the following write cycle, WR signal resets the LS74 to indicate that the interrupt acknowledge cycle is completed.

By using this circuit, you can use the 8259 with Z80 CPU.

Z80 DMA

Q: Does DMA recognize only 8-bit I/O addresses?

A: The DMA device does not care whether the I/O addresses or memory addresses are 8-bit or 16-bit. The Z80 DMA can address just as many I/O locations as it can memory locations.

Q: What is the importance on the placement of the "LOAD" commands?

A: The "LOAD" command only loads the contents of the source address register into the source address counter. The contents of the destination address register are automatically loaded into the destination address counter the first time the destination address gets incremented or decremented.

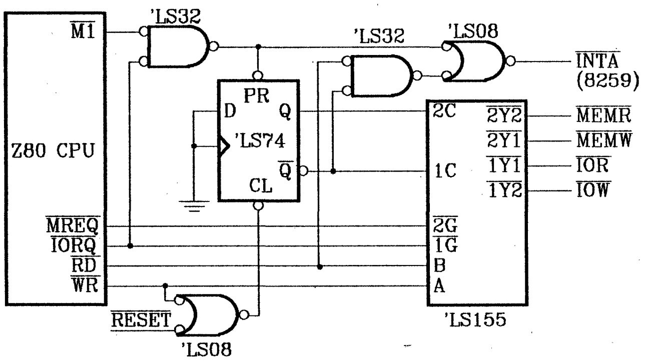


Figure 1. Z80 CPU to 8259 interface example

Q: When using the variable timing modes, are there any constraints in setting up the two ports that the user should be aware of?

A: Yes. When using the early cycle end timing feature of the DMA, it is strongly recommended that both ports be initialized with the same timing constraints.

Q: Is there any way to reset the DMA besides the RESET command and power-down?

A: With the CMOS DMA: On 44-pin PLCC package, there is a newly added "hardware reset pin" on pin 12 (This pin is left open on NMOS PLCC). Also, we've added special functions to the M1 signal line that allows you to reset C-MOS DMA. During an active M1 signal, without an active RD or IORQ, the DMA is reset. This feature is the same as that with Z80 PIO.

With NMOS, the only way to reset the DMA is by reset command. Actually, the RESET command can only reset the DMA if the CPU has control of the bus. if the DMA has control of the bus there is no way to reset it other than powering down the system (or the DMA).

Q: How long does power need to be removed from the DMA for an internal reset to occur?

A: Zilog tests the power-on reset circuit at 10mS. If the user is going to remove power from the DMA, Zilog recommends that it be done with the CLK input high.

Q: What limitations are not specified in the data book?

A: For NMOS DMA, when using the DMA in BURST mode with 2 cycle timing, an extra transaction is generated at the end of the burst.

For CMOS DMA, we've fixed all limitations.

Q: How can I use the DMA to transfer a page of information but do it one line at a time and wait between lines? (Printer application)

A: Operate the DMA in the Burst mode and use the printer I/O READY line to control DMA. Program the DMA for auto-restart mode to transfer the same "page" area continuously. When the printer is unable to accept a "line", the DMA will allow the CPU to control the bus.

Z80 PIO

Q: When using a port of the PIO in bit mode (mode 3), can any of the bits, programmed as outputs, affect the interrupt conditions set for recognizing inputs?

A: While it is undocumented, it is possible that the state of the bits programmed as outputs could be used as satisfying conditions for the mode 3 interrupt equation. It is recommended that all bits not needed for the interrupts be masked off.

Q: Can the PIO be programmed to provide a 16-bit input port and an 8-bit bidirectional port at the same time?

A: Yes. but there are some major concerns in doing it. Remember that when Port A is programmed into the bidirectional mode (mode 2), the handshake lines from Port B are used as input handshake lines for Port A. Some confusion occurs within the PIO if Port B is also programmed into the input mode (mode 1) and tries to use the handshake lines. A combination of software and hardware can be used to insure that data will not change until both ports can be read.

Q: Is the PIO port protected against hysteresis?

A: No.

Q: Do you have to strobe data into the port for proper mode 1 operation?

A: Yes, if you want to generate interrupts for mode 1 operation. If you only want to read the port data, then the STB input can be held low to make the input data latches transparent.

Q: How can I get Port B interrupt in Mode 3 and Port A interrupt in Mode 2?

A: You can get them, but it can cause severe interrupt conflicts if you choose that option. Port B interrupts are used by Port A in bidirectional mode for receive data interrupts. To prevent interrupt conflicts, Port B interrupts should be enabled, but all bits of Port B should be masked from affecting the interrupts. In the PIO Technical Manual it states that, "the same interrupt vector will be returned for a Mode 3 interrupt on Port B and an input interrupt during Mode 2 operation of Port A" (Section 5.3).

Q: Can the PIO control register be written while the PIO IUS bit is set?

A: Yes. But it is a safer programming practice to program the device after the RETI command.

Q: The on-chip power-on reset does not always work properly. How can I get around this?

A: Use the external hardware reset condition. Activate M1 for a minimum of two clock cycles without activating either RD of IORQ.

Q: When using the PIO in Mode 2, a 55h is written to the port. On the port side, an 0AAh is storobed into the port via BSTB. When the processor reads the data port, the 55h is read back instead of the 0AAh. Why and How?

A: The only way that the system can read the same data that it wrote into the PIO was if the ASTB signal was active (place the 55h onto the port bus) and the BSTB went active to strobe it into the data register. Suggest that system logic inhibit both strobe signals from becoming active during the same time.

Q: On which clock edge is the PIO reset (with M1 active and IORQ and RD inactive)?

A: The actual reset function will take place when the M1 signal goes low active (must have been active a minimum of 2 clock cycles).

Q: Can the PIO catch pending interrupts while interrupts are disabled?

A: Yes. Enabling the interrupts allow the interrupt daisy chain to function and the interrupt under service flip-flops to be set.

Q: A question came in concerning how the Z80 PIO handled its interrupts. Is the PIO capable of storing pending interrupts or must an interrupt be serviced and cleared (via either RESET or RETI) before another interrupt can be accepted?

A: It seems that the Z80 PIO interrupt structure is designed so that pending interrupts can be stored. There are to caveats to watch for in this however. The only way to store a pending interrupt is while another one is under service, and only one pending interrupt can be stored. Be aware that if you are operating in Modes 0,1 or 2, the transition of the STB signal can cause new data to be latched into the input data register and generate a pending interrupt. Be sure that any previous data can be read from the PIO before any new data is strobed in.

The storage for pending interrupts is only one deep. This means that a second interrupt condition cannot be stored if the first one has not been acknowledged.

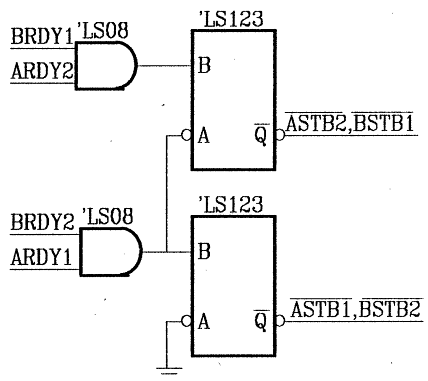
Q: Does an interrupt mask word have to follow the interrupt control word (assuming bit 4 was set) if the PIO is not programmed for Mode 3 operation?

A: Yes. Follow the interrupt control word with a dummy write to reset the PIO's write control logic.

Q: How can you get two PIOs to talk with each other in Mode 2 operation?

A: Suggest using ARDY1 and BRDY2 to generate a strobe pulse for ASTB1 and BSTB2. Same setup could be used for ARDY2 and BRDY1 and for ASTB2 and BSTB1. The logic basically consists of a 74LS123 (one shot) and a 74LS08 (AND gate). The ARDY1 and BRDY2 signals are ANDed together and supplied as B-TRG for generating ASTB1 and BSTB2. The ARDY2 and BRDY2 signals are ANDed together and supplied as the A-TRG for generating BSTB1 and ASTB2. The A-TRG for generating ASTB1 and BSTB2 is always low (grounded). In this manner, the port control signals are used to set the priority for strobe signal generation.

Please refer to Figure 2 and Table 1.



'LS123
BRDY1
'LS08
ARDY2
B
A
Q
$\overline{ASTB2}$,$\overline{BSTB1}$
'LS123
BRDY2
'LS08
ARDY1
B
A
Q
$\overline{ASTB1}$,$\overline{BSTB2}$

Figure 2. I/F circuit example

ARDY1	BRDY2	BRDY1	ARDY2	$\overline{ASTB1}$-$\overline{BSTB1}$	$\overline{ASTB2}$-$\overline{BSTB1}$	Direction
0	0	0	0	1	1	
0	0	0	1	1	1	
0	0	1	1	1	1	
0	0	1	1	1	0	2->1
0	1	0	0	1	1	
0	1	0	1	1	1	
0	1	1	0	1	1	
0	1	1	1	1	0	2->1
1	0	0	0	1	1	
1	0	0	1	1	1	
1	0	1	0	1	1	
1	0	1	1	1	0	2->1
1	1	0	0	0	1	1->2
1	1	0	1	0	1	1->2
1	1	1	0	0	1	1->2
1	1	1	1	0	1	1->2

Table 1. Truth Table for strobe signal generation

Q: How can the PIO be reprogrammed without having pending interrupts locking the system?

A: Try the following procedure.

1. Disable CPU interrupts;
2. Disable interrupt in the PIO;
3. Clear any pending interrupts within the PIO by using the interrupt control word with bit D4 set;
4. Reprogram the PIO as desired; and
5. Re-enable CPU interrupts.

Q: The PIO generates false interrupts during the programming sequence. What can cause this?

A: This symptom is almost always the result of a programming error. Depending upon the details of the problem, there are several solutions.

1. The interrupts should be enabled last in the initialization sequence. The Interrupt Control Word should be written with interrupts disabled so that the logical interrupt equation should be set (Mode 3). Finish the initialization with the Interrupt Enable Control WOrd (83H) to enable the interrupts.

2. The STB and RDY signals should be in a defined state. A transition on the STB input could cause a pending interrupt to be stored and executed as soon as interrupts are enabled.

3. A change in bit pattern (while in Mode 3) may cause an interrupt. A defined state for external inputs is recommended for power-up sequences.

