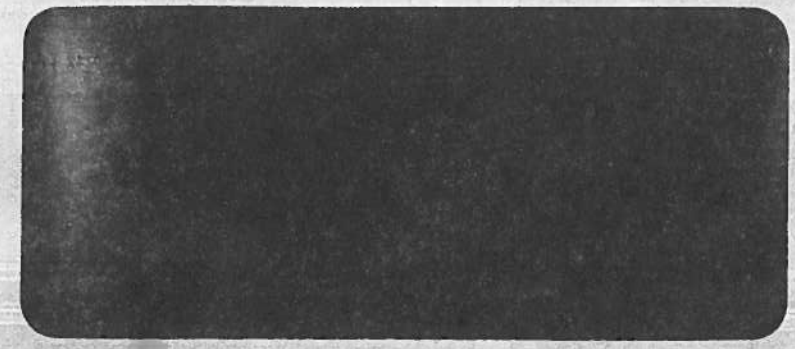


DY-4 SYSTEMS INC.

+5V 900 mA
+12 50 mA } measured
-12 30 mA

DSTD-102 CPU AND SERIAL I/O

OPERATION MANUAL

PREPARED BY dy-4 SYSTEMS INC
DATED JULY 11th, 1983

DY-4 SYSTEMS INC., 888 LADY ELLEN PLACE, OTTAWA, ONTARIO, CANADA K1Z 5M1 (613) 728-3711

NOTICE

CHANGE NOTICE

Revision 6 of the DSTD-102 contains several significant enhancements over revision 4 and earlier boards. These enhancements have been added followng custommer requests to take advantage of recent technology developments.

1. The DSTD-102 now supports 8 kbyte RAMs (Intel 2186) and 16 kbyte EPROMs. Note that the addition of this feature required the re-layout of the memory device jumper blocks JB9, JB10 and JB12. Refer to section 3.2 for a complete description.

2. The memory decode PAL now supports six different memory configurations, including 2K, 4K, 8K and 16K configuations in the one PAL

3. Using a jumper block the DSTD-102 will now support both "synchronous" edge triggered push button rest (available on rev 4 boards) and level sensitve resets (new). The level sensitive push button reset capability is required when using brown-out detection logic such as that on the DSTD 703.

4. This rev allows full access to the on-board I/O devices from other cards in the system. This is particularly useful when the DSTD-103 slave processor is being used.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1

1.0 GENERAL INFORMATION

1.1 Introduction

The dy-4 SYSTEMS' DSTD-102 CPU, Figure 1 - 1, is a Z80 based microcomputer board. It features a CPU chip, two serial communications channels, 4 counter/timers and three 28 pin memory sockets for byte-wide memory devices.

1.2 DSTD Series General Description

The DSTD series was designed to satisfy the need for low cost OEM microcomputer modules. The DSTD-Z80 BUS uses a motherboard interconnect system concept. The modules for the STD-Z80 BUS are a compact 4.5 x 6.5 inches which provides for system partitioning by function, e.g. CPU, Memory, I/O, etc. This smaller module size makes system packaging easier, while increasing MOS-LSI densities provide high functionality per module.

1.3 DSTD-102 Features

- Utilizes the powerful Z80 microprocessor
- Provides three 28 pin sockets which may be strapped to accept any combination of the following industry standard memory devices.

EPROM	STATIC RAM	ROM
2758 (1kx8)	4118 (1kx8)	
2759 (1kx8)		
2716 (2kx8)	4802 (2kx8)	MK34000 (2kx8)
2732 (4kx8)		
2764 (8kx8)	2186 (8kx8)	
27128 (16kx8)		

- Four cascadable counter/timer channels
- 2 serial RS-232C channels - Channel A has two additional RS-232C drivers and receivers for external clocking allowing full synchronous operation.
- Transmit and Receive LEDs on Channel A
- Fully buffered signals for system expandability

- Selectable reset address to either 0000H or E000H
- All on-board memory can be disabled and enabled under software control
- Selectable WAIT state generator for memory devices on all M1 cycles, MEMRQ cycles or all INTAK cycles
- Compatible with MDX-SST for single step operation during debugging
- 4MHz version available
- STD-Z80 bus compatible

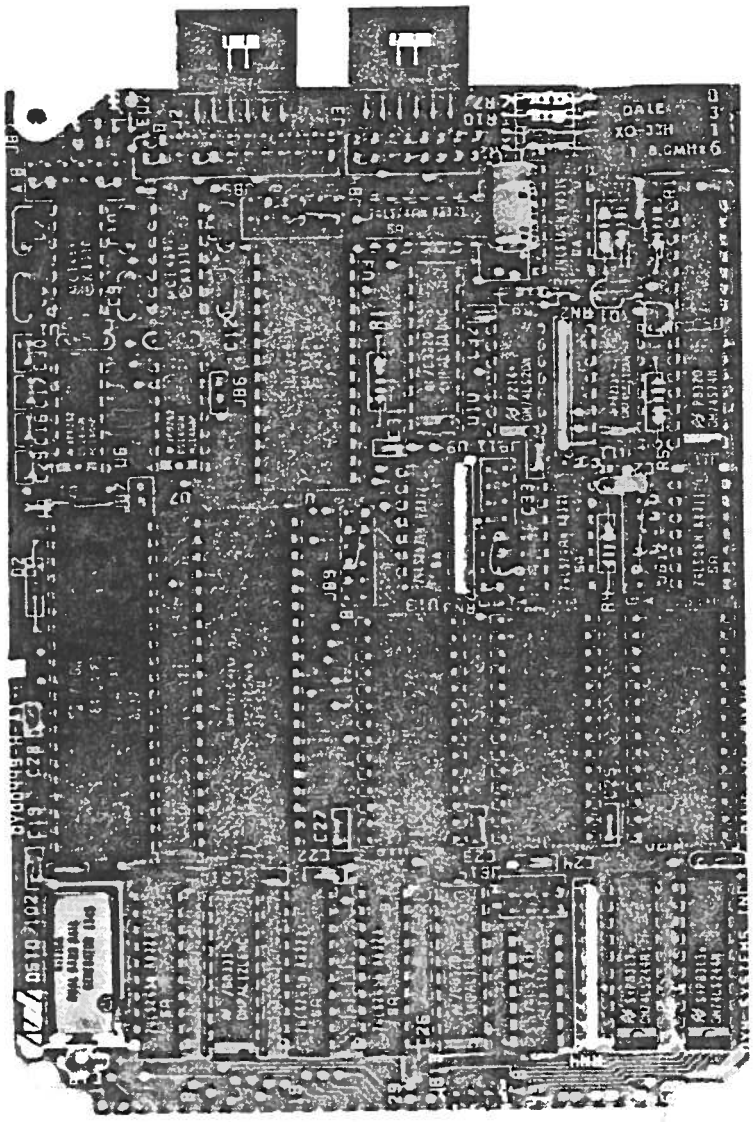


FIGURE 1 - 1 DSTD-102 MODULE

SECTION 2

2.0 FUNCTIONAL HARDWARE DESCRIPTION

2.1 Introduction

The DSTD-102 utilizes a Z80 microprocessor as the system controller. It features three 28 pin memory sockets which enables the user to populate the module with any combination of designated ROM and EPROM. Custom address decoding allows the user to configure the memory on any 8K boundary of the 64K memory map. A PAL decoder is supplied to allow the user to choose one of six popular memory configurations, or if desired the user may implement other mixtures of memory devices simply by programming the PAL accordingly.

A 4 channel counter/timer circuit is included for software controlled counting and timing functions. On-board strapping options make it possible to cascade the four CTC channels for long count sequences. The CTC may also be used as a baud rate generator for the serial channels if non-standard baud rates are required.

The DSTD-102 has two serial channels implemented using the Z80-SIO LSI chip. The SIO allows for both asynchronous and synchronous (SDLC, HDLC, BISYNC, etc.) modes. Channel A can be used as both asynchronous and synchronous modes and channel B provides asynchronous operation. (Synchronous operation is available on the DSTD-102A version of the board - not the standard DSTD-102.) Channel A has additional RS-232C drivers and receivers for external clocks. In asynchronous mode both channels will operate up to 19.2k baud using the baud rate generator. The CTC may be used for higher rates. Channel A will run to 307 kilobaud in synchronous mode.

A strapping option allows the user to select the reset address to be either 0000H or E000H. The E000H option is required for use of standard software and hardware products including dy-4 SYSTEMS Debug Monitor (DDM) and Disk Control Monitor (DCM) firmware products. Also these products require onboard RAM strapped to reside at location FC00H to FFFFH.

The DSTD-102 is available in 2.5 MHz and 4 Mhz versions.

2.2 Block Diagram Description

Figure 2 - 1 is a block diagram illustrating the flow of system address, data and control signals on the DSTD-102. The following paragraphs describe the function of each of the major blocks.

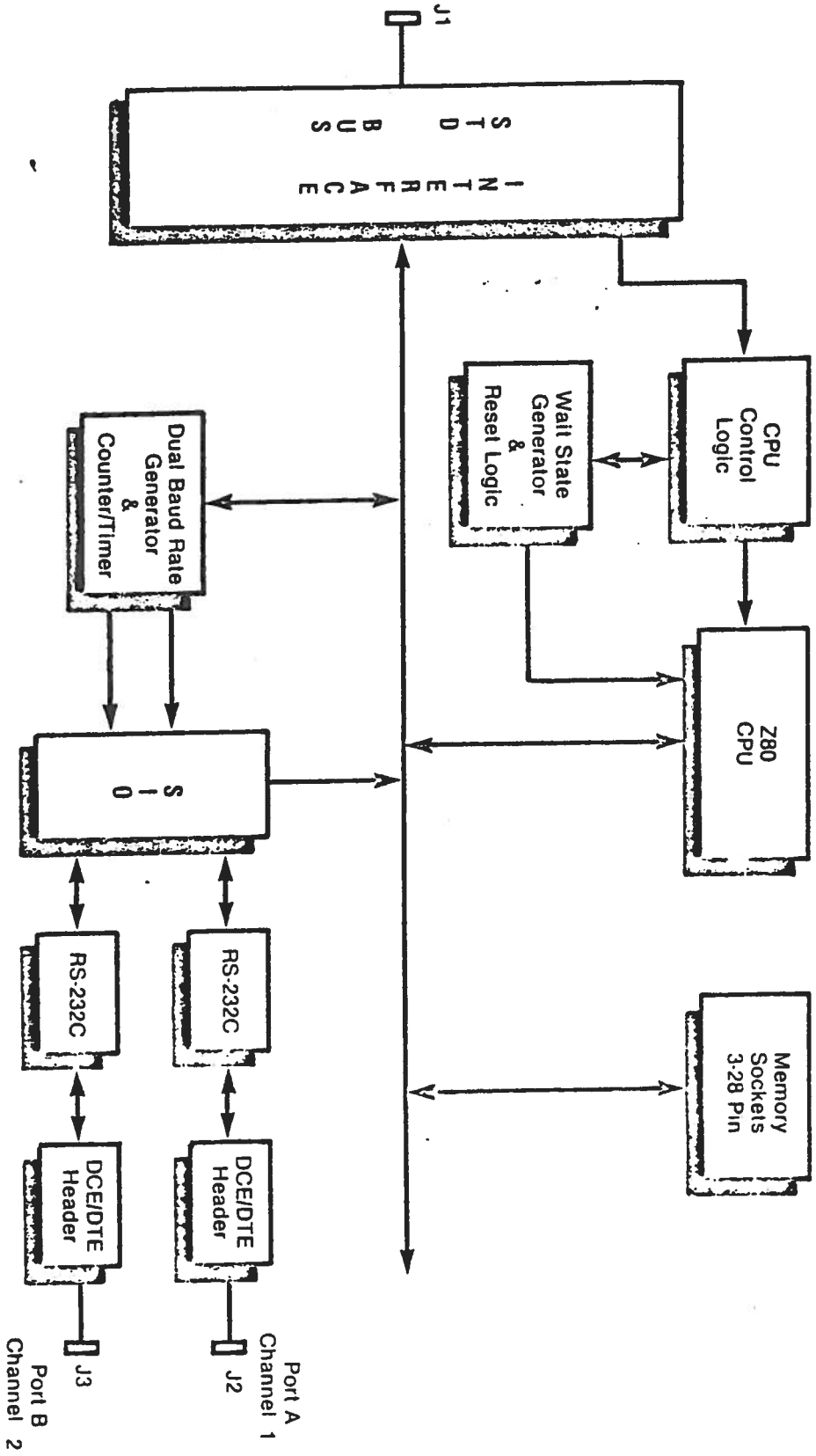


FIGURE 2 - 1 FUNCTIONAL BLOCK DIAGRAM

2.2.1 CPU

The Z80 is the system controller. It fetches, decodes and executes instructions from memory and generates the necessary address and control signals to co-ordinate data flow between the CPU and memory or between the CPU and system I/O devices.