Q: How can I get around the fact that only one "bit set" can be detected at a time in the OR bit mode?
A: One possibility would be to "mask" that bit during the interrupt service routine. Another possibility is to use external hardware to "mask" the bit.

Q: How can I get the PIO to give me interrupts on both transition of an input signal (Mode 3) ?
A: One method to use would require the PIO to be reprogrammed with a different logic equation during the interrupt service routine for the first transition. When the second transition occurs, then a new interrupt can be generated and the logic equation could be set back to its original state. Another method would require the use of external hardware to change the state of interrupting bit. Some possible logic could be to use an output port along with an exclusive-or (XOR) gate to control the state of the input bits.

Z80 CTC

Q: How does the software reset command to the CTC affect the rest of the CTC's operation?
A: The data book and the technical manual differ in the information that is presented on this subject. A software reset command stop the counter from counting any further. In order to start the counter again, a new time constant must be loaded into the time constant register. All bits in a mode control word will cause the operation of the CTC to be affected.

Q: What is the maximum frequency of the counter?
A: If external input is synchronized to the system clock, it's half that of the CLK (system clock) input. If it's not, 1/3 of the system clock.

Q: Are there any other uses for the CTC besides counting and timing?
A: Yes, the CTC makes a very nice interrupt controller for the Z80 bus. By programming the counter for a terminal count of one and defining the transition of the trigger, you can interface non-vectored interrupting devices onto the Z80 bus.

Q: How can I have control over an individual counter so that it cannot be started, stopped, and started again?
A: Use an external gate to qualify the clock input to the counter.

Q: When does the time constant (from the time constant register) get loaded into the down-counter?
A: On the first down count ---- ? verify.

Q: The CTC product specification states that no additional wait states (other than the automatic wait state inserted by the CPU) are allowed in the I/O cycles. Why?
A: It is not that the the wait states aren't allowed, it is just that they don't accomplish anything. The data will arrive at a particular time for the read cycles, and the internal write strobe is generated as a result of the clock edges that will be available. During the read cycle, it is possible that an improper value of the down-counter could be released onto the bus if additional wait states were added (the counter could change in the middle of the read operation).

Z80 SIO

This section contains the most commonly asked questions about the Zilog SIO. They are divided into following groups:

- Features
- Registers
- Interrupt
- Modem control signals
- Enable & Disable Tx & Rx, Auto enable mode
- Questions around DMA
- Internal timings
- External interface
- Asynchronous mode of operation
- Synchronous mode
- Questions about SDLC mode

Features

Q: What is the maximum data rate of the SIO?
A: 1/5 of the system clock rate. So it is 1.6Mb/s max for 8MHz version.

Q: What are the differences between Z80 SIO/0, /1, /2 and /4?
A: The differences between those four devices is "a combination of Channel B Modem signals". In fact, the SIO die itself has 41 pins internally. But a 40 pin DIP package has only "40 PINs", so we made three kinds of SIO's:

1. Z80 SIO/0: Have all channel B modem signals, exceptTxCB and RxCB, bonded together internally.

2. Z80 SIO/1: Lacks "DTRB".
3. Z80 SIO/2: Lacks "SYNCB".

For PLCC packages we are only offering "SIO/4", which covers all, since PLCC has 44 pins to bond out all signals.

Q: What are the differences between the SIO and Z80 DART?
A: The Z80 DART (Dual Asynchronous Receiver/Transmitter) is the device which only supports asynchronous mode of operation. The functionality, internal architecture and AC/DC characteristics are identical to the SIO in asynchronous mode. Also, pin assignment of it is identical to Z80 SIO/0 with the exception of one signal name. The "SYNC" pin on SIO/0 is "RI" (Ring Indicator) on SIO/0, but the functionality is exactly the same as the SIO/0 in asynchronous mode.

Registers

Q: How do you read the status registers?
A: Reads from RR0 (Read Register 0) are accomplished by simply doing a read from the SIO. Reads from RR1 or RR2 are accomplished by writing a register pointer to the SIO (WR0) and then doing a read operation.

Q: What happens when you read an empty FIFO?
A: You will read the last character in the buffer.

Q: How do you avoid an overrun in the receiver FIFO?
A: The receive buffer must be read before the recently received data character on the serial input is shifted into the receive data FIFO. This FIFO is three bytes deep. Thus, if the buffer is not read, the fifth character that just arrived caused an overrun condition. There is no set or reset bit to disable the buffering.

Q: When the FIFO gets locked due to an error condition, can it still receive?
A: The SIO continues to receive until an overrun error occurs.

Q: When does the FIFO buffer lock on an error condition?
A: The receive data FIFO gets locked when the following receiver interrupt modes are selected:

- Receive interrupt on Special condition only.
- Receive interrupt on First character or Special condition.

In both of these modes the special condition interrupt occurs after the character with the special condition has been read. The error status has to be valid when read in the service routine. The special condition locks the FIFO and guarantee that the DMA will not transfer any character until the special condition has been serviced.

Q: When a special condition occurs due to parity error, will a receive interrupt for that byte still be generated?
A: No. In the case of Receive interrupt on Special condition only mode, the interrupt will not occur until after the character with the special condition is read. In the case of Receive interrupt on First character or Special condition mode, the interrupt is generated on every character whether or not it has a special condition.

Q: What is the function of the Error FIFO?
A: The Error FIFO buffers the error conditions status bits for each of the received characters.

Q: When should the status in RR1 be checked?
A: Always read RR1 before reading the data.

Q: What information is contained in the Error FIFO?
A: End of frame, CRC/Framing error, Receive overrun error and Parity error. These are all contained in RR1 as well. The other status offered in RR1 is not part of an Error FIFO.
The Overrun and Parity error bits are held in the FIFO until they are reset by issuing the Error Reset Command. They will not be overwritten by new error information.

Q: How many register pointers does the SIO have?
A: The SIO has one for each channel. So it's possible to set the pointers for each channel first, then accessing each channel's register afterward. But it's not recommended, since program readability gets worse.

Interrupt

Q: What are the various Interrupting conditions?
A: The SIO can generate interrupts from the receiver, Transmitter and External/status for each channel (6 sources). This is a list of all conditions that could possibly generate an interrupt (one channel only listed):

Transmitter:	Transmit Buffer Empty
Receiver:	Receiver Character Available, Parity Error, Framing Error, Receive Overrun Error
External/Status: (Transition on DCD)	CTS, Sync/Hunt, Transmit, Underrun/EOM, Break/Abort Detection

Q: Can the IP bits be set while the SIO is servicing other interrupts?
A: Yes. If the interrupting condition has a higher priority than the interrupt currently being serviced it will cause another interrupt, thus nesting the interrupt service.

Q: How many levels of pending interrupts are there and how does the internal daisy chain operate?
A: Each possible source of an interrupt (6 possible) has one level of pending interrupts. The internal daisy chain operates in the same manner as would an external daisy chain.

Q: Does the RETI Instruction reset any status register?
A: No.

Q: If the CPU does not have the Return From Interrupt sequence (RETI instruction on the Z80 CPU), how may the SIO be informed of the completion of interrupt handling?
A: This may be done by writing the Return From Interrupt command (38h) to WR0 in Channel A of the SIO.

Q: Can the IUS bits be accessed?
A: No.

Q: When do IUS bits get set?
A: The IUS bits will be set during an interrupt acknowledge cycle on the falling edge of RD.

Q: When responding to an Interrupt, can you have the following sequence:

Int Ack, Disable INT, RETI, Clear interrupt condition?

A: No. The correct sequence is : Int Ack, Disable INT., Reset.

Q: Will enabling Interrupt after a transition on the Sync/Hunt bit cause Interrupt to occur?
A: No. External/Status Interrupt should be enabled before the transition occurs.

Note: It is advisable to execute the Reset Ext/Status Interrupt command in advance, so that the status of RR0, bit D4 reflects the current condition.

Q: Why is the Reset/Status Interrupt command recommended to be used several times in SIO setup?
A: Because many of the status bits that reflect interrupting conditions are latched bits and need to be reset to reflect current status rather than what may have occurred due to earlier Interrupts (changes in state).

Q: Will the SIO continue to request interrupt if the condition has not been satisfied?
A: Yes. There are several methods that can be used to clear the interrupt conditions. If it is a transmitter interrupt, then the transmitter must either be loaded with data or the Reset Transmit Interrupt Pending command must be issued. If the interrupt is for External/Status, then the Reset External/Status Interrupt command must be issued. If the interrupt is for a receive character being available, then the receive character must be read. If the interrupt is for an error condition, then the Error Reset command must be given.

Q: What conditions cause the transmit IP to be set?
A: Either the buffer empty or the flag after CRC is being loaded.

Q: How do the external/status bits affect the interrupts?
A: The external/status interrupt structure is affected by bits D7-D3 of RR0. These bits can be "reset" by either a hardware reset, a channel reset, or by the Reset External/Status command. The first status change on any one of the five bits after the reset will cause an interrupt to be issued and also will cause all five status bits to be latched. The latching effect is caused whether or not External/Status interrupts are enabled. If the current status at the time of reset is different than the latched status, then another Interrupt request is generated immediately. To clear the interrupt structure, two resets are necessary. The configuration of the SIO can change the definition of some of these signals. If the state of the bit changes across definition boundaries, an interrupt can be generated. Issue the Reset External/Status Interrupts command after definition. To process an external/Status interrupt, the Reset External/Status Interrupts command must be issued after reading these status bits and before the RETI.

Q: Can you use the SIO without an interrupt acknowledge cycle sequence (Z80 CPU)?
A: Reset the responsible interrupt pending bit (IP). The INT line will follow the IP bit.

Q: If the CPU can be interrupted but cannot be used with vectored interrupts, how should processing be done?
A: Immediately after being interrupted, proceed in a manner similar to polling the SIO for both receive and transmit. Alternatively, the Status affects vector bit (Bit D2 in WR1) may be set and a 0 byte placed into the interrupt vector register(WR2 in channel B). Then, the contents of the interrupt vector register can be used to determine the cause of the interrupt and the channel on which the interrupt occurred. This is queried by reading register RR1 of channel B. Also, IEI is tied high and M1 is tied high. No equivalent to an interrupt acknowledge is issued.

Q: When interfacing the SIO to the CPU other than the Z80 CPU, is it possible to assert M1 and IORQ at the same time as the Interrupt acknowledge cycle to simulate Z80 timing?

A: The SIO requires "Internal daisy chain settle time" even if you don't have devices other than the SIO on the interrupt daisy chain. The period for that purpose is "M1 is active but IORQ is inactive", and is at least 100nS (for 4MHz clock ; Parameter # 16, IEI-IEO delay time).

Modem control signals

Q: What is the state of the transmitter output when data is no longer available in the following modes?

a) Asynchronous?
b) Synchronous
c) SDLC?

A: a) In asynchronous modes, the transmitter goes into a marking state whenever all data has been sent.
b) In the synchronous mode, the SIO will send out 16 bits of CRC (2 bytes; if programmed and the Transmitter underrun/EOM Latch has been reset) followed by the appropriate number of Sync character. The line will then continue to idle sync characters.
c) In the SDLC mode, the SIO will send out 16 bits of CRC (2 bytes ; if programmed and the Transmitter underrun/EOM Latch has been reset) followed by the SDLC flag character (7Eh). The line will then continue to idle SDLC flag characters.

Q: What is the delay time for RTS/ to TxD?

A: Two Tx clocks for asynchronous and synchronous, 7 Tx clocks for SDLC.

Q: What is the delay time for the transmit buffer empty to RTS/?

A: Two Tx clocks for asynchronous gate delays for synchronous and SDLC.

Q: Does the frequency of the CTS or DCD signals have any adverse affects on the External/Status Interrupt? (even if auto enable is not programmed)?

A: Since every transition locks the External/Status latches, you could get constant interrupts (if External/Status Interrupt are enabled) or constant status latches.

Q: Is it possible to deactivate the DTR output without reprogramming WR5?

A: Only by resetting the channel or chip.

Q: Can you gate data by stretching the receive clock?

A: You can hold the clock until you have valid data. There are no maximum specs on the RxC period, and the edges are used to sample the data. If there are no edges, no data is sampled.

Enable&Disable Tx&Rx, Auto enable mode

Q: What happens to the character being assembled if the receiver becomes disabled?

A: Assembly of a character stops immediately and the character is lost.

Q: What happens to the characters already in the receive FIFO if the receiver becomes disabled?

A: They will remain in the FIFO until they are either read by the CPU or DMA, or until the channel is reset.

Q: When Auto enable bit is set, will DCD & CTS going true cause an Interrupt?

A: Interrupt will occur only on transition of DCD & CTS since both are edge triggered if WR1,D0 is set for Ext. Int enable.
However, since these are latched conditions in Status Register RR0 (D3 & D5), current status must only read after issuing Reset Ext/Status Interrupt command.

Q: In the auto enable mode, what happens when CTS goes inactive (High) in the middle of transferring a byte?

A: If the Auto Enable mode is selected, the CTS pin is an enable for transmitter (Ideally, Transmitter enable bit is ANDed with the status of CTS). So when CTS is inactive, transmit stops immediately. (The data being shifted out will be sending out completely, however).

Questions around DMA

Q: Can the SIO operate with a DMA in full duplex on each channel?

A: No. The SIO has only one ready line per channel and can only operate in half duplex mode.
If full duplex operation is required under DMA control, both channels A & B need to be used ; One for transmit and one for receive.

Q: Can both channels make simultaneous DMA requests?

A: Yes.

Q: What happens when you program the SIO to interrupt on Buffer Empty and the DMA to act on Buffer Empty?

A: This would not be a wise thing to do. However the Interrupt occurs, the DMA will take over the bus before Interrupt has acknowledged. The buffer will be filled by the DMA and the Interrupt Request will go away due to a Buffer Full condition and the Interrupt Acknowledge will occur causing bus confusion. The same thing occurs on Receive buffer empty interrupt and DMA on Receive character.

Q: How can the SIO/DMA combination be used for synchronous communications and ensure that the CRC characters are also transmitted?
A: Try the following procedure:

1. Initialize the SIO for use of the READY function with a DMA controller and then poll (or interrupt on) external/status (not transmit buffer empty).
2. Initialize DMA controller for data transfer and bus release at end-of-block. DO NOT ENABLE DMA YET!
3. Send first byte of data to SIO for transmission followed by a Reset Transmit Underrun/EOM Latch command.
4. Enable the DMA controller now (it should take control of the bus).
5. When the end-of-block is reached, the DMA controller should release the bus back to the CPU.

Q: When does the SIO terminate the READY signal?
A: The rising edge of the system clock that samples IORQ low causes READY to go inactive. The delay is specified by parameter 19 in the data sheet.

Q: When does the READY signal become active after an access to the SIO?
A: The READY signal will be inactive for a minimum of 5 clock cycles and will become active again 700 nS after CE goes inactive.

Internal timing

Q: When the transmitter is disabled, when does the TxD line go to a marking state?
A: One bit time after the last bit of the data leaves the transmit shift register.

Q: When the transmitter is empty, does status register RR0, bit D2 indicate that the buffer is now empty or that the last data in the buffer is in the process of being shifted out?
A: It indicates the buffer is now empty. The status register has nothing to do with the transmit shift register.

Q: Does the Transmit interrupt occur when Transmit Buffer is empty or Transmitter itself is empty?
A: Interrupt occurs when the Transmit Buffer is empty.

Q: How many bit times from external clock is the Transmit Buffer Empty Interrupt delayed?
A: The interrupt occurs a maximum of 9 clock periods from the Txc clock edge that causes the buffer to become empty. The exact time is highly dependent upon the mode of operation and is transparent to the user.

Q: When is the data available at the top of the FIFO?
A: Data is available after a maximum of 13 clock periods from the rising edge of RxC.

Q: What is the delay time between transmit shift register to the TxD pin?
A: Two Tx clocks for asynchronous and synchronous, Seven Tx clocks (five for zero inserter, two for internal delay) for SDLC.

Q: Does an Interrupt occur on RxC for last data bit assembled or does it occur relative to the RxC, but delayed?
A: Interrupt occurs when data is moved from the receive shift register to FIFO. The relationship of this event is relative to an external clock edge. This relationship however, is of no concern to the user. There is, however a specific delay from the external clock edge to the interrupt, caused by internal SIO logic.

External interface

Q: Can a sloppy system clock cause problems in SIO operation?
A: Yes. The specs on this system clock are very tight and must be met to prevent SIO malfunction. The specifications are:

Symbol	Description	Min	Max	Unit
VIHC	Clock "H"	Vcc-0.6	5.5	Volt
VIHL	Clock "L"	-0.3	0.45	Volt

Clock rise/Fall time = 30nS each edge
(For N-MOS, 4MHz device).

Should there be any ringing or undershoot/overshoot on the clock input, the SIO could behalf in any number of indeterminable ways.

Q: Must the system clock, fed to the SIO, have a 50% duty cycle?
A: The duty cycle doesn't have to be 50% as long as the minimum specification is met.

Q: Are input control lines to the SIO synchronized to system clocks so that garbage may exist on the buses anytime before setup requirements are satisfied?
A: Yes.

Q: Since setup time for CE and IORQ may be satisfied during T2, is time T1 required?

A: If the Z80 CPU is being used, then T1 timing state is required in order to utilize the interrupt structure. (interrupt request, acknowledge, and RETI).
No, if not using the Z80.

Q: Do wait states have to be added to provide I/O response to the SIO in non-Z80 based systems? (The Z80 adds wait states automatically)

A: No. As long as setup times as specified for the SIO are met. The SIO does not know about wait states inserted by the Z80. The Z80 puts in wait states in order to match the Z80 SIO setup times.

Q: What pins are noise sensitive and should be strapped to avoid strange interrupts?

A: The Ext Sync pin, and any Ext status pin that is not used. Also, all inputs are sensitive to signal ringing and undershoot problems.

Q: Is M1 required if no Interrupts are used in the SIO?

A: No. M1 should then be tied high.

Q: Can you use the Ready output for an Interrupt request?

A: Yes, for byte move action in or out and Respond to Interrupt. However, it is not recommended to use Ready for Interrupt with the CPU.

Q: How long must RD and the other control signals remain active?

A: Although RD and IORQ are latched internally, they must remain active for a minimum of two system clock periods.

Q: Are there any timing specifications for "Access recovery time"?

A: No.

Asynchronous Mode

Q: Why are there different Clock factors?

A: These clock factors enable the SIO to sample the center of the data cell. In the X16 mode, the SIO divides the bit cell into 16 counts and samples on count 8.

Q: For asynchronous mode of operation, must the clock rates selected be the same for Receiver and Transmitter?

A: No. However, the multiplier for both RxC & TxC must be the same because of internal logic.

Q: When running in the Async mode, is it necessary to use the X16 clock scalar?

A: No, X1 synchronization can be selected but the user must maintain data synchronization with the clock. The start bit detection logic does not work in X1 mode and the 1.5 stop bit cannot be used in X1. In other words, X1 Mode for Async mode is NOT asynchronous mode, its a "clocked serial channel".

Q: What does the SIO recognize as the Break character ?

A: A character of all zeros including stop bits (indicating a framing error).

Q: When attempting to detect a break condition by sensing the break/abort status bit, is it necessary to enable External/Status interrupts?

A: No. The External/Status latches work regardless of whether or not the external/status interrupts are enabled. This can be confusing because once the latches are strobed, the status in RR0 is frozen until a Reset External/Status Interrupt command is issued. If you desire the true current status, issue this command before reading RR0.

Q: Can a break sequence be sent for a fixed number of character periods?

A: Yes. Break is continuously transmitted as logic 1 by setting bit 4 of WR5. You can then send characters to the transmitter as long as the break level persists. A Break signal rather than the characters sent is transmitted, but each bit of each character sent will be clocked as if it were transmitted. The All sent bit, bit 0 of RR0, is set to 1 when the last bit of a character is clocked for transmission. This may be used to determine when to reset bit 4 of WR5 and stop the Break signal.

Q: If a Break sequence is initiated by setting bit 4 of WR5, will any character in the process of being transmitted, be completed?

A: No. Break is effective immediately when bit 4 of WR5 is set. The "all sent" bit in RR1 should be monitored to determine when it is safe to initiate a Break sequence.