2.2.2 Clock Generator

The DSTD-102 has a crystal controlled oscillator to generate the basic clock signals for the CPU and peripheral chips. A divide-by-two circuit ensures a 50% duty cycle and an active pullup circuit ensures proper clock levels. An inverted clock is supplied to the bus for use by other modules.

2.2.3 CTC (Counter/Circuit)

The Counter/Timer Circuit (MK3882/Z80-CTC) provides four independent, programmable channels for either software or hardware controlled counting and timing functions. Each channel can be configured by the CPU for various modes of operation and the built-in daisy chain priority interrupt logic provides for automatic, independent interrupt vectoring. The I/O port addresses for the CTC are hard-wired as follows:

I/O PORT ADDRESS	CTC CHANNEL
7C	0
7D	1
7E	2
7F	3

A strapping option has also been included to permit any or all of the four CTC channels to be cascaded for long count sequences.

Section 3 provides the necessary information for utilizing this option. For a complete description of the CTC operation, refer to either the Mostek MK3882 or Zilog Z80-CTC Technical Manual or Appendix A-15 of this manual.

2.2.4 Memory

The DSTD-102 has been designed to accommodate any combination of the byte-wide RAM, ROM and EPROM devices. Three 28-pin sockets have been provided, each of which may be strapped for any of the allowable memory types. These user-selectable options are fully described in Section 3.

2.2.5 Decode Logic

This section consists primarily of a PAL which decodes the high order six bits of memory address and generates the applicable chip select if on-board memory is to be selected. The PAL provides six separate memory configurations. The memory configurations are selected using an option jumper block as explained in Section 3.

The DSTD-102 has a latch to disable all on-board memory under software control. On power-up and reset, the latch is preset enabling the on board memory.

On-board memory is disabled by writing a '1' to I/O port 07BH. The on-board memory can be re-enabled by writing a '0' to port 07BH.

2.2.6 Reset Control Logic

This is a strapping option that causes a hardware-forced memory starting address upon system reset. A reset address of either 0000H or E000H may be selected.

This logic is required for use of standard MOSTEK hardware and software products including DDT-80, FLP-80DOS/MDX, MDX-SST, and MDX-DEBUG and dy-4 Debug Monitor (DDM) and CP/M software.

Also the push button reset function may be edge triggered or level sensitive depending on jumper block JB15. The edge triggered reset is synchronised with M1 to ensure that the contents of any dynamic RAM in the system are preserved during the reset process. The level sensitive option is required when it is necessary to hold the processor in a reset state indefinitely such as in a 'brown-out' situation.

2.2.7 Wait State Generator

This function, if enabled, causes memory read and write cycles to be lengthened by one clock period in order to allow sufficient access time when slower memory devices are used. Wait states can be enabled selectively, namely, - all memory cycles or opcode fetch cycles only or all memory cycles accessing on-board memory devices or all opcode fetch cycles and all memory cycles accessing onboard memory. An additional wait state may also be inserted during INTAK cycles.

2.2.8 Serial Ports

The DSTD-102 has two RS-232C serial ports implemented using the Z80-SIO/MK3884. Each port has a software programmable baud rate generator. The baud rate is set by writing to port 7AH. The least significant 4 bits set the baud rate for Channel A and the most significant 4 bits are for Channel B. Both Channels will operate from 50 baud to 19.2K baud. In additon the CTC may be used to generate the receive and transmit data clocks of Channel A allowing for non-standard baud rates. The CTC can only be used for asynchronous modes because it does not generate a 50% duty cycle clock. Channel A also has additional RS-232C drivers and receivers to enable it to handle external clocks for full synchronous operation (SDLC, HDLC, BYSYNC, MONOSYNC etc.).

SECTION 3

3.0 USER-SELECTABLE OPTIONS

3.1 Introduction

The DSTD-102 incorporates many strapping options to provide the user with a high degree of flexibility in system configurations. This section describes the use of the available jumper options.

3.2 Debug/Single Step Configurations

The DSTD-102 supports the MDX-SST module. This module generates a NMI (non-maskable interrupt) and asserts the DEBUG signal. This debug signal when enabled forces a logic '1' onto the most significant three bits of the address bus. Thus the interrupt service routine is located at E066H. If the debug is disabled the interrupt service routine is at the normal address (0066H). To enable the debug line, install a jumper between JB8-2 and JB8-3 and between JB13-4A to JB13-4B. Ensure that the memory option strap position the monitor software at E000H. dy-4 SYSTEMS' DDM firmware supports the single step facilities.

3.3 Memory Options

The PAL memory decoder shipped with DSTD-102 from the factory supports the options discussed in the following sections.

3.3.1 Restart Address

The DSTD-102 is capable of starting execution at either 0000H or E000H after reset. Reset address E000H is implemented in hardware. Since the program counter (internal to the Z80 microprocessor) always resets to 0000H, external hardware is required to force the most significant three bits of the data bus to all ones to get 0E000H. A multiplexer and a latch to control the multiplexer are used to perform this function. The first instruction at E000H should be 'JMP E003' to set the processors internal program counter to the correct memory location. The hardware latch forcing the address bit must then be cleared. This is done automatically by the first I/O cycle that the processor performs. If no I/O port access is normally made then a dummy I/O read of an unused port address must be done otherwise memory accessed will be constrained to addresses E000H through FFFFH. To ensure proper operation after reset, the following code sequence should be placed in memory at the E000H.

```
E000        C3 03 E0      JP E003H      ; jump instruction
                                        ; to update program
                                          counter

E003        DB nn         IN A,(nn)     ; read unused I/O
                                        ; port nn to clear
                                        ;resetaddress latch

E005                                    ; first instruction
                                        ; of user program
```

When using standard dy-4 SYSTEMS' or Mostek software (including DDM, DCM, DDT-80, FLP-80, DOS/MDX or MDX-DEBUG), the reset address must be E000H. The program counter and address latch modification instructions previously described are already contained with the DDM ROM. Ensure that pins 2 and 3 of JB8 are connected when the MDX-SST module is used.

3.3.2 Memory Configuration

The DSTD-102 incorporates three 28 pin sockets which can be independently configured to accept a variety of pin compatible memory devices. Table 3 - 1 lists each socket, its corresponding jumper block, and its address space for the standard configurations. The memory decoding is done using a PAL device. Table 3 - 2 shows for reference the signals brought to the jumper block for each of the different memory types. Table 3- 3 illustrates the necessary jumper connections for configuring a socket to accept each memory device.

Consult the factory for PAL programming details for non-standard requirements.

Option numbers are binary coded using JB14. JB14, 1A-1B has a weighting of '1'; JB14, 2A-2B has a weighting of '2' and JB14, 3A-3B has a weighting of '4'. For example, if option 5 (101) is desired JB14, 1A-1B, 3A-3B are left open and JB14, 2A-2B are inserted.

TABLE 3 - 1

MEMORY SOCKET/JUMPER BLOCK ASSIGNMENT

MEM TYPE	OPTION	MEM RANGE	U20(JB11)	U19(JB10)	U18(JB9)
2K	7(111)	E000-FFFF	E000-E7FF	E800-EFFF	F000-FFFF
2K	6(110)	0000-1FFF	0000-07FF	0800-0FFF	1000-1FFF
4K	5(101)	E000-FFFF	E000-EFFF	F000-FBFF	FC00-FFFF
4K	4(100)	0000-3FFF	0000-0FFF	1000-1FFF	2000-3FFF
8K	2(010)	0000-7FFF	0000-1FFF	2000-3FFF	4000-7FFF
16K	0(000)	0000-BFFF	0000-3FFF	4000-7FFF	8000-BFFF

NOTE: For 2K, 4K and 8K devices the memory range for U18 is twice as large as for U19 and U20. Hence when using the same memory device sizes for all three sockets, memory expansion off board will not be contiguous.

Option 5 is the memory configuration used for the boot proms and monitor in dy-4's STD Bus based microcomputer development systems.

TABLE 3 - 2

MEMORY DEVICE JUMPER STRAPS

TYPE	PART NO.	PINS 27	26	23	21	1
1Kx8 EPROM	2758	-	Vcc	Vpp	GND	-
1Kx8 EPROM	2759	-	Vcc	Vpp	Vcc	-
2Kx8 EPROM	2716	-	Vcc	Vpp	A10	-
4Kx8 EPROM	2732	-	Vcc	A11	A10	-
8Kx8 EPROM	2764	PGM	n/c	A11	A10	Vpp
16Kx8 EPROM	27128	PGM	A13	A11	A10	Vpp
1Kx8 RAM	4801	-	Vcc	/WE	GND	-
2Kx8 RAM	4802	-	Vcc	/WE	A10	-
8Kx8 RAM	2186	/WE	n/c	A11	A10	RDY
2Kx8 EEROM	X2816A	-	Vcc	/WE	A10	-

TABLE 3 - 3

MEMORY DEVICE JUMPER STRAPS

TYPE	PART NO.	JUMPER BLOCKS JB9 and JB12	JUMPER BLOCK JB10
1Kx8 EPROM	2758	B2-A6 ; A5-A6 B3-B4	A2-B4 ; B3-B4 C1-C2
1Kx8 EPROM	2759	B2-A6 ; A5-A6 B3-B2	A2-B4 ; B3-B4 C1-B3
2Kx8 EPROM	2716	B2-A6 ; A5-A6 B3-A3	A2-B4 ; B3-B4 B1-C1
4Kx8 EPROM	2732	B2-A6 ; A4-A5 B3-A3	A2-B4 ; B3-B2 B1-C1
8Kx8 EPROM	2764	B1-B5 ; A4-A5 B3-A3 ; A1-A6	A3-C3 ; B2-B3 B1-C1 ; A4-B4
16Kx8 EPROM	27128	B1-B5 ; A4-A5 B3-A3 ; A1-A6 B2-B6	A3-C3 ; B2-B3 B1-C1 ; A4-B4 C4-A2
1Kx8 RAM	4801	B2-A6 ; A5-B5 B3-A6	A2-B4 ; B3-C3 C1-C2
2Kx8 RAM	4802	B2-A6 ; A5-B5 B3-A3	A2-B4 ; B3-C3 B1-C1
8Kx8 RAM	2186	B1-B5 ; A4-A5 B3-A3 ; A1-A2	A3-C3 ; B2-B3 B1-C1 ; A1-A4
2Kx8 EEROM	X2816A	B2-A6 ; A5-B5 B3-A3	A2-B4 ; B3-C3 B1-C1

3.3.3 On-board Memory Disable Latch

All on-board memory can be enabled and disabled under software control. To use this feature jumper JB14 4A-4B is installed. This jumper allows the memory disable latch to be used. The latch is located at address 7BH. Writing a '0' to this latch enables on-board memory. Writing a '1' to the latch disables on-board memory. A power-up or RESET clears the latch thus enabling on-board memory.

3.4 WAIT State Generator

Three jumpers are provided to allow the use of slow memory devices. The first jumper generates a WAIT state on all memory cycles. The second jumper generates a WAIT state for M1 memory cycles only. Table 3 - 4 lists the access times of memory devices internal and external to the card for the two different memory cycle types for both the 2.5 MHz and 4.0 MHz DSTD 102 cards. The third jumper generates a WAIT state on internal memory accesses only. This means that slower EPROMS can be used on the DSTD-102 along with a high speed RAM card. A fourth jumper allows the generation of a WAIT state on interrupt acknowledge cycles. Table 3 - 5 gives the connections for the WAIT state options.

TABLE 3 - 4A

M1-MEMORY CYCLE WAIT STATES TIMING 2.5MHz

FUNCTION	JB11 Connections	INTERNAL M1	INTERNAL Other	EXTERNAL M1	EXTERNAL Other
No WAIT states	---	580	780	550	750
WAIT states on M1 cycles	1A to 1B	620	780	950	750
WAIT states on all memory cycles	3A to 3B	620	1180	950	1150

(in nanoseconds)

TABLE 3 - 4B

M1-MEMORY CYCLE WAIT STATES TIMING 4.0MHZ

FUNCTION	JB11 Connections	INTERNAL M1	INTERNAL Other	EXTERNAL M1	EXTERNAL Other
No WAIT states	---	330	455	300	425
WAIT states on M1 cycle	1A to 1B	590	455	560	425
WAIT states on all memory cycles	3A to 3B	590	705	560	675

(in nanoseconds)

TABLE 3 - 5 WAIT STATE OPTIONS

OPTION	JB11
No WAIT states	No Jumpers
All M1 cycles	1A to 1B
All Memory cycles	3A to 3B
Internal Memory cycles only	4A to 4B
Internal Memory cycles and external M1 cycles	4A to 4B 1A to 1B
Interrupt acknowledge cycle	2A to 2B

3.5 Counter/Timer Options

The four Counter/Timer channels may be cascaded for extended counting and timer functions. Appendix A-6 shows the jumper pin numbers for the CTC. Refer to the MK3882 Technical Manual or the Zilog Data Book for a complete description of the CTC operation.

Provision is made on the Counter/Timer option block to enable the NMI input of the processor to be connected to one of the outputs of the CTC. NMI is pin 5A of JB5.

In addition the CTC can be used as a baud rate generator for the serial channels to create non-standard baud rates.

Two commonly unused pins on the STD bus (MEMEX and IOEXP) may be connected through JB13 and buffers to the CTC. One pin is used as an input (IOEXP) and the other is used as an output (MEMEX).

3.6 Serial Channel Options

3.6.1 Baud Rate Generator

The DSTD-102 has a dual software-programmable baud rate generator. It is accessed through I/O port 7AH. This port is a write-only port. Bits 0 to 3 control channel A and bits 4 to 7 control channel B. Table 3 - 7 shows the programming information for the baud rate generator.

Table 3 - 6

Baud Rate Generator Programming

BAUD RATE	D3/D7	D2/D6	D1/D5	D0/D4	(HEX)
19,200	1	1	1	1	F
9,600	1	1	1	0	E
7,200	1	1	0	1	D
4,800	1	1	0	0	C
3,600	1	0	1	1	B
2,400	1	0	1	0	A
2,000	1	0	0	1	9
1,800	1	0	0	0	8
1,200	0	1	1	1	7
600	0	1	1	0	6
300	0	1	0	1	5
150	0	1	0	0	4
134.5	0	0	1	1	3
110	0	0	1	0	2
75	0	0	0	1	1
50	0	0	0	0	0

Thus to set port A to 9600 baud and port B to 1200 baud output a 7EH to I/O address 7AH.

3.6.2 DTE/DCE Configurations

3.6.2.1 DCE Configuration

When connecting to a CRT, printer or similar equipment the serial port is wired as Data Comunications Equipment. The signal names indicate control and data flow with respect to the CRT. Table 3 - 7 itemizes the jumper configurations for this mode of operation.

TABLE 3 - 7

RS-232C DCE Jumper Configuration

EIA(DCE) Signal Name	SIO Function	Installed Jumpers JB3,JB4	J2/J3 Pin Numbers
TX (2)	RX	2B to 1B	2
RX (3)	TX	1A to 2A	3
RTS (4)	CTS	4A to 5A	4
CTS (5)	RTS	5B to 4B	5
DTR (20)	DCD	7A to 6B	9
DCD (8)	DTR	6A to 7B	8
DSR (6)	+12V	8A to 8B	6 (JB3 only)

3.6.2.2 DTE Configuration

When connecting to a MODEM or similar equipment the serial port is wired as Data Terminal Equipment. The signal names indicate control and data flow with respect to the DSTD-102. Table 3 - 8 itemizes the jumper configuration for the mode of operation.

TABLE 3 - 8

RS-232C DTE Jumper Configuration

EIA(DTE) Signal Name	SIO Function	Installed Jumpers JB3,JB4	J2/J3 Pin Numbers
TX (2)	TX	1A to 1B	2
RX (3)	RX	2B to 2A	3
RTS (4)	RTS	3A to 3B	4
CTS (5)	CTS	4A to 4B	5
DTR (20)	DTR	6A to 6B	9
DCD (8)	DCD	7A to 7B	8
DSR (6)		------	6

3.6.3 Synchronous Operation

The DSTD-102A allows synchronous operation on Channel A. That is, additional RS-232C drivers and receivers are provided for interfacing external clocks. Two configurations are possible.

i) The DCE provides both transmit and receive timing information. When the DSTD-102A is the DCE, two RS-232C drivers are required. When the DSTD-102A is the DTE two RS-232C receivers are required.

ii) The DCE provides the transmit timing information and the DTE provides the receive timing information. The DSTD-102A provides the receive timing information. The DSTD-102 uses both the RS-232C driver and the RS-232C receiver.

Table 3 - 9 shows the jumpering required for each configuration. Note that the same drivers used for the external clocks are also used to drive the on-board TX and RX LED's. When these drivers are to be used for external clocking the LED's should be disconnected.

TABLE 3 - 9

DCE provides both clocks. DSTD-102A is the DCE

	JB2	JB3	J2	EIA
TX Clock	2A - 2B 1A - 2A	9A - 9B	10	15
RX Clock	3A - 4A 4A - 4B	11A - 11B	11	17

DCE provides both clocks. DSTD-102A is the DTE

	JB2	JB3	J2	EIA
TX Clock	1A - 1B	8A - 8B	10	15
RX Clock	3A - 3B	10A - 10B	11	17

DTE provides the transmit clock. DCE provides the receive clock. DSTD-102A is DCE.

	JB2	JB3	J2	EIA
TX Clock	1A - 2A 2A - 2B	9A - 9B	10	24
RX Clock	3A - 3B	10A - 10B	11	17

DTE provides the transmit clock, DCE provides the receive clock. DSTD-102A is DTE.

	JB2	JB3	J2	EIA
RX Clock	1A - 1B	8A - 8B	10	24
TX Clock	2A - 3A 4A - 4B	11A - 11B	11	17

Note that the clock names given above refer to data flow with respect to the DTE. EIA refers to the DB25 pin numbers assigned to these signals by the EIA RS-232C specifications.

Table 3-10 shows the cable connections to a standard RS-232C DB25S connector. Typically the cable is the same for both DCE and DTE systems with the configuration being determined by the on-based jumpers.

TABLE 3-10

SERIAL CABLE CONNECTIONS

J2/J3	RS232C/DB25S	EIA CIRCUIT
1	1	AA
2	2	BA
3	3	BB
4	4	CA
5	5	CB
6	6	CC
7	7	AB
8	8	CF
9	20	CD
10	15	DB
11	17	DD
12	19	--

NOTE: Pin 12 (DB25S pin 19) is included to accomodate some printers that use pin 19 for flow control.

SECTION 4

4.0 SPECIFICATIONS

4.1 Functional Specifications

4.1.1 Word Size

Instructions:	8, 16, 24, or 32 bits
Data:	8 bits

4.1.2 Cycle Time

Clock period (T state):	400 ns for DSTD-102-2.5 250 ns for DSTD-102-4.0
Instruction Cycle:	Min. 4 T states Max. 23 T states

4.1.3 Memory Capacity

Three 28 pin sockets are provided which may be populated with any mixture of the following devices:

2758 (1K x 8 EPROM)
2759 (1K x 8 EPROM)
2716 (2K x 8 EPROM)
2732 (4K x 8 EPROM)
2764 (8K x 8 EPROM)
27128(16Kx 8 EPROM)
MK 34000 (2K x 8 EPROM)
4118 (1K x 8 Static RAM)
4801 (1K x 8 Static RAM)
4802 (2K x 8 Static RAM)
2186 (8K X 8 Pseudo Static RAM)
X2816 (2K x 8 EEROM

4.1.4 Memory Access Time

The time required to access on-board memory by external DMA controllers is 100 ns plus the access time of the memory device. This is defined as the time interval between the time that the memory address is valid on the STD-BUS and the time that the output data is valid on the STD-BUS.

4.1.5 I/O Addressing

The on-board I/O addressing is hard wired to the following port addresses:

	PORT				ADDRESS
	BAUD RATE GENERATOR				7A
	ON-BOARD DISABLE LATCH				7B
8430	CTC	CH	0		7C
8430	CTC	CH	1		7D
8430	CTC	CH	2		7E
8430	CTC	CH	3		7F
8440	SIO	CH	A	DATA	BC
8440	SIO	CH	A	CONTROL	BD
8440	SIO	CH	B	DATA	BE
8440	SIO	CH	B	CONTROL	BF

4.1.6 I/O Capacity

The Z80 CPU utilizes the lower 8 bits of its address bus for I/O addressing to yield a total of 256 possible port addresses.

4.1.7 Interrupts

The CPU may be programmed to process interrupts in any of three different modes (mode 0, 1, or 2 as described in any Z80 Technical Manual). Mode 2 operation (vectored interrupts) is by far the most powerful and is compatible with dy-4 DSTD and MOSTEK MDX Series cards.

Multi-level interrupt processing is also possible with the Z80 CPU. The level of stacking is limited only by available memory space.

The DSTD-102 will also accept non-maskable interrupts which force a restart at location 0066H.

4.1.8 System Clock

DSTD-102-2.5	2.5MHz ±0.05%
DSTD-102-4.0	4.0MHz ±0.05%

4.2 Electrical Specification

4.2.1 STD Bus Interface

Bus Inputs: One 74LS load max.

Bus Outputs: I_{OL} = 24 mA min. @ V_{OL} = 0.5 Volts
I_{OH} = 15 mA min. @ V_{OH} = 2.4 Volts

4.2.2 Serial Ports

Inputs: One 74LS load max.

Outputs: +/- 12V Current Limited to 10mA

4.2.3 Operating Temperature

0 Degrees C to 50 Degrees C
95% humidity non-condensing

4.2.4 Power Supply Requirements

+5V +/- 5% @ 1.2A
+12V +/- 5% @ 0.1A
-12V +/- 5% @ 0.1A
(excluding memory power requirements)

4.3 Mechanical Specifications

4.3.1 Card Dimensions

4.50 in. (11.43 cm.) wide by 6.50 in. (16.51 cm) long

0.48 in. (1.22 cm.) maximum height

0.062 in. (0.16 cm.) printed circuit board thickness

4.3.2 STD Bus Edge Connector

56 pin Dual Readout; 0.125 in. centers

Mating Connector

Viking 3VH28/1CE5 (printed circuit)
Viking 3VH28/1CND5 (wire wrap)
Viking 3VH28/1CN5 (solder lug)

4.3.3 Serial Port Connector

12 Pin Dual Readout; 0.100 inch grid

Mating Connector

Amp 87631-8 (housing)
Amp 86016-2 (contact)
or equivalent

SECTION 5

5.0 FACTORY NOTICES

5.1 Factory Repair Service

In the event that difficulty is encountered with this unit, it may be returned directly to dy-4 for repair. This service will be provided free of charge if the unit is returned within the warranty period. However, units which have been modified or abused in any way will not be accepted for service, or will be repaired at the owner's expense.