Q: When using the SIO only in Asynchronous mode, can the SYNC pin have any use?

A: It may be used as a general purpose input. For example, by connecting it to a modem ring indicator, the status of that ring indicator can be monitored by the CPU.

Q: How can the SIO be used to transmit characters containing fewer than 5 bits?

A: First, set bit 6 and 5 in WR5 to indicate that five or fewer bits per character will be transmitted. The SIO then determines the number of bits to actually transmit from the data byte itself. The data byte should consist of zero or more 1s, three zeros, and the data to be transmitted. Thus, beginning the data byte with 1111001 will cause only the last bit to be transmitted.

Contents of data bytes(D=arbitrary value)

	D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	1	1	0	0	0	d
2	1	1	1	0	0	0	d	d
3	1	1	0	0	0	d	d	d
4	1	0	0	0	d	d	d	d
5	0	0	0	d	d	d	d	d

Synchronous Mode

Q: Can you cause interrupts on CRC error bit (RR1;D6) changes?

A: No. The CRC error status is not one of the special receive conditions. Perhaps, explanation of cyclic redundancy block checking and how the SIO operates for CRC is relevant.

CRC

Cyclic Redundancy Checking is a method of checking for errors in serial data transmission. It is also known as the polynomial error code check. The polynomial is an algebraic function used to create a constant from the message bit pattern. This constant, generated and accumulated in both the Transmitter and Receiver, is used to divide the binary numeric value of the character. The quotient is discarded and the remainder added to the next character, which again is divided. This continues until the last character, when the remainder is transmitted to the receiver for comparison with the Receiver's remainder. An equal comparison indicated no errors, while an unequal comparison indicates an error in transmission.

SIO-CRC

The SIO contains CRC generation and checking in the Transmitter and Receiver.
It allows for either of two polynomials to be used.

a) $X^{16}+X^{15}+X^2+1$: Called CRC-16, generally used in synchronous communication.

b) $X^{16}+X^{12}+X^5+1$: Called CRC-CCITT, generally used in SDLC communication and also recommended by the CCITT.

CRC Error Check

Status register RR1 which contains error conditions, allocates bit D6 for CRC error status. Since CRC checking is a continuous process and takes place character by character and intermediate results are shifted into the Receive Error FIFO continuously, bit D6 of RR1 is continuously updated because it is not latched.

However, checking the status of this bit at any intermediate point in time in the middle of a transmission is meaningless. It must be remembered that the result of a CRC check is valid only on completion of a message. Also, in most cases, bit D6 will usually be a "1" in the middle of a message since most serial bit combinations result in a non-zero CRC. Unless it is at completion of a message transmission.

The SIO does not generate an Interrupt for CRC Error Status.

Q: Suggest a hardware way to count or determine when the 16 or 20 bit times have passed before the CRC check is valid in BiSync mode?

A: Allow two "buffer full" interrupts to occur to determine that 16 bit times have elapsed, or have an external clock count 20 bit times.

Q: How do you read the CRC error status bit when receiving data in the bisync mode?

A: This is one possible method.
After two CRC bytes have been received and read, wait for the next receive character interrupted and stop CRC accumulation, then read the next received character. After the next character is interrupted, disable the receiver and read the status byte.

Q: In switched carrier Bisync application, the clock may go away before CRC calculate is complete since only one pad will be received. How can valid CRC be ensured?

A: SIO spec requires at least two pads for a valid CRC check in Bisync mode.

Q: Is CRC enabled automatically after first data in a non-SDLC node?

A: Only if it is programmed to be so.

Q: Are Sync patterns (or flags) included in CRC?

A: SDLC - No.
Yes for Bisync - CRC must be turned on/off as required or Sync will be included in CRC.

Q: In synchronous mode, does CRC get stripped from data?

A: Not normally, but it is possible if the CRC byte happens to match the contents of WR6 and the sync character load inhibit feature is enabled.
Otherwise, SIO won't delete the CRC bytes from the data stream.

Q: What is the proper sequence for a valid reading of the CRC error status bit?

A: To check the CRC error status and read the CRC bytes. The following sequence is recommended because of delays in Receive logic and the time at which EOM Interrupt occurs.

1. Interrupt, read and discard - 1st CRC byte.
2. Interrupt, read and discard - 2nd CRC byte.
3. Interrupt, read 1st pad character and discard.
4. Disable CRC
5. Interrupt, read CRC status then read 2nd pad and discard.

Q: In Monosync, is the Sync Comparison done in the Receive shift register or the Sync register?

A: Sync comparison is done in the Sync Register against the contents of WR7.

Q: For Monosync, which register contains the Sync character for comparison?

A: Write Register 7. Comparison is done in the Receive Sync Register.

Q: How does the SIO avoid losing a single sync character in the case of back-to-back Bisync messages that are separated by a single sync pad?

A: It does not. The SIO loses sync characters because the Bisync spec requires a minimum of two pad characters.

Q: Do Sync patterns (or flags) in data get stripped and still cause Interrupts?

A: All leading sync patterns (and all flags) are stripped automatically. In SDLC, sync characters (flags) will cause Interrupts if programmed to. Sync characters may or may not be stripped in Bisync depending on the state of the Sync Character Load Inhibit Bit (WR3,D1). Any data stripped from the data stream cannot cause a receive character available interrupt but may cause other interrupts (such as External/status for Sync/Hunt and special receive condition for EOM). In SDLC, programming Sync Character Load Inhibit will cause stripping of the address field and not cause Interrupts.

Q: Do interrupts occur after each Sync pattern?

A: Yes, if programmed to do so (External/Status interrupts).

Q: Do sync patterns automatically get transmitted in Bisync mode when Transmit Buffer becomes empty?

A: Yes, but the CRC bytes may be allowed to precede those sync characters.

Q: How does the SIO handle synchronous protocols which use less than 8-bit sync characters?

A: The sync character match logic within the SIO only makes comparison on 8-bit boundaries (8-bits for monosync and 16-bits for bisync). In order to match on patterns that are not integer multiples of 8-bits, the sync character must overlay the pattern stored in the sync register, or use "External Sync mode".

Q: Are sync characters subject to parity?

A: No.

Q: Assuming that there are characters available in the FIFO, what happens to them if the receiver goes into the hunt mode?

A: They will remain in the FIFO until they are either read by the CPU or DMA, or until the channel is reset.

Q: Is sync character transmission suppressed when the SIO is programmed for external sync operation?

A: Yes.

Q: How is it possible for the SIO to achieve synchronization on erroneous sync patterns (in monosync and bisync modes)?

A: The design of the SIO is such that the sync register serves as the CRC delay register after synchronization has been achieved. If the SIO goes out of synchronization or is placed into the hunt mode, the CRC delay register again becomes the sync register but it's contents are not cleared. Any data in it can be used by the comparison logic for synchronization. The best solution is to disable the receiver each time you place it in hunt mode and then re-enable it. This sequence will reset the contents of the sync register.

SDLC mode

Q: How does the SIO send CRC?

A: The SIO can be programmed to automatically send the CRC. First, write the first byte of the message to be sent. This guarantees the transmitter is full. Then, reset the Transmit Underrun/EOM latch (WR0;10h). Write the rest of the data frame. When the transmit buffer underruns, the CRC will be sent. The following table describes the action taken by the SIO for the bit oriented protocols.

Tx Underrun	Action	Comment
EOM Latch Bit	Upon Tx Underrun	
0	Send CRC+Flags	Valid Frame
1	Send Flags	Software CRC

Q: In SDLC mode, when do you get the End of Message (EOM) interrupt?

A: The EOF interrupt occurs after the 1st CRC is loaded to the transmit buffer and 2 bit times before the 2nd CRC is loaded to the buffer.

Q: How can you make sure that a flag is transmitted after CRC?
A: Use the external status End of Message (EOM) interrupt to start the CRC transmission, then enable the transmit buffer empty interrupt. When you get the interrupt, it means that the buffer is empty, a flag is loaded in the shift register, and you can send the next packet of information.

Q: When using the SIO in the SDLC mode of operation, the transmitter loses two data bits (gets shifted by two bit positions) when the last character before the closing flag is transmitted. Transmit data is looped to the receiver and CRC is not enabled?
A: The transmitter is working. The receiver actually causes the shift of two bits upon recognition of the closing flag.

Q: Why is the second CRC byte in the receiver FIFO not the full CRC byte?
A: The 2nd CRC byte read from the FIFO is not the full 8 bit byte. The transmitted byte is made up of the last two bits of the 1st CRC byte and the first 6 bits of the 2nd CRC byte. This is because of the delay in the Receive path and the point in time when the EOM Interrupt occurs causing transfer of contents of the Receive Shift Register into the FIFO.
However, since the above 2 bytes are to be discarded by the user, it does not matter.
Except in cases where users may want to include two bytes of data in place of CRC, it is important to note that the last byte will be off by two bits.

Q: In SDLC, when do you reset the CRC generator and checker?
A: The reset Tx CRC generator command should be issued when transmitter is enabled and idling (WR0). This needs to be done only once at initialization time for SDLC mode.

Q: If the SIO is idling flags and a byte of data is loaded into the transmit buffer, what will be transmitted?
A: Data takes priority over flags, and will be loaded into the shift register and transmitted.

Q: What does the SEND ABORT command do to the SDLC transmit sequence?
A: The transmission of the current character is aborted and a sequence of 8-one's are inserted into the data stream. This means that the user may see between 8 and 13 one's in the data stream because of the zero inserter. If there is data in the transmit buffer, it is destroyed and a Transmit Buffer Empty interrupt is pended.

Q: Can the SIO detect multiple aborts?
A: The SIO searches for seven consecutive 1's on the receive data line for the abort detection. This condition may be allowed to cause an external status interrupt. After these seven 1's are received, the receiver automatically enters Hunt mode, where it looks for flags. So even if more than seven 1's are received in case of multiple aborts, only the first sequence of 1's is significant.

Q: In the SDLC mode of operation, what is the relationship between the TxD output and transmitter interrupts?
A: Transmitter interrupts occur when the data from the transmit buffer is loaded into the transmit shift register. The output to the TxD pin is delayed by 6 bit times (five for the zero inserter and one for the pin delay) from the last bit leaving the shift register.

Q: Is it possible to monitor all received characters, including flags, in the SDLC mode?
A: No. if you want to monitor everything, then use the SIO in another mode of operation where the sync pattern has no special meaning.

Q: Can interrupts be generated on the idle line flag in SDLC?
A: Upon receipt of seven continuous ones, the break/abort bit will be set to indicate the abort condition. This bit will remain set until the SIO receives a zero.

Q: Does Hunt in SDLC continue until the Address Field has been recognized?
A: It does if the address search mode feature has been programmed.

Q: Does IBM SDLC specify parity?
A: No.

Q: Can the SIO include parity in SDLC mode?
A: Yes. It is appended at the end of the character.

Zilog

January 1989

Zilog Quality and Reliability

ZILOG'S QUALITY AND RELIABILITY PROGRAM

Introduction

The Zilog corporation is fortunate to have an excellent reputation for quality and reliability in its products. We recognize that the expectations of our customers is acceptable.

Zilog's Quality and Reliability Program is based on careful study of the principles laid down by such pioneers as W.E. Seming and J.M. Jurna and, perhaps even more important, observation of the practical implementation of those principles in Japanese, European and American manufacturing facilities.

The Zilog program begins with employee involvement. Whether the judgement of our performance is based on perfection in incoming inspection, trouble-free service in the field or timely and accurate customer service, we recongnize that our employees ultimately control these factors. Hence, our Quality Program is broadly shared throughout the organization.

1. Harmony Between Design and Process

High product quality and reliability in VLSI products is possible only if there is structural harmony between product design and the manufacturing process. Great care is taken to assure that the statistical process control limits observed within the manufacturing plants properly guardband the design technology used to configure the circuit and layout in Zilog's automated design methodology.

By use of a technique which we call Process Templating, the technology file in the automated design system periodically is updated to assure that product design parameters fall within the statistical control limits with which the process is actually operated.

In simple terms, the Process Template is the profile displayed by the process evaluation parameters which are automatically recorded from the test patterns on wafers as they proceed through the production line. These parameters are translated into the design technology file attributes such that the product design bears a key and lock relationship with the process.

2. Training

Product Design and Processing are people-dependent. Zilog training emphasizes the fundamentals involved in design for quality and reliability.

Customer Service, an important aspect of Zilog's quality performance as a vendor, also depends upon our people clearly understanding their jobs, and our obligations to our customers. This too is part of the curriculum administered by Zilog.

3. Order Acknowledgement Policy

One definition of vendor quality performance is that the vendor "does what he promises or acknowledges". Reliability and quality warranties can be met only if Zilog and the customer agree on product and delivery specifications. Zilog makes an extra effort by providing a series of documents as part of its purchase order acknowledgements. These clearly state what Zilog understands the specifications to be.

4. Test Guardbanding

No physical attribute is absolute. Customers' test methods may differ from Zilog's due to variations in test equipment, temperature or specification interpretation. To assure that every Zilog product performs to full customer expectations, Zilog uses a "waterfall" methodology in its testing. The earliest electrical tests made on the circuit, at the wafer probe operations, are guardbanded to the final test specifications. Final test specifications are guardbanded to the quality control outgoing sample. The quality control outgoing sample is guardbanded to the customer procurement or data sheet specifications. This technique of "waterfall" guardbanding assures that circuits which may be marginal to the customer's expectations are eliminated in the manufacturing process long before they get to the shipping container.

5. Probe at Temperature

Semiconductor devices tend to exhibit their most limited performance at the highest operating temperature. Therefore, it is Zilog's policy that all chips are tested at high temperature the very first time they are electrically screened, at the wafer probe station. The circuits are tested again at their upper operating temperature limit in the 100% final test operation.

6. Process Characterization

Before release to production, every process is thoroughly characterized by an exhaustive series of pilot production runs and tests which identify the statistical, electrical, and mechanical limits of which that particular process regime is capable. This documentation, which fills a large looseleaf binder for each process, is maintained as the historical record or "footprint" for that particular regime.

Process recharacterization is done any time there is a major process or manufacturing site change, and that documentation is added to the characterization history. The daily test site evaluation work recorded in the process template noted earlier in this presentation, demonstrates that the process remains in specification between times of formal characterization.

7. Product Characterization

Every Zilog product design is evaluated over extremes of operating temperature, supply voltage and clock frequency, prior to release to production. This information permits the proper guardbanding of the test program waterfall and identification of any marginal "corners" in design tolerances.

A product characterization report, which summarizes the more important tolerances identified in the process of this exhaustive product design evaluation, is available to Zilog's customers.

8. Process Qualification

Just as Zilog measures the robustness and reliability of its products by a qualification process separate from the performance characterization process, Zilog also qualifies every process prior to production by an exhaustive stress sequence performed on test chips and on representative products. Once a process regime is qualified, a process requalification is performed any time there is a major process change, or whenever the process template statistical quality limits are significantly exceeded or adjusted.

9. Product Qualification

In addition to characterization, every new Zilog product design is fully qualified by a comprehensive series of life, electrical, and environmental tests before release to production. Again, a qualification report is available to our customers which summarizes certain key life and environmental data taken in the course of these evaluations. Whenever possible, industry standard environmental and life tests are employed.

10. PPM Measurement, Direct and Indirect

It is frequently said that if you want to improve something, you need to put a measure on it. Therefore, Zilog measures its outgoing quality "parts per million" by the maintenance of careful records on the statistical sampling of production lots prepared for shipment. This information is then translated by our statisticians to a statement of our parts per million (or parts per billion) outgoing quality performance.

Of course, it is one thing for Zilog to think it is doing a good job in outgoing product quality and it is another for a customer to agree. Therefore, we ask certain key customers to provide us with their incoming inspection data which helps us calibrate our own outgoing performance in terms of the actual results in the field. The fact that Zilog has been awarded "ship to stock" status by many customers testifies to our success in this area.

11. Field Quality Engineers

It is also frequently said that "The customer is always right". If the customer has an application quality or reliability problem while using a Zilog product, whether it is Zilog's responsibility or not, we believe that we have a responsibility to resolve it. Therefore, Zilog maintains a force of skilled Applications Engineers who are also trained as field quality engineers and are available on immediate call to consult at the customers' locations on any problems they may be experiencing with Zilog product performance.

12. Product Analysis

As noted earlier, we feel that a customer problem is a Zilog problem. Accordingly, Product Analysis facilities, staffed by experienced professionals, exist at each Zilog site to provide rapid evaluation of in-process and in-field rejects to determine the cause and provide corrective action through a feedback loop into the production, design, and applications process. Zilog is pleased to share product analysis reports on specific products with the customer upon request.

13. FIT Measurement Direct and Indirect

Just as Zilog records its outgoing quality in terms of parts per million, it also measures its outgoing product reliability in terms of "FITS" or failures per billion device hours, using the results of weekly operating life test measurements on the circuits, performed in accordance with the standard specifications.

14. Test Site Step-Stress

The process evaluation test sites on the wafer are packaged and subjected to step-stress testing. Any drift in parameters under severe conditions of stress outside the norm is taken as an indication of possible process contamination or variation.

15. Statistical Process Control

Zilog employs statistical Process Control at all critical process steps. Deviations from norms must be evaluated by a Q/R review board.

16. Document Control

Skilled quality control professionals maintain careful and up-to-date specifications on all aspects of Zilog's products and processes in an elaborate document control system administered and controlled from the Zilog headquarters site. Specification changes and updates are electronically transmitted to the factory floor in order to assure that processing operations are being performed to the most up-to-date specifications. We are pleased to have a customer audit of this system at any time.

ZILOG'S QUALITY AND RELIABILITY SUMMARY

Zilog's Quality and Reliability program employs effective controls and gates. All quality control monitors are documented to ensure consistency of test methods, testing frequency, sample selection, sampling plan, reject disposition, and reporting format. Statistical Quality Control (SQC) charts are used to record the monitor results. This form of record keeping is used to ensure minimum process variation in such operations as ion-implant, diffusion, delineation, wire bonding, and plastic molding.

Zilog subjects each lot of finished goods to an independent electrical and mechanical quality control audit prior to shipment. The Quality Monitor Data in the next section is typical of the information gathered at this point. This quality monitor data is summarized in Figure I as an indicator of the company's progress on this report.

Z8/Super8 Microcomputer Family

	Part No	Unit Cost
Z8 DESIGN HANDBOOK	03-8275-02	12.50

Z8 NMOS MCU MICROCOMPUTERS
Z8600 Z8 MCU 2K 28-pin Product Spec
Z8601/03/11/13 Z8 MCU 2K/4K Product Spec
Z8671 MCU with Basic/Debug Interpreter
Z8681/82 Z8 MCU ROMless Product Spec
Z8691 Z8 MCU ROMless Product Spec
Super8 MCU ROMless Product Spec

Z8 CMOS MICROCOMPUTERS
Z86C08 MCU 2K 18-pin Product Spec
Z86C00/C10/C20 MCU 4K/8K 28-pin OTP Pd Spec
Z86C11/ MCU 4K
Z86C21/Z86E21/C12 8K/OTP Product Spec
Z86C91 MCU ROMless

Z8 APP NOTES AND TECHNICAL ARTICLES
Memory Space and Register Org App Note
A Programmer's Guide to the Z8 MCU
Z8 Subroutine Library
A Comparison of MCU Units
Z86xx Interrupt Request Registers
Z8 Family Framing

Z8 MCU Technical Manual

SUPER8 MCU MICROCOMPUTER
Z8800/01 MCU ROMless
Z8820 MCU 8K
Z8822 MCU 8K Protopak

SUPER8 APP NOTES AND TECHNICAL ARTICLES
Getting Started with the Zilog Super8
Polled Async Serial Operations with the Super8
Using the Super8 Interrupt Driven Communications
Using the Super8 Serial Port with DMA
Generating Sine Waves with Super8
Generating DTMF Tones with Super8
A Simple Serial Parallel Converter Using the Super8

Other Z8 Literature

	Part No	Unit Cost
Z8 Basic/Debug Software Manual	03-3149-02	5.00
Univ Obj File Utilities User's Mnl	00-8236-03	5.00
ASM 8 Cross Assembler User's Gde	00-8267-03	5.00