When returning a circuit board, place it inside the conductive plastic bag in which it was delivered to protect the MOS devices from electrostatic discharge. **THE CIRCUIT BOARD MUST NEVER BE PLACED IN CONTACT WITH STYROFOAM MATERIAL.** Enclose a letter containing the following information with the returned circuit board:

Name, address and phone number of purchaser
Date and place of purchase
Brief description of the difficulty

Mail a copy of this letter SEPARATELY to:

Service Department
dy-4 SYSTEMS INC.,
888 Lady Ellen Place,
Ottawa, Ontario
K1Z 5M1, Canada

or

Service Department
dy-4 SYSTEMS INC.,
3582 Dubarry Rd.
Indianapolis, IN 46226

Securely package and mail the circuit board, prepaid and insured, to the same address.

5.2 Limited Warranty

dy-4 warrants this product against defective materials and workmanship for a period of 90 days. This warranty does not apply to any product that has been subjected to misuse, accident, improper installation, improper application, or improper operation, nor does it apply to any product that has been repaired or altered by other than an authorized factory representative.

There are no warranties which extend beyond those herein specifically given.

NOTICE

The antistatic bag is provided for shipment of the dy-4 PC boards to prevent damage to the components due to electrostatic discharge. Failure to use this bag in shipment will **VOID** the warranty.

APPENDIX A

OPTION PROGRAMMING SUMMARY

APPENDIX A

OPTION PROGRAMMING SUMMARY

- 1 OPTIONAL JUMPER BLOCKS

The following is a list of the option Jumper Blocks on the STD-102 card.

JB1	LED connection
JB2	Serial Channel A Clock TTL Side
JB3	Serial Channel A DTE/DCE Configuration Block
JB4	Serial Channel B DTE/DCE Configuration Block
JB5	Counter Timer Jumper Block
JB6	LED Transmit
JB7	LED Receive
JB8	Restart Address Jumper Block
JB9	Memory Socket Configuration Block for U18
JB10	Memory Socket Configuration Block for U19
JB11	WAIT State Generator options
JB12	Memory Socket Configuration Block of U20
JB13	CTC/Bus Interface Jumper Block
JB14	On-board Memory Options
JB15	Reset Mode

A - 2 LED Connections (JB1)

These jumpers are installed to drive the LED's. Note the jumpers should not be installed if Serial Channel A is used in synchronous mode and is supplying the clocks to external equipment.

		A B	
Driver 1	1	o-----o	LED 1
Driver 2	2	o-----o	LED 2

---- Indicates Factory Default

A - 3 Serial Channel A Clock Jumpers TTL Side (JB2)

This jumper block allows the selection of the Transmit and receive clocks for Channel A.

		A	B	
Transmit Clock (input)	1	o	o	RS232 Clock Receiver 1
		|		
Internal Baud Rate Generator	2	o	o	RS232 Clock Transmitter 1
Receiver Clock (input)	3	o	o	RS232 Clock Receiver 2
Internal Baud Rate Generator	4	o	o	RS232 Clock Transmitter 2
Receiver Clock (input)	5	o	o	CTC output
		|		
Transmit Clock (input)	6	o	o	CTC output

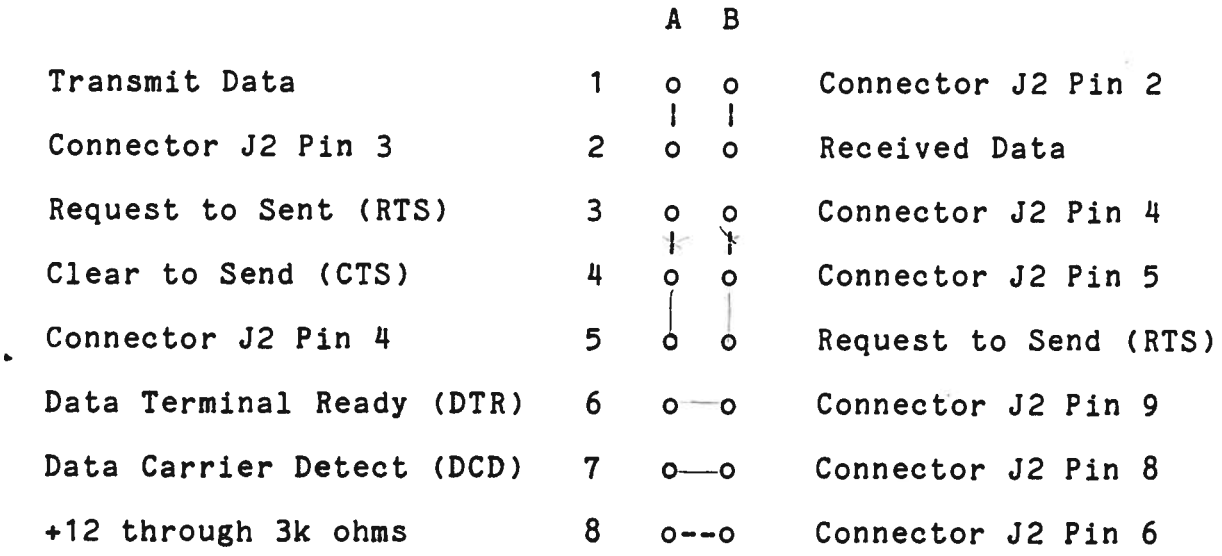
A - 4 Channel A DTE/DCE Configuration Block (JB3)

These jumpers allow the board to be configured as Data Terminal Equipment or Data Communications Equipment when used with a standard dy-4 SYSTEMS Cable. The signals given are those of the SIO device which is labelled as Data Terminal Equipment.

		A	B	
Transmit Data	1	o	o	Connector J2 Pin 2
		!	!	
Connector J2 Pin 3	2	o	o	Received Data
Request to Sent (RTS)	3	o	o	Connector J2 Pin 4
		!		
Clear to Send (CTS)	4	o	o	Connector J2 Pin 5
Connector J2 Pin 4	5	o	o	Request to Send (RTS)
Data Terminal Ready (DTR)	6	o	o	Connector J2 Pin 9
Data Carrier Detect (DCD)	7	o	o	Connector J2 Pin 8
RS-232C Receiver 1	8	o	o	Connector J2 Pin 10
RS-232C Transmit 1	9	o	o	Connector J2 Pin 10
RS-232C Receiver 2	10	o	o	Connector J2 Pin 11
RS-232C Transmit 2	11	o	o	Connector J2 Pin 11
+12 through 3k ohms	12	o--o		Connector J2 Pin 6

A - 5 Channel B DTE/DCE Configuration Block (JB4)

This jumper block allows the channel to be configured as Data Terminal Equipment or Data Communications Equipment.



A - 6 Counter Timer Jumper Block (JB5)

This jumper block allows the counter/timer channels to be cascaded for longer sequences. It also provides access to the auxiliary input and output buffers which are connected through JB12 to MEMEX and IOEXP bus signals. The SIO clock is used when the CRT is used as a baud rate generator.

		A	B	
Auxiliary Input	1	o	o	SIO Clock
CTC Channel 0 input	2	o	o	CTC Channel 0 zero detect
CTC Channel 1 Zero detect	3	o	o	CTC Channel 1 input
CTC Channel 2 input	4	o	o	CTC Channel 2 zero detect
Internal Non-Maskable Interrupt	5	o	o	Auxiliary Output
CTC Channel 3 input	6	o	o	N/C

A - 7 JB6/7 LED Blocks

These jumpers are installed to drive the LEDs. They should be removed when Channel A is operated in Synchronous mode.

	A B	
TX Driver	o--o	LED Driver U7 (JB6)
RX Driver	o--o	LED Driver U6 (JB7)

A - 8 Restart Address Jumper Block JB8

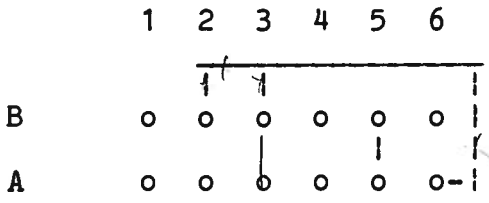
Installing the jumper between pins 2 and 3 forces the restart address to E000H. Installing the jumper between pins 1 and 2 forces a restart address to 0000H.

JB8

1	o	Force 0000H
2	o	Restart address control
3	o	Force E000H

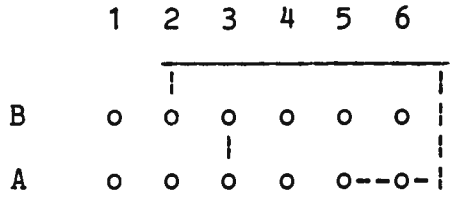
A - 9 Memory Socket Configuration Blocks JB9, JB12

JB9



for socket U18
(1K RAM)

JB12

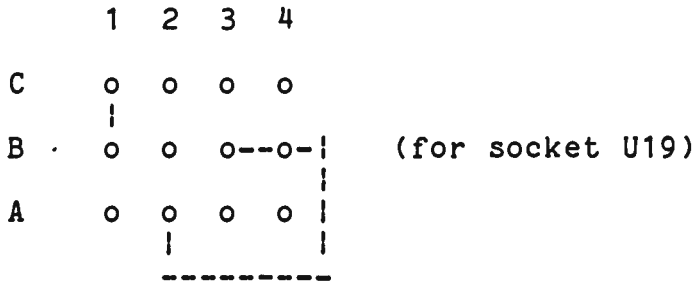


for socket U20
(2K EPROM)

A1	Socket Pin 1	(pstatic RAM ready/Vpp)
B1	Socket Pin 27	(Pseudo static RAM /WE)
A2	Processor wait logic	
B2	Socket Pin 26	(Vcc/A13)

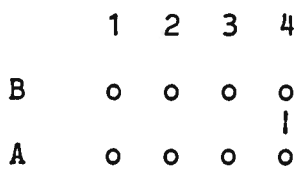
A3	Processor Address Bit A10
B3	Socket Pin 21 (A10/L)
A4	Processor Address Pin A11
B4	Ground
A5	Socket Pin 23 (A11/WE/Vpp)
B5	Processor Write Strobe
A6	+5V
B6	Processor Address Bit 13

A - 10 Memory Socket Configuration Block JB10



A1	Processor wait logic
A2	Socket Pin 26 (Vcc/A13)
A3	Socket Pin 27 (Pseudo static RAM /WE)
A4	Socket Pin 1 (pstatic RAM ready/Vpp)
B1	Processor Address Bit A10
B2	Processor Address Pin A11
B3	Socket Pin 23 (A11/WE/Vpp)
B4	+5V
C1	Socket Pin 21 (A10/L)
C2	Ground
C3	Processor Write Strobe
C4	Processor Address Bit 13

A - 11 Wait State Generator Configuration Block JB11



A1,B1	Wait on M1 cycles
A2,B2	Wait on Interrupt Acknowledge cycles
A3,B3	Wait on MREQ cycles
A4,B4	Wait on On-board Memory Cycles

A - 12 CTC/BUS Interface (JB13)

These jumpers are installed to allow counter/timer I/O to be accessed using two lines of the backplane that are not normally used by the Z80 STD bus cards. These signals use the BUS lines normally referred to as MEMEX and IOEXP. This jumper block also contains the Debug function enable jumper.

	A	B	
MEMEX (J1-36)	o	o	CTC Output
Ground	o	o	Ground
CTC Input	o	o	IOEXP (J1-35)
DEBUG (J1-38)	o--o		debug f/f

A - 13 On Board Memory Options (JB14)

Memory option weight '1', '2', and '4' selects the memory configuration for the DSTD-102. This jumper block is used in a binary coded fashion. See section 3.3.2 for details.

To use the on-board memory disable feature jumper 4A - 4B has to be installed. Port 7B can then be used to control the memory.

			A B	
Memory Option 1	Weight '1'	1	o—o	Ground
Memory Option 2	Weight '2'	2	o o	Ground
Memory Option 3	Weight '4'	3	o o	Ground
DSMEN Latch Input		4	o-o	DSMEN Option Output

A - 14 Reset Mode (JB15)

Jumper JB15 is used to select the push button reset mode. The push button logic is edge sensitive if the jumper is omitted and is level sensitive if it is installed.

	JB15	
Push Button Input	o o	Reset Logic

A- 15 Programming The CTC

1) Channel Selection

DSTD products using the Z80 CTC decode the CTC to occupy 4 contiguous port addresses. Writing to the appropriate port address will automatically select the correct register in the CTC.