Z80/Z280 Microprocessor Family

	Part No	Unit Cost
Z80 FAMILY DATA BOOK	00-2480-01	10.00

Z80 NMOS/CMOS MICROCOMPUTERS
Z84C00 NMOS/CMOS Z80 CPU Prelimin Product Spec
Z84C01 Z80 CPU w/CTC
Z84C10 NMOS/CMOS Z80 DMA Product Spec
Z84C20 NMOS/CMOS Z80 PIO Product Spec
Z84C30 NMOSCMOS Z80 CTC Product Spec
Z84C40 NMOS/CMOS Z80 SIO Product Spec
Z8470 DART
Z84C80 Product Spec
Z84C90 CMOS Z80 KIO Product Spec
Z80180 Z180 MPU
Z280 MPU Preliminary Product Spec

Z80 APP NOTES AND TECHNICAL ARTICLES
Z80 Interrupt Structure
Using the Z80 SIO in Async Communications
Using the Z80 SIO with SDLC
Binary Synchronous Comm Using the Z80 SIO
Timing in Interrupt-Based System with Z80 CTC
Interfacing Z80 CPU's to the Z8500 Periph Family
A Z80 Based System Using the DMA with SIO
Z80 Q&A's
Package Information
Literature List
PSI List
Ordering Information

Z80 Technical Manuals

	Part No	Unit Cost
Z80 CPU Technical Manual	03-0029-02	15.00
Z80 CPU Programmer's Ref Guide	03-0012-04	7.00
Z80 DMA Technical Manual	00-2013-02	6.00
Z80 PIO Technical Manual	03-0008-02	9.00
Z80 CTC Technical Manual	03-0036-02	5.00
Z80 SIO Technical Manual	03-3033-01	7.50
Z180 Technical Manual	03-8276-01	12.00
Z280 Technical Manual	03-8224-02	15.00

Z8000/80,000 Microprocessor Family

Z8000/80,000 Microprocessor Family	Part No	Unit Cost
Z8000 FAMILY DATA BOOK	**00-2488-01**	**10.00**

Z8000/80,000 NMOS/CMOS MICROS
Z160 CPU Product Spec
Z320 CPU Product Spec
Z328 ICE
Z5380 CMOS SCSI Product Spec
Z7220A HPGD Product Spec
Z765A FDC Product Spec
Z8001/Z8002 CPU Product Spec
Z8010 MMU Product Spec
Z8016 Z-DTC Product Spec
Z16C20 CMOS Z-BUS GLU
Z80C30/Z85C30 CMOS SCC Product Spec
Z8030/8530 SCC Product Spec
Z8036/Z8535 CIO Technical Manual
Z8536 CIO Product Spec
Z8038/8538 FIO FIFO Product Spec
Z8060/8560 FIFO Product Spec
Z8068 Z-DCP Product Spec
Z8516 DMA (DTC) Product Spec
Z8581 Clock Generator Product Spec
Z80,000 CPU Product Spec

APP NOTES AND TECHNICAL ARTICLES
Interfacing Z80 CPUs to Z8500 Peripheral Family
Interfacing the Z8500 Peripheral to 68000
Design Considerations Using Quartz Crystals
Using Z8581 Clock Stretches in Z80 CPU Apps
Interfacing Z-BUS Periph. to the V20/V30/8086/8088
Interfacing the Z-BUS Peripherals Articles Reprint
Using SCC with Z8000 in SDLC Protocol
SCC in Binary Synchronous Communications
Z8000 Development Support
Zilog Quality and Reliability
Literature Guide
Ordering Information

Z8000/80,000 Technical Manuals

Z8000/80,000 Technical Manuals	Part No	Unit Cost
Z8000 CPU Technical Manual	**00-2010-06**	**12.00**
Z8000 Programmer's Pocket Guide	**03-0122-03**	**7.00**
Z8010 MMU Technical Manual	**00-2015-A0**	**4.00**
Z8030/Z8530 SCC Technical Manual	**00-2057-05**	**6.00**
Z8036/28536 CIO Technical Manual	**00-2091-02**	**8.50**
Z8038 Z-FIO Technical Manual	**00-2051-01**	**8.50**
Using Z8581 Clock Stretches in 280 CPU Applications Apps Note	**00-2807-01**	**1.00**
Z80,000 Technical Manual	**03-8225-01**	**17.50**
Memory Management w/Z80,000 Apps Note	**00-2324-01**	**1.00**

Components Military Literature

Components Military Literature	Part No	Unit Cost
ZILOG MILITARY PRODUCTS BINDER	**00-5498-01**	**N/C**
Z8681 ROMless Military Spec	00-2392-02	
Z800 1/2 CPU Military Spec	00-2342-03	
Z851 CGC Military Spec	00-2346-01	
Z8030 Z-SCC Military Spec	00-2388-01	
Z8530 SCC Military Spec	00-2397-01	
Z8036 Z-CIO Military Spec	00-2389-01	
Z8038/8538 FIO FIFO Military Spec	00-2463-02	
Z8536 CIO Military Spec	00-2396-01	
Z8400 Z80 CPU Military Spec	00-2351-02	
Z84C00 Military Spec	00-2441-03	
Z8420 PIO Military Spec	00-2384-01	
Z8430 CTC Military Spec	00-2385-01	
Z8440/1/2/4 Military Spec	00-2386-01	
Z80C30/85C30 Military Product Spec	00-2478-01	
Z84C20 PIO CMOS Military Spec	00-2384-02	
Z84C30 CTC CMOS Military Spec	00-2481-01	
Z84C40/1/2/4 SIO CMOS Military Spec	00-2482-01	

Note: Military Specs may be ordered individually

General Literature

General Literature	Part No	Unit Cost
Component Short Form Catalog	00-5472-04	N/C
Reliability Handbook	00-2475-02	N/C
Corporate Profile	00-3124-00	N/C

New Product Preliminary Specs

New Product Preliminary Specs	Part No	Unit Cost
Z86E21 OTP	00-2487-01	
Z16C30 USC	Jan '89	

ORDERING INFORMATION

Z80 CPU NMOS/CMOS

NMOS 4MHz

40pin DIP	44pin PLCC
Z0840004DSE	Z0840004VSC
Z0840004PSC	

CMOS 4MHz

40pin DIP	44pin PLCC	44pin QFP
Z84C0004DEE	Z84C0004VEC	Z84C0004FEC*
Z84C0004PSC	Z84C0004VSC	
Z84C0004PEC		

NMOS 6MHz

40pin DIP	44pin PLCC
Z0840006DSE	Z0840006VSC
Z0840006PSC	

CMOS 6MHz

40pin DIP	44pin PLCC	44pin QFP
Z84C0006DEE	Z84C0006VEC	Z84C0006FEC*
Z84C0006PSC	Z84C0006VSC	
Z84C0006PEC		

NMOS 8MHz

40pin DIP	44pin PLCC
Z0840008PSC	Z0840008VSC

CMOS 8MHz

40pin DIP	44pin PLCC	44pin QFP
Z84C0008PSC	Z84C0008VEC	Z84C0008FEC*
Z84C0008PEC	Z84C0008VSC	

CMOS 10MHz

40pin DIP	44pin PLCC	44pin QFP
Z84C0010PEC	Z84C0010VEC	Z84C0010FEC*

Z80 DMA NMOS/CMOS:

NMOS 4MHz

40pin DIP	44pin PLCC
Z0841004PSC	Z0841004VSC
Z0841004DSE	

CMOS 4MHz

40pin DIP	44pin PLCC
Z84C1004PEC	Z84C1004VEC
Z84C1004DEE	

Z80 DMA NMOS/CMOS: (Continued)

CMOS 6MHz

40pin DIP	44pin PLCC
Z84C1006PEC	Z84C1006VEC
Z84C1006DEE	

CMOS 8MHz

40pin DIP	44pin PLCC
Z84C1008PEC	Z84C1008VEC
Z84C1008DEE	

Z80 PIO NMOS/CMOS:

NMOS 4MHz

40pin DIP	44pin PLCC
Z0842004DSE	Z0842004VSC
Z0842004PSC	

CMOS 4MHz

40pin DIP	44pin PLCC	44pin QFP
Z84C2004DEE	Z84C2004VEC	Z84C2004FEC*
Z84C2004PSC	Z84C2004VSC	
Z84C2004PEC		

NMOS 6MHz

40pin DIP	44pin PLCC
Z0842006DSE	Z0842006VSC
Z0842006PSC	

CMOS 6MHz

40pin DIP	44pin PLCC	44pin QFP
Z84C2006DEE	Z84C2006VEC	Z84C2006FEC*
Z84C2006PSC	Z84C2006VSC	
Z84C2006PEC		

CMOS 8MHz

40pin DIP	44pin PLCC	44pin QFP
Z84C2008PSC	Z84C2008VEC	Z84C2008FEC*
Z84C2008PEC	Z84C2008VSC	

ORDERING INFORMATION (Continued)

Z80 CTC NMOS/CMOS:

NMOS 4MHz

40pin DIP	44pin PLCC
Z0843004DSE	Z0843004VSC
Z0843004PSC	

CMOS 4MHz

40pin DIP	44pin PLCC	44pin QFP
Z84C3004DEE	Z84C3004VEC	Z84C3004FEC*
Z84C3004PSC	Z84C3004VSC	
Z84C3004PEC		

NMOS 6MHz

40pin DIP	44pin PLCC
Z0843006DSE	Z0843006VSC
Z0843006PSC	

CMOS 6MHz

40pin DIP	44pin PLCC	44pin QFP
Z84C3006DEE	Z84C3006VEC	Z84C3006FEC*
Z84C3006PSC	Z84C3006VSC	
Z84C3006PEC		

CMOS 8MHz

40pin DIP	44pin PLCC	44pin QFP
Z84C3008PSC	Z84C3008VEC	Z84C3008FEC*
Z84C3008PEC	Z84C3008VSC	

Z80 SIO NMOS/CMOS:

4 MHz SIO/0

NMOS	CMOS
40pin DIP	40pin DIP
Z0844004DSE	Z84C4004DEE
Z0844004PSC	Z84C4004PEC
	Z84C4004PSC

4 MHz SIO/1

NMOS	CMOS
40pin DIP	40pin DIP
Z0844104DSE	Z84C4104DEE
Z0844104PSC	Z84C4104PEC
	Z84C4104PSC

Z80 SIO NMOS/CMOS: (Continued)