2) Interrupt Vectors

If any one of the CTC channels is going to be used with its interrupt enabled, an Interrupt Vector must be written to the CTC. The user need only supply the 5 high bits of one vector as the CTC assumes the vector points to 4 contiguous byte pairs corresponding to the 4 channels. Note that D0 must equal 0 to indicte that the word being written to the CTC is an interrupt vector; this also requires vectored addresses to start at an even memory location.

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	X	X	0
	<USER SUPPLIED VECTOR>				<SUPPLIED BY CTC>		

3) Channel Control Register

The control register bit functions are as illustrated below.

D7	D6	D5	D4	D3	D2	D1	D0
INT ENA	MODE	RANGE	SLOPE	TRIG	LOAD TC	RESET	1

D0 = 0 indicates the byte is an INTERRUPT VECTOR.
D0 = 1 indicates the byte is a CONTROL WORD.

D1 = 0 the channel continues current operation.
D1 = 1 the channel is immediately RESET to control word values.

D2 = 0 indicates NO TIME CONSTANT to follow.
D2 = 1 the next I/O byte will be a TIME CONSTANT. (1 to 256)

D3 = 0 timer will FREE-RUN starting on next processor cycle.
D3 = 1 indicates timer will start on EXTERNAL TRIGGER.

D4 = 0 indicates external trigger on NEGATIVE-GOING edge.
D4 = 1 indicates external trigger on POSITIVE-GOING edge.

D5 = 0 indicates prescaler factor of 16. (timer mode only)
D5 = 1 indicates prescaler factor of 256. (timer mode only)

D6 = 0 indicates TIMER mode. (prescaler is enabled)
D6 = 1 inidcates COUNTER mode. (prescaler disabled)

D7 = 0 INTERRUPT DISABLED for that channel.
D7 = 1 INTERRUPT on zero count ENABLED for the channel.

APPENDIX B

STD-Z80 BUS PIN OUT

APPENDIX B

STD-Z80 BUS PIN OUT AND DESCRIPTION

BUS	MNEMONIC	DESCRIPTION
1	5V	5Vdc system power
2	5V	5Vdc system power
3	GND	Ground - System signal ground and DC return
4	GND	Ground - System signal ground and DC return
5	-5V	-5Vdc system power
6	-5V	-5Vdc system power
7	D3	
8	D7	
9	D2	Data Bus (Tri-state, input/output active high). D0-D7 constitute an 8-bit bidirectional data bus. The data bus is used for data exchange with memory and I/O devices
10	D6	
11	D1	
12	D5	
13	D0	
14	D4	
15	A7	
16	A15	
17	A6	Address Bus (Tri-state, output, active high).
18	A14	

19	A5	A0-A15 make up a 16-bit address bus The address bus provides the address for memory (up to 65k bytes) data exchanges and for I/O device data exchanges. I/O addressing uses the lower 8 address bits to allow the user to directly select up to 256 input or 256 output ports. A0 is the least signficant address bit. During refresh time, the lower 7 bits contain a valid refresh address for dynamic memories in the system.
20	A13	
21	A4	
22	A12	
23	A3	
24	A11	
25	A2	
26	A10	
27	A1	
28	A9	
29	A0	
30	A8	
31	/WR	Memory Write (Tri-state, output, active low). /WR indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.
32	/RD	Memory Read (Tri-state, output, active low). /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.
33	/IORQ	Input/Output Request (Tri-state, output, active low). The /IORQ signal indicates that the lower half of the address bus holds a valid I/O address for an I/O read or write operation. An /IORQ signal is also generated with an /M1 signal when an interrupt is being acknowledged to indicate than an interrupt response vector can be placed on the data bus. Interrupt Acknowledge operations occur during /M1 time, while I/O operations never occur during /M1 time.
34	/MEMRQ	Memory Request (Tri-State output, active low). The /MEMRQ signal indicates that the address bus holds a valid address for a memory read or write operation.
35	/IOEXP	I/O expansion, not used on dy-4 Systems DSTD.

36	/MEMEX	Memory expansion, not used on dy-4 Systems DSTD cards.
37	/REFRESH	/REFRESH (Tri-state, output, active low). /REFRESH indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and the /MEMRQ signal should be used to perform a refresh cycle for all dynamic RAMs in the system. During the refresh cycle A7 is a logic zero and the upper 8 bits of the address bus contains the I register.
38	/DEBUG	/DEBUG (Input) used in conjunction with DDT-80 operating system and the MDX Single Step card for implementing a hardware single step. When pulled low, the /DEBUG line will set a latch that will force the upper three address lines to a logic 1. To reset this latch, an I/O operation must be performed.
39	/M1	Machine Cycle One (Tri-state, output, active low). /M1 indicates that the current machine cycle is in the opcode fetch cycle of an instruction. Note that during the execution of a 2-byte opcodes, /M1 will be generated as each opcode is fetched. These two-byte op-codes always begin with a CBH, DDH, EDH or FDH. /M1 also occurs with /IORQ to indicate an interrupt acknowledge cycle.
40	STATUS 0	DMA priority chain input.
41	/BUSAK	Bus Acknowledge (Output, active low). Bus Acknowledge is used to indicate to the requesting device that the CPU address bus, data bus, and control bus signals have been set to their high impedance state and the external device can now control the bus.

42	/BUSRQ	Bus Request (Input, active low). The /BUSRQ signal is used to request the CPU address bus, data bus, and control signal bus to go to a high impedance state so that other devices can control those buses. When /BUSRQ is activated, the CPU will set these buses to a high impedance state as soon as the Current CPU machine cycle is terminated, and the Bus Acknowledge (/BUSAK) signal is activated.
43	/INTAK	Interrupt Acknowledge (Tri-state output, active low). The /INTAK signal indicates that an interrupt acknowledge cycle is in progress, and the interrupting device should place its response vector on the data bus.
44	/INTRQ	Interrupt Request (Input, active low). The Interrupt Request Signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal software controlled interrupt enable flip-flop (IFF) is enabled and if the /BUSRQ signal is not active. When the CPU accepts the interrupt, an acknowledge signal (/IORQ during an /M1) is sent out at the beginning of the next instruction cycle.
45	/WAITRQ	WAIT REQUEST (Input, active low). Wait request indicates to the CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter wait states for as long as this signal is active. The signal allows memory or I/O devices of any speed to be synchronized to the CPU.

46	/NMIRQ	Non-Maskable Interrupt request (Input, negative edge triggered). The Non-Maskable Interrupt request has a high priority than /INTRQ and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. /NMIRQ automatically forces the CPU to restart to location 0066H. The program counter is automatically saved in the external stack so that the user can return to the program that was interrupted. Note that continuous WAIT cycle can prevent the current instruction from ending, and that a /BUSRQ will over-ride a /NMIRQ.
47	/SYSRESET	System Reset (Output, active low). The System Reset line indicates that a reset has been generated from either an external reset or the power-on reset circuit. The system reset will occur only once per reset request and will be approximately 2 microseconds in duration. The system reset will also force the CPU program counter to zero, disable interrupts, set the I register to 00H, set the R register to 00H and set Interrupt Mode 0.
48	/PBRESET	Pushbutton Reset (Input, active low). The Pushbutton reset will generate a debounced system reset.
49	/CLOCK	Processor Clock (Output, active low). Single phase system clock.
50	CNTRL	Auxiliary Timing
51	PCO	Priority Chain Output (Output, 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.

52	PCI	Priority Chain 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.
53	AUX GND	Auxiliary Ground (Bussed)
54	AUX GND	Auxiliary Ground (Bussed)
55	+12V	+12Vdc system power
56	-12V	-12Vdc system power

NOTES:

1. The reference to input and output of a given signal is made with respect to the CPU module.

APPENDIX C

DSTD-102 PARTS LIST

DSTD 102 PARTS LIST

DY4PART	QTY	DESCRIPTION	DESIGNATION
PT012008	1	74LS08 TTL-LS	U15
PT012014	1	74LS14 TTL-LS	U5
PT012020	1	74LS20 TTL-LS	U10
PT012074	2	74LS74 TTL-LS	U3,U14
PT012112	1	74LS112 TTL-LS	U11
PT012164	1	74LS164 TTL-LS	U4
PT012243	1	74LS243 TTL-LS	U27
PT012244	2	74LS244 TTL-LS	U28,U29
PT012245	3	74LS245 TTL-LS	U22,U24,U25
PT012257	1	74LS257 TTL-LS	U13
PT013074	1	74S74 TTL-S	U12
PT015009	1	MK3880n (Z80-CPU) 2.5 MHZ CPU	U17
PT015013	1	MK3882n (Z80-CTC) 2.5 MHZ CTC	U8
PT015017	1	MK3884n (Z80-SIO/0) 2.5 MHZ SIO/0	U16
PT016001	2	75188 OR MC1488 INTERFACE	U6,U7
PT016002	2	75189 OR MC1489 INTERFACE	U1,U2
PT036001	1	PAL12L6	U23
PT036002	2	PAL16L8	U9,U26
PT041101	1	1/4 WATT, 100 OHM, 5% RESISTOR	R12
PT041122	1	1/4 WATT, 1.2K OHM, 5% RESISTOR	R4
PT041220	1	1/4 WATT, 22 OHM, 5% RESISTOR	R6
PT041221	1	1/4 WATT, 220 OHM, 5% RESISTOR	R5
PT041302	2	1/4 WATT, 3K OHM, 5% RESISTOR	R1,R7
PT041472	2	1/4 WATT, 4.7K OHM, 5% RESISTOR	R2,R3
PT041473	1	1/4 WATT, 47K OHM, 5% RESISTOR	R10
PT041681	1	1/4 WATT, 680 OHM, 5% RESISTOR	R13
PT043012	2	8 PIN, 7 RESISTOR, 4.7K OHM, SIP RESISTOR NETWORK	RN2,RN3
PT043017	1	10 PIN, 9 RESISTOR, 4.7K OHM, SIP RESISTOR NETWORK	RN4
PT051004	1	034-55101 OR 035-56101, 100uf, RADIAL ELECTROLYTIC CAPACITOR	C35
PT052003	1	CK05BX330K, 33pf, 200V CERAMIC CAPACITOR	C2
PT052004	8	CK05BX331K,330pf, 200V CERAMIC CAPACITOR	C6-13
PT052009	1	8131-100-Z5U-474M, .47uf, 50V, CERAMIC CAPACITOR	C3
PT052010	18	.1uf, 50V(.1 LD. SP.) 8121-050-Z5U-104M, CERAMIC CAPACITOR	C16,17,19,21-34,36
PT052013	1	.1uf,50V (.2 LD. SP.) 8121-050-Z5U-104M) (102 BOARD ONLY)	C14
PT053000	1	TAG10M25, 10uf, 25V TANTALUM CAPACITOR	C18
PT061003	1	2N3906 TRANSISTOR	Q1
PT071000	1	1N4148 SIGNAL DIODE	D3
PT073001	2	1N4001 RECTIFIER	D1,D2
PT091000	1	HLMP6300 SMALL RED LED	LED2
PT091002	1	HLMP6500 SMALL GREEN LED	LED1
PT101000	1	K1135A CRYSTAL OSCILLATOR GENERATOR	U21
PT101005	1	K1116A 5.000 MHZ CRYSTAL OSCILLATOR	U30
PT111873	1	S208-1 CARD EJECTOR WITH PINS	
PT122003	3	CHD6960WIS 60 PIN DOUBLE ROW HEADER	JB1-JB5,JB9-JB15
PT122004	1	CHS6936WIS 36 PIN SINGLE ROW HEADER	JB6,JB7-JB10c,JB15
PT123003	2	87516-2 12 PIN RIGHT ANGLE CONNECTOR (AMP ONLY)	J2,J3
PT126020	3	640464-3 20 PIN I.C. SOCKET	U9,U23,U26
PT126028	4	640362-3 28 PIN I.C. SOCKET	U18-U20,U8
PT126040	2	640379-3 40 PIN I.C. SOCKET	U16,U17
PT344901	1	DSTD 102 DY00449-H-A1-6	
PT711003	1	102 MANUAL	

APPENDIX C

DSTD 102 PARTS LIST

DY4PART	QTY	DESCRIPTION	DESIGNATION
PT012008	1	74LS08 TTL-LS	U15
PT012014	1	74LS14 TTL-LS	U5
PT012020	1	74LS20 TTL-LS	U10
PT012074	2	74LS74 TTL-LS	U3,U14
PT012112	1	74LS112 TTL-LS	U11
PT012164	1	74LS164 TTL-LS	U4
PT012243	1	74LS243 TTL-LS	U27
PT012244	2	74LS244 TTL-LS	U28,U29
PT012245	3	74LS245 TTL-LS	U22,U24,U25
PT012257	1	74LS257 TTL-LS	U13
PT013074	1	74S74 TTL-S	U12
PT015010	1	MK3880n-4 (Z80A-CPU) 4.00 MHZ CPU	U17
PT015014	1	MK3882n-4 (Z80A-CTC) 4.00 MHZ CTC	U8
PT015020	1	MK3884n-84 (Z80A DART) 4.00 MHZ DART	U16
PT016001	2	75188 OR MC1488 INTERFACE	U6,U7
PT016002	2	75189 OR MC1489 INTERFACE	U1,U2
PT036001	1	PAL12L6	U23
PT036002	2	PAL16L8	U9,U26
PT041101	1	1/4 WATT, 100 OHM, 5% RESISTOR	R12
PT041122	1	1/4 WATT, 1.2K OHM, 5% RESISTOR	R4
PT041220	1	1/4 WATT, 22 OHM, 5% RESISTOR	R6
PT041221	1	1/4 WATT, 220 OHM, 5% RESISTOR	R5
PT041302	2	1/4 WATT, 3K OHM, 5% RESISTOR	R1,R7
PT041472	2	1/4 WATT, 4.7K OHM, 5% RESISTOR	R2,R3
PT041473	1	1/4 WATT, 47K OHM, 5% RESISTOR	R10
PT041681	1	1/4 WATT, 680 OHM, 5% RESISTOR	R13
PT043012	2	8 PIN, 7 RESISTOR, 4.7K OHM, SIP RESISTOR NETWORK	RN2,RN3
PT043017	1	10 PIN, 9 RESISTOR, 4.7K OHM, SIP RESISTOR NETWORK	RN4
PT051004	1	034-55101 OR 035-56101, 100uf, RADIAL ELECTROLYTIC CAPACITOR	C35
PT052003	1	CK05BX330K, 33pf, 200V CERAMIC CAPACITOR	C2
PT052004	8	CK05BX331K,330pf, 200V CERAMIC CAPACITOR	C6-13
PT052009	1	8131-100-Z5U-474M, .47uf, 50V, CERAMIC CAPACITOR	C3
PT052010	18	.1uf, 50V(.1 LD. SP.) 8121-050-Z5U-104M, CERAMIC CAPACITOR	C16,17,19,21-34,36
PT052013	1	.1uf,50V (.2 LD. SP.) 8121-050-Z5U-104M) (102 BOARD ONLY)	C14
PT053000	1	TAG10M25, 10uf, 25V TANTALUM CAPACITOR	C18
PT061003	1	2N3906 TRANSISTOR	Q1
PT071000	1	1N4148 SIGNAL DIODE	D3
PT073001	2	1N4001 RECTIFIER	D1,D2
PT091000	1	HLMP6300 SMALL RED LED	LED2
PT091001	1	HLMP6400 SMALL YELLOW LED	LED1
PT101000	1	K1135A BAUD RATE GENERATOR	U21
PT101007	1	K1116A 8.000 MHZ CRYSTAL OSCILLATOR	U30
PT111073	1	S208-1 CARD EJECTOR WITH PINS	
PT122003	2	CHD6960WIS 60 PIN DOUBLE ROW HEADER	JB1-JB5,JB9-JB15
PT122004	1	CHS6936WIS 36 PIN SINGLE ROW HEADER	JB6,JB7-JB10c,JB15
PT123003	2	87516-2 12 PIN RIGHT ANGLE CONNECTOR (AMP ONLY)	J2,J3
PT126020	3	640464-3 20 PIN I.C. SOCKET	U9,U23,U26
PT126028	4	640362-3 28 PIN I.C. SOCKET	U18-U20,U8
PT126040	2	640379-3 40 PIN I.C. SOCKET	U16,U17
PT344901	1	DSTD 102 DY00449-H-A1-7	
PT711003	1	102 MANUAL	

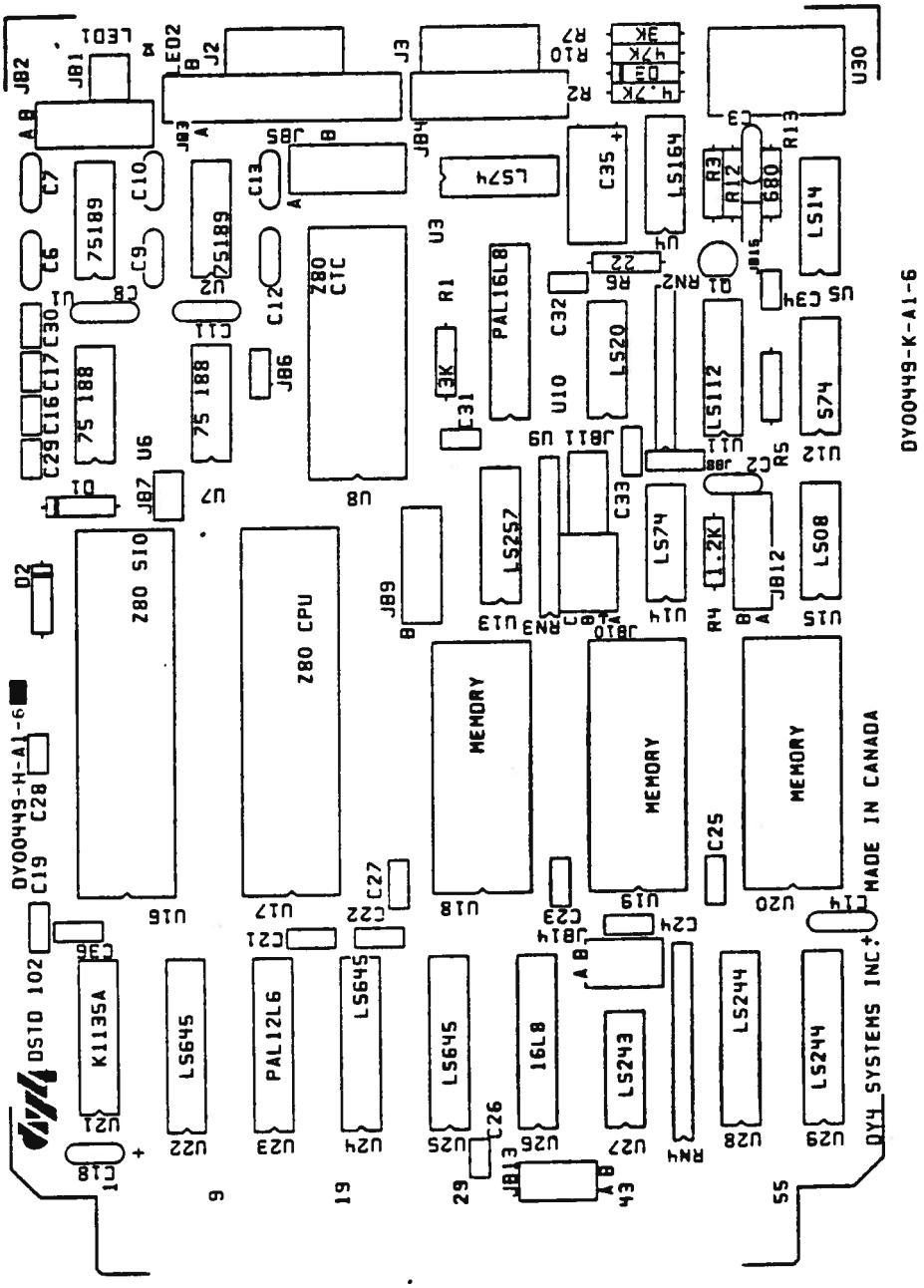


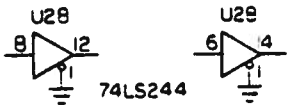
FIGURE C-1 DSTD-102-4 SILK SCREEN

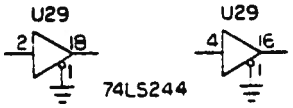
APPENDIX D

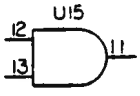
SCHEMATIC

IC POWER PINS					
TYPE	+12	-12	+5	-5	GND
74LS08			14		7
74LS14			14		7
74LS20			14		7
74LS74			14		7
74S74			14		7
74LS112			16		8
74LS164			14		7
74LS243			14		7
74LS244			20		10
74LS257			16		8
74LS645			20		10
2716/2764			28		14
75188	14	1			7
75189			14		7
KI135A	9		2		11
PAL12L6			20		10
Z80 CTC			24		5
Z80 SIO			9		31
Z80 CPU			11		29
XTAL LOCOII			14		7