4 MHz SIO/2

NMOS	CMOS
40pin DIP	40pin DIP
Z0844204DSE	Z84C4204DEE
Z0844204PSC	Z84C4204PEC
	Z84C4204PSC

4MHz SIO/3

CMOS 44pin QFP

Z84C4304FEC*

4MHz SIO/4

NMOS 44pin PLCC	CMOS 44pin PLCC
Z0844404VSC	Z84C4404VEC
	Z84C4404VSC

6 MHz SIO/0

NMOS 40pin DIP	CMOS 40pin DIP
Z0844006DSE	Z84C4006DEE
Z0844006PSC	Z84C4006PEC
	Z84C4006PSC

6 MHz SIO/1

NMOS 40pin DIP	CMOS 40pin DIP
Z0844106DSE	Z84C4106PEC
Z0844106PSC	Z84C4106PSC

6 MHz SIO/2

NMOS 40pin DIP	CMOS 40pin DIP
Z0844206DSE	Z84C4206DEE
Z0844206PSC	Z84C4206PEC
	Z84C4206PSC

6 MHz SIO/3

CMOS 44pin QFP

Z84C4306FEC*

6 MHz SIO/4

NMOS 44pin PLCC	CMOS 44pin PLCC
Z0844406VSC	Z84C4406VEC
	Z84C4406VSC

8 MHz SIO/0

CMOS 40pin DIP

Z84C4008DEE
Z84C4008PEC
Z84C4008PSC

ORDERING INFORMATION (Continued)

Z80 SIO NMOS/CMOS: (Continued)

8 MHz SIO/2
CMOS 40pin DIP
Z84C4208DEE
Z84C4208PEC
Z84C4208PSC

8 MHz SIO/3
CMOS 44pin QFP
Z84C4308FEC*

8 MHz SIO/4
CMOS 44pin PLCC
Z84C4408VEC
Z84C4408VSC

Z80 DART NMOS:

4 MHz
40pin DIP
Z0847004PSC

6 MHz
40pin DIP
Z0847006PSC

Z180 MPU:

6 MHz 64pin

shrink DIP	**68pin PLCC**	**80pin QFP**
Z8018006PSC	Z8018006VEC	Z8018006FEC*
Z8018006PEC	Z8018006VSC	

8 MHz 64pin

shrink DIP	**68pin PLCC**	**80pin QFP**
Z8018008PSC	Z8018008VEC	Z8018008FEC*
Z8018008PEC	Z8018008VSC	

10 MHz 64pin

shrink DIP	**68pin PLCC**	**80pin QFP**
Z8018010PSC	Z8018010VEC	Z8018010FEC*
Z8018010PEC	Z8018010VSC	

Z280 MPU:

10 MHz
68pin PLCC
Z8028010VSC

Z84C01 CPU:

10MHz
44pin PLCC
Z84C0110VEC

Z80 GLU:

6 MHz
68pin PLCC
Z84C8006VSC

Z84C90 KIO:

8 MHz
84pin PLCC
Z84C9008VSC
Z84C9008VEC

* QFP package (package designator : F) will be available in Q2 '89.

- For military grade devices and the package types other than listed above, please contact your local Zilog sales office.
- Please check the availability before placing order.

ORDERING INFORMATION

CODES

PACKAGE
Preferred
D = Cerdip
P = Plastic
V = Plastic Chip Carrier

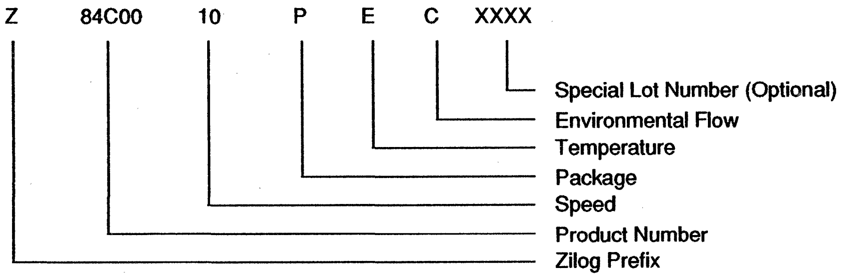
Longer Lead Time
C = Ceramic
F = Plastic Quad Flat Pack

ENVIRONMENTAL
Preffered
C = Plastic Standard
E = Hermetic Standard

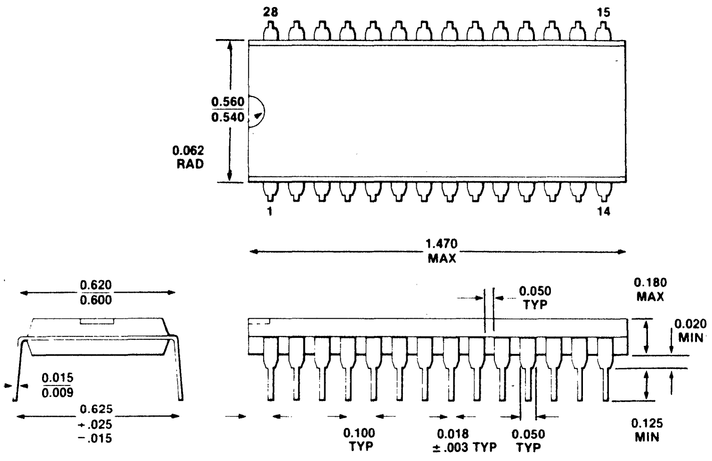
Longer Lead Time
A = Hermetic Stressed
B = 833 Class B Military
D = Plastic Stressed
J = JAN 38510 Military

TEMPERATURE
S = 0°C to + 70°C
E = -40°C to 100°C
M = -55°C to +125°C

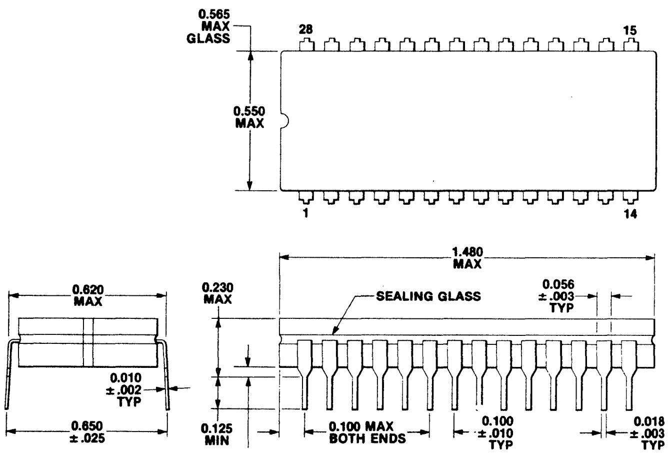
Example:
Z84C0010PEC is a CMOS 8400, 10 MHz, Plastic, -40°C to 100°C, Plastic Standard Flow.



PACKAGE INFORMATION



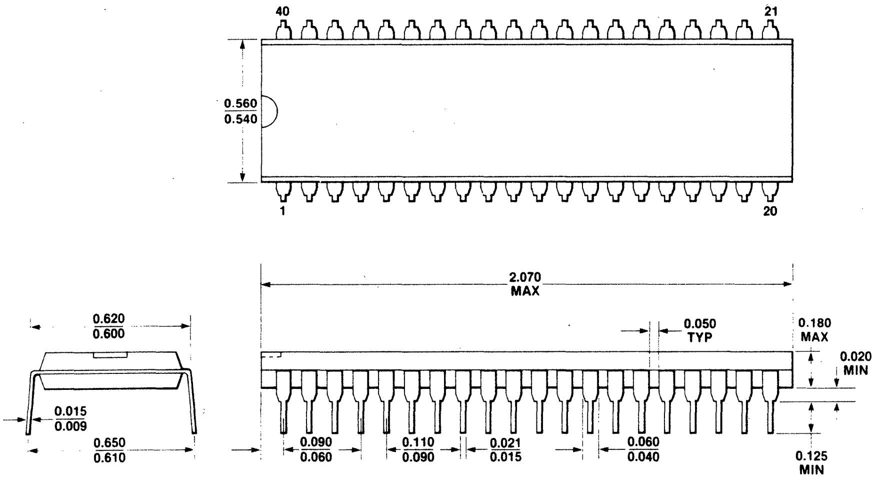
28-Pin Dual-in-Line Package (DIP), Plastic



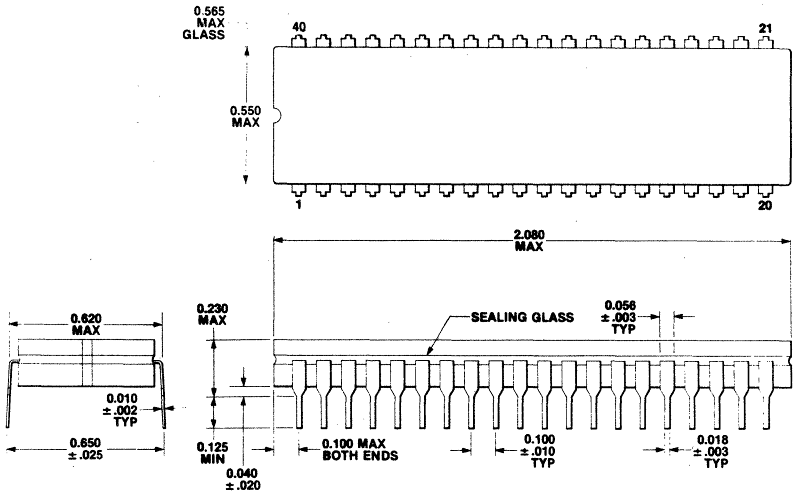
28-Pin Cerdip Package

NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

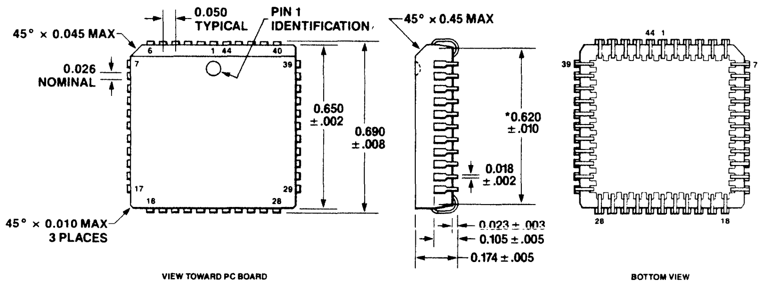
PACKAGE INFORMATION (Continued)