UNUSED GATES

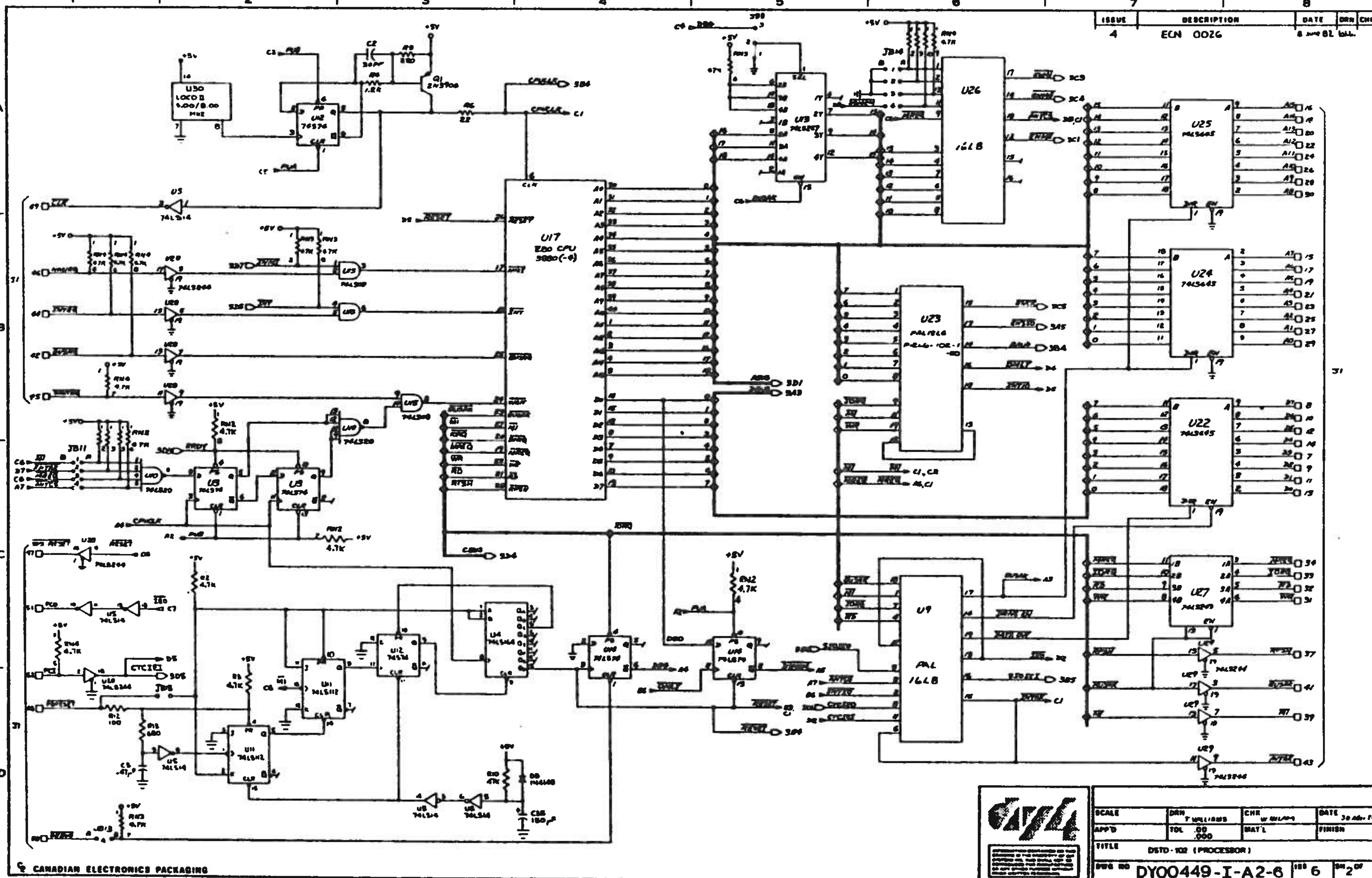




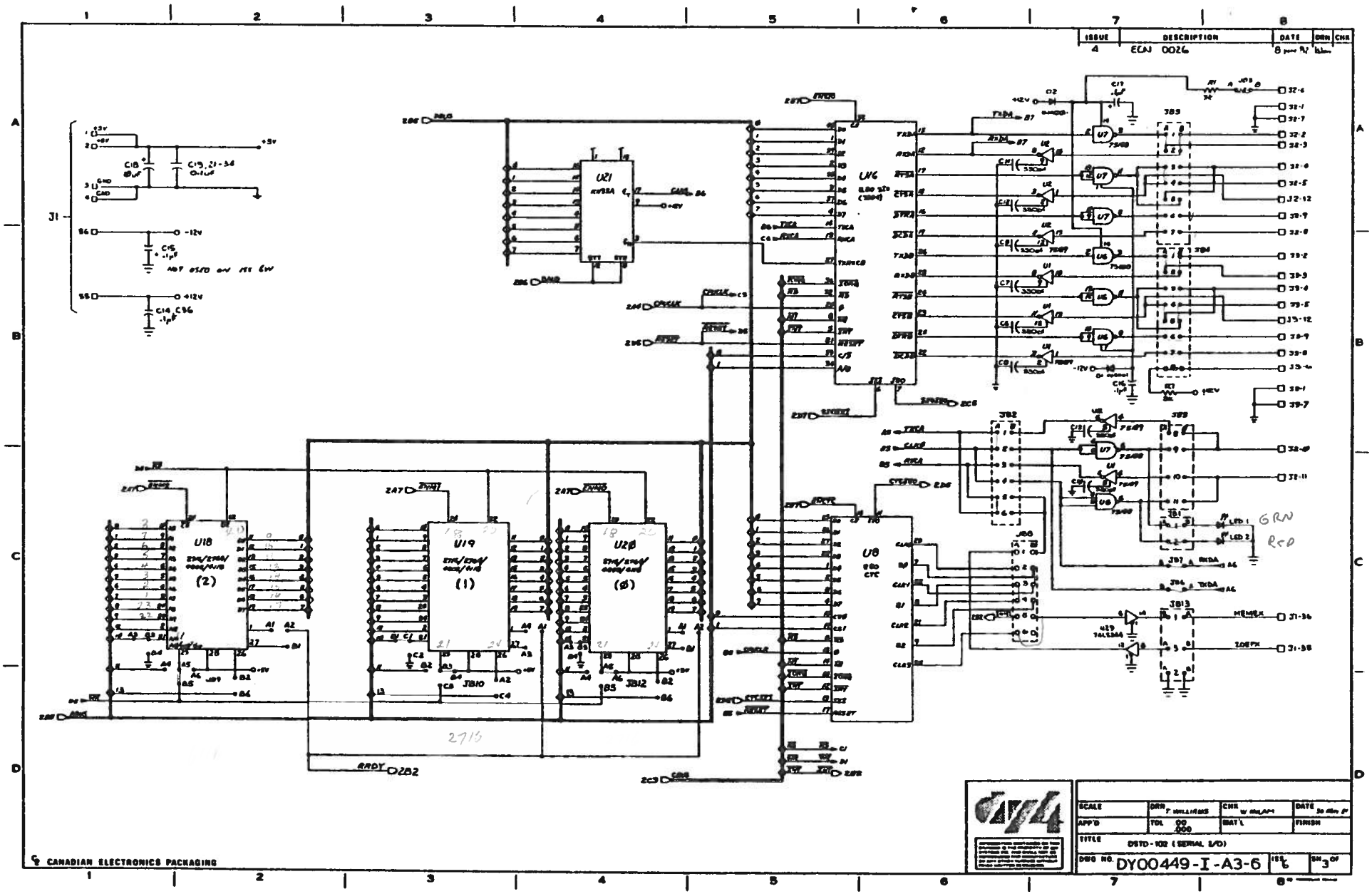


74LS08

DSTD 102			
SCALE	DRN D. NICHOLLS	CHK	DATE 4/11/82
APP'D	TOL	MAT'L	FINISH
TITLE DSTD 102 (POWER & GND. TABLE)			
DWG NO DY00449-I-A1-6	ISS 6	SH OF 3	



ISSUE | DESCRIPTION | DATE | DRN | CHK
4 | ECN 0026 | 6 JUNE 82
U17 Z80 CPU 5880(-4)
U30 LOCO II 4.00/8.00 MHZ
U26 16L8
U9 PAL 16L8
U23
U25 74LS645
U24 74LS645
U22 74LS645
CANADIAN ELECTRONICS PACKAGING
TITLE DSTD-102 (PROCESSOR)
DWG NO DY00449-I-A2-6
ISS 6


ISSUE
DESCRIPTION
DATE
4
ECN 0026
J1
U21
U16
U18
(2)
U19
(1)
U20
(0)
U8
CTC
J3B2
J3B3
J3B4
J3B5
J3B6
J3B7
J3B8
J3B9
J3B10
J3B12
J3B13
GRN
RED
RRDY
2715
SCALE
APP'D
TITLE
DSTD-102 (SERIAL I/O)
DWG NO. DY00449-I-A3-6
CANADIAN ELECTRONICS PACKAGING

Z80 APPLICATION NOTES AND TECHNICAL ARTICLES

Z80 Family Interrupt Structure

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.

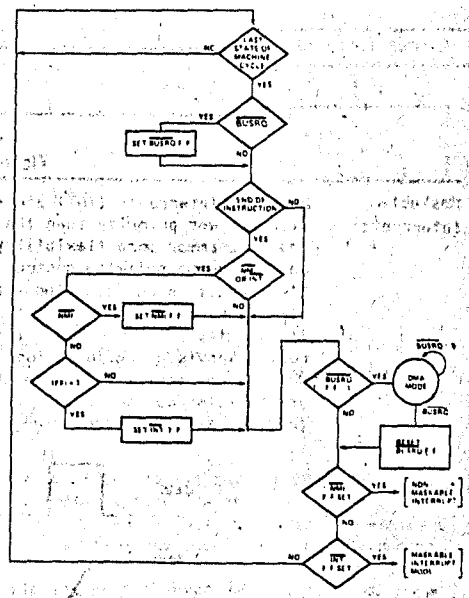


Figure 1. Z80 Flow Diagram Interrupt Sequence

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 specify 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 $\overline{NMI}$ service routine.
The RETN instruction is discussed in detail in the Z80 CPU Technical Manual. Figure 2 shows the timing used for $\overline{NMI}$ interrupts.

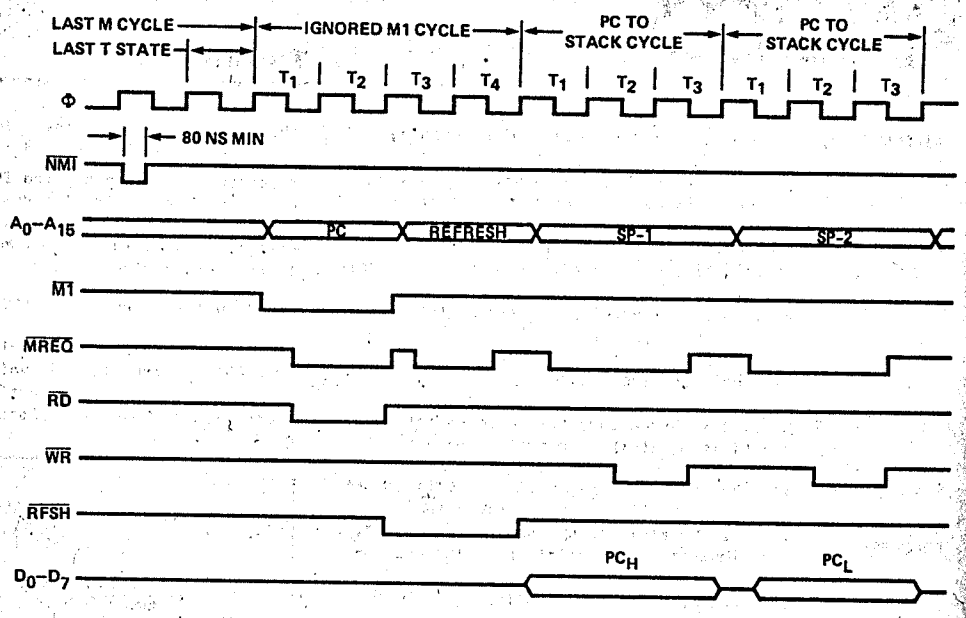


Figure 2. Non-maskable Interrupt Request Operation

Maskable Interrupts

Maskable interrupts ($\overline{INT}$) are acknowledged with a lower priority than the $\overline{NMI}$ but allow the programmer more flexibility. $\overline{INT}$ is enabled under software control by way of the EI instruction and disabled via the DI instruction. When the Z80 CPU samples $\overline{INT}$ and it is active, the processor begins an interrupt acknowledge cycle so long as $\overline{BUSRQ}$ and $\overline{NMI}$ are not active. The processor does not use an interrupt acknowledge signal but instead issues the acknowledge by executing a special $\overline{M1}$ cycle. During an interrupt acknowledge cycle, $\overline{RD}$ is inactive, $\overline{IORQ}$ 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.

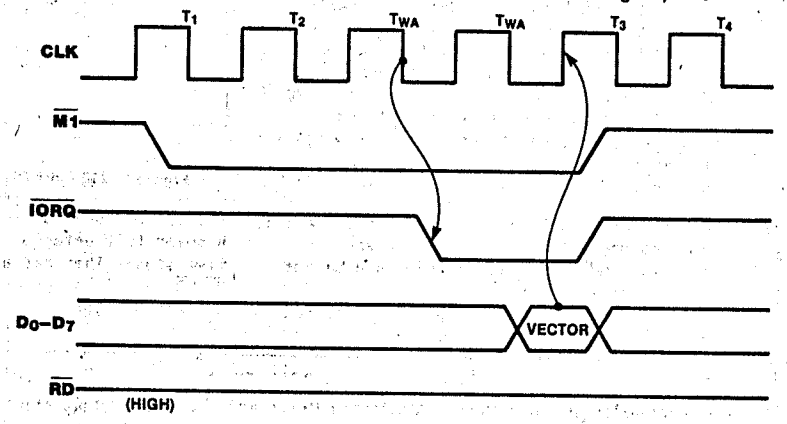


Figure 3. Interrupt Acknowledge Cycle

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.