40-Pin Dual-in-Line Package (DIP), Plastic

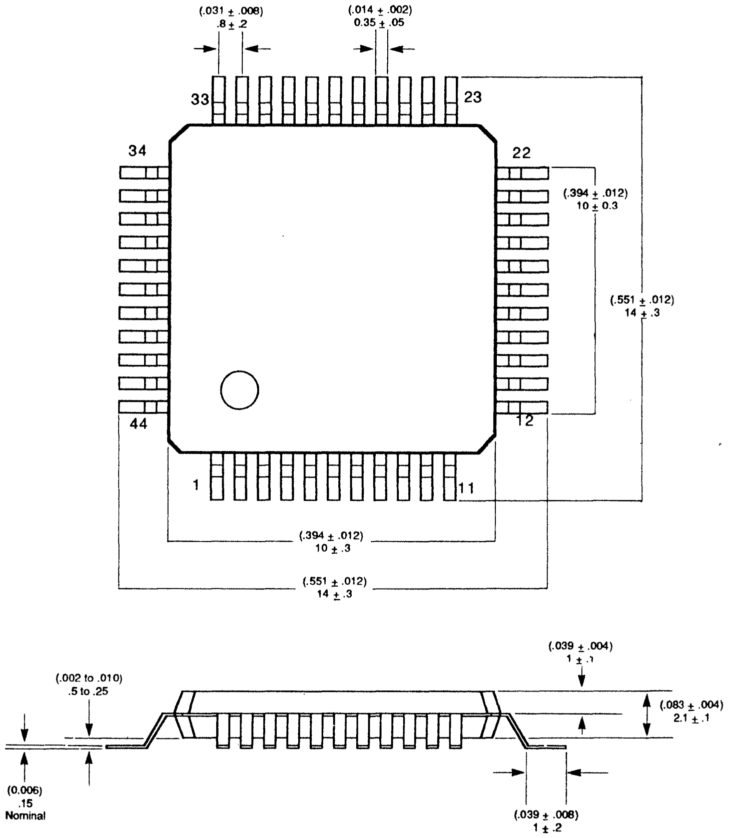


40-Pin Dual-in-Line Package (DIP), Cerdip



*DIMENSION FROM CENTER TO CENTER OF RADII

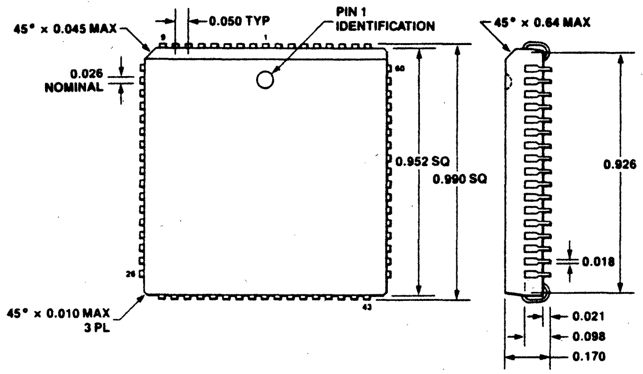
44-Pin Plastic Chip Carrier (PCC)



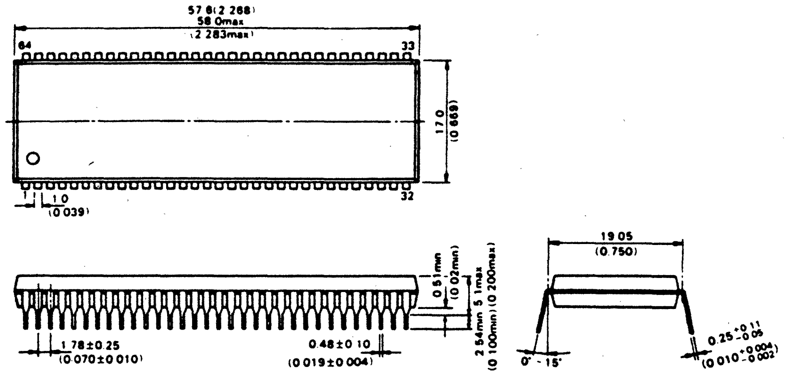
44-Pin Quad Flat Pack (QFP)

NOTE: QFP package dimensions are in millimeters
Units with ( ) are in inches.

PACKAGE INFORMATION (Continued)

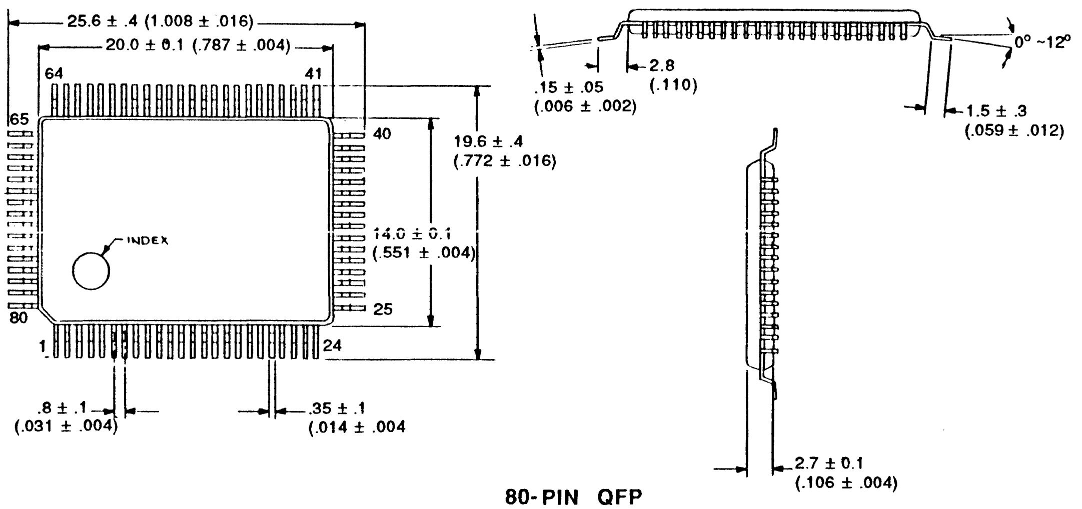


68-Pin Plastic Chip Carrier (PCC)

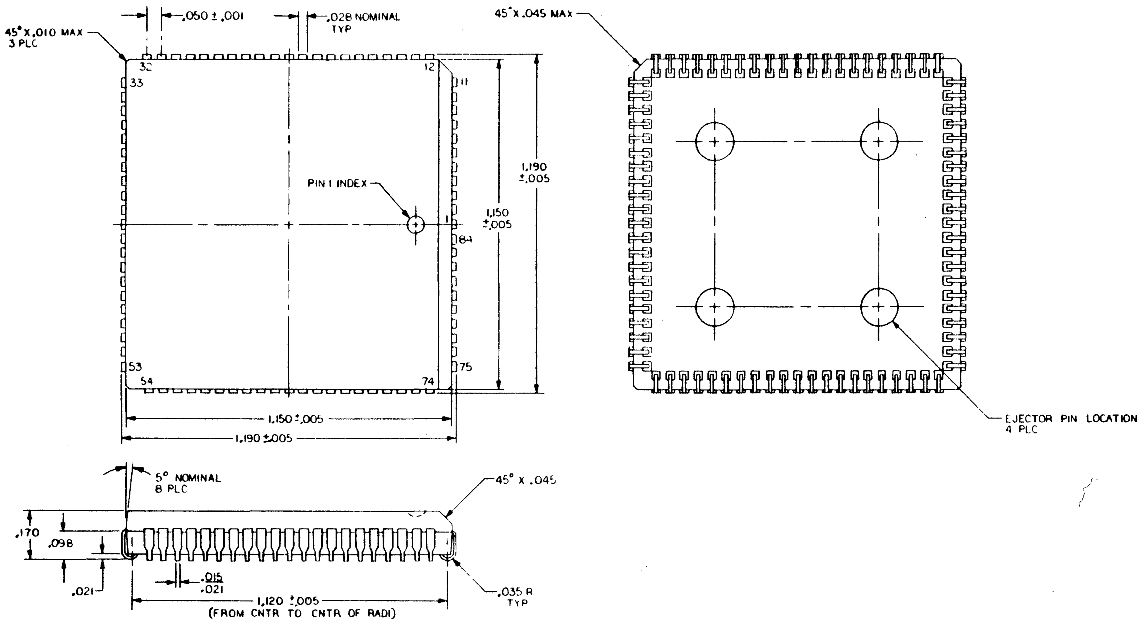


64-Pin Dual-In-Line Package (DIP)

PACKAGE INFORMATION (Continued)



80-PIN QFP



84-Pin PLCC

Notes

ZILOG DOMESTIC SALES OFFICES AND TECHNICAL CENTERS

CALIFORNIA	
Agoura	818-707-2160
Campbell	408-370-8120
Tustin	714-838-7800
COLORADO	
Boulder	303-494-2905
FLORIDA	
Largo	813-585-2533
GEORGIA	
Norcross	404-923-8500
ILLINOIS	
Schaumburg	312-517-8080
NEW HAMPSHIRE	
Nashua	603-888-8590
MINNESOTA	
Edina	612-831-7611
NEW JERSEY	
Clark	201-382-5700
OHIO	
Seven Hills	216-447-1480
PENNSYLVANIA	
Ambler	215-653-0230
TEXAS	
Dallas	214-987-9987
MARYLAND	
Bethesda	301-652-8104

INTERNATIONAL SALES OFFICES

CANADA	
Toronto	416-673-0634
GERMANY	
Munich	49-89-672-045
JAPAN	
Tokyo	81-3-587-0528
HONG KONG	
Kowloon	852-3-723-8979
KOREA	
Seoul	822-552-5401
SINGAPORE	
Singapore	65-235-7155
TAIWAN	
Taipei	886-2-741-3125
UNITED KINGDOM	
Maidenhead	44-628-392-00

Zilog, Inc. 210 Hacienda Ave., Campbell, CA 95008-6609
Telephone (408) 370-8000 TWX 910-338-7621

00-2490-01 Printed in USA