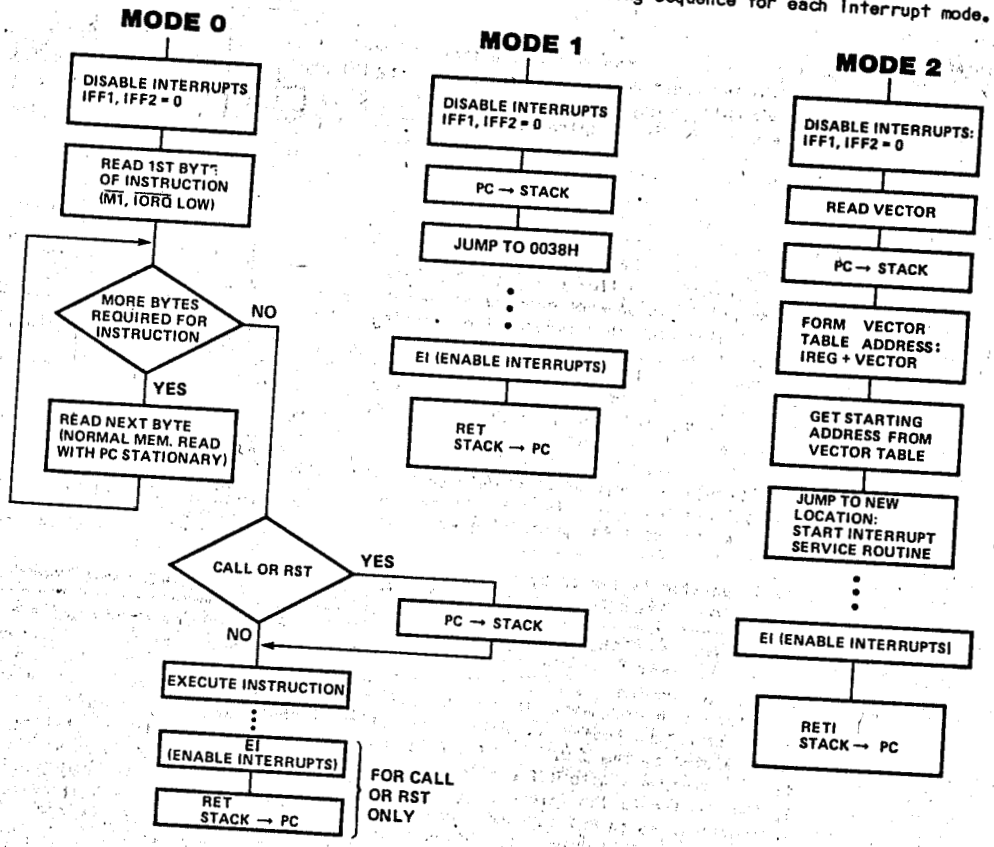


Figure 4. Maskable Interrupt Sequences

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 $\overline{M1}$ 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 $\overline{RESET}$, the CPU automatically sets Interrupt Mode 0.

Maskable Interrupt Mode 1

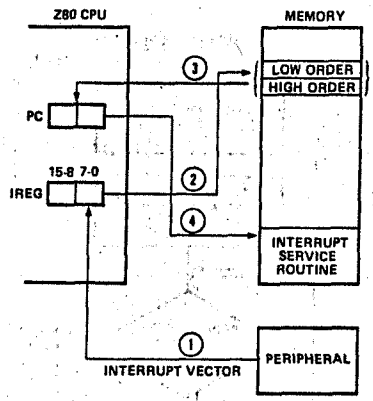
Interrupt Mode 1 provides minimally complex peripherals access to interrupt processing. It is similar to the $\overline{NMI}$ interrupt, except that the CPU automatically CALLs to location 38H instead of 66H. As with the $\overline{NMI}$, the CPU pushes the Program Counter onto the stack automatically (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.

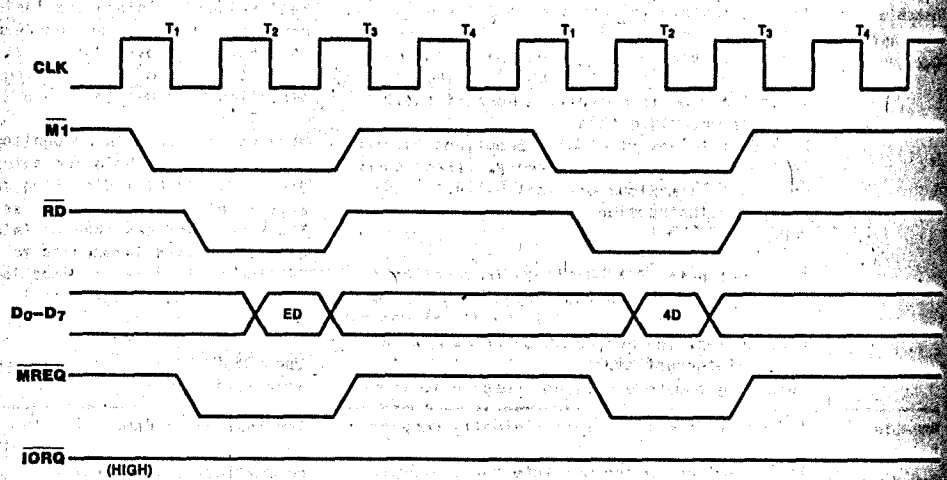


Figure 6. Return From Interrupt Timing (RETI) for Mode 2 Interrupts

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.

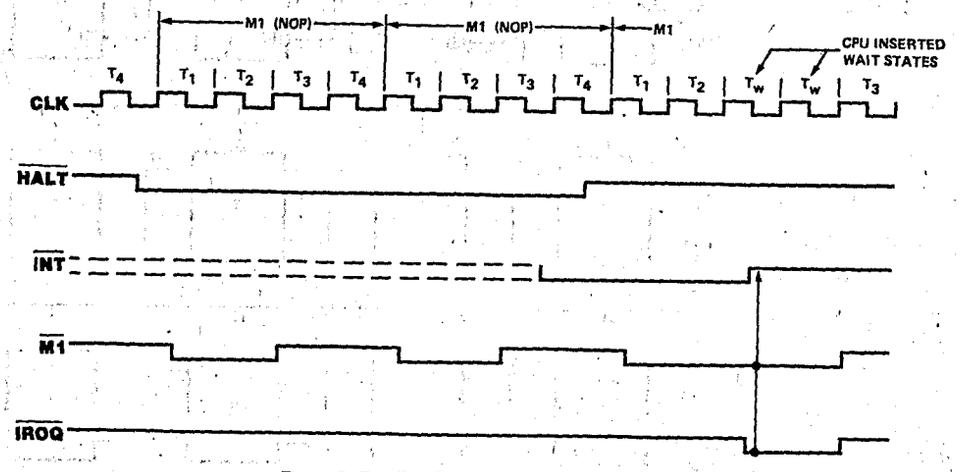


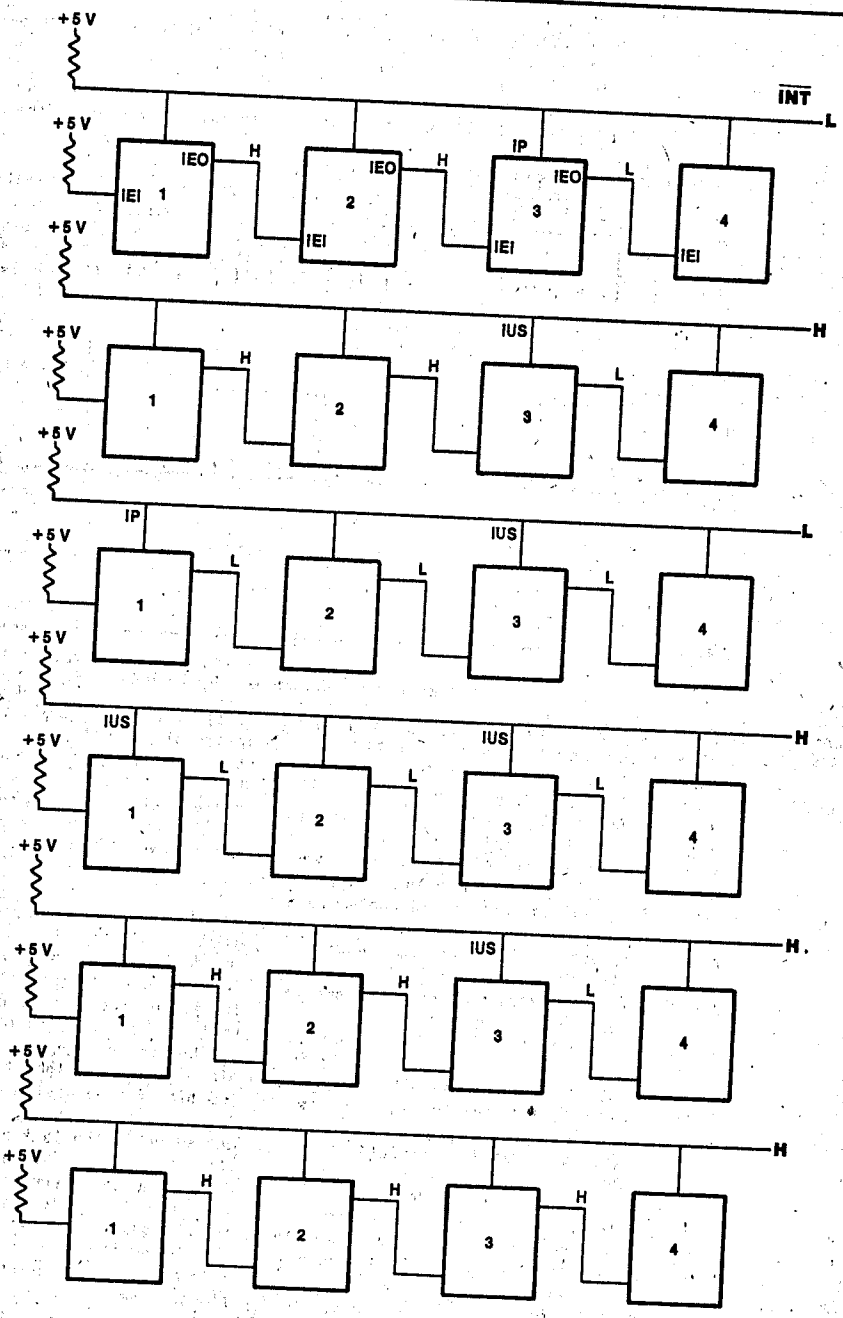
Figure 7. Exit Halt State with Maskable Interrupt

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.

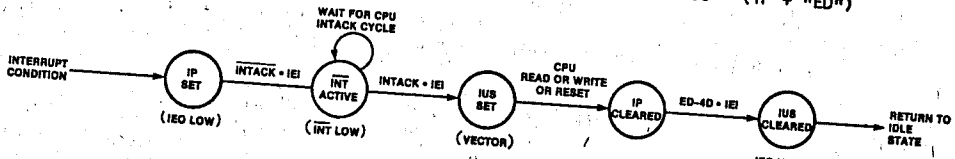
Figure 8. Z80 Peripheral Device Interrupt Processing Sequence

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.

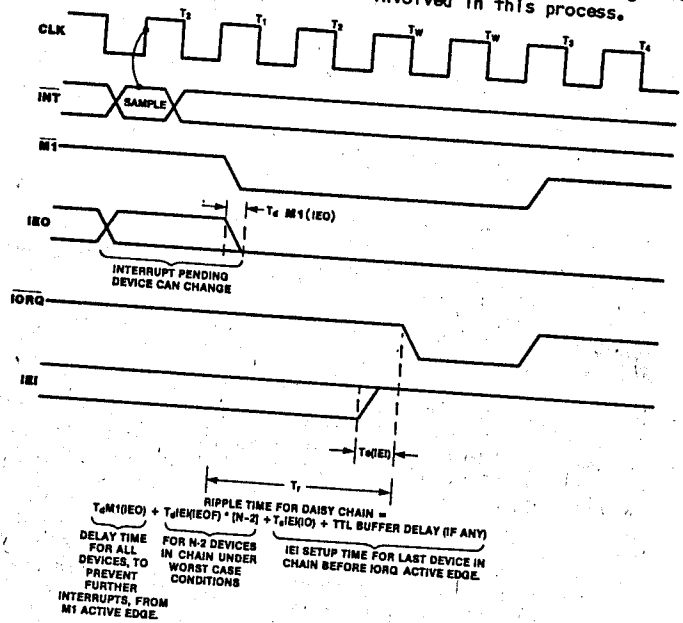


Figure 11. Interrupt Acknowledge Peripheral Propagation Delay

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.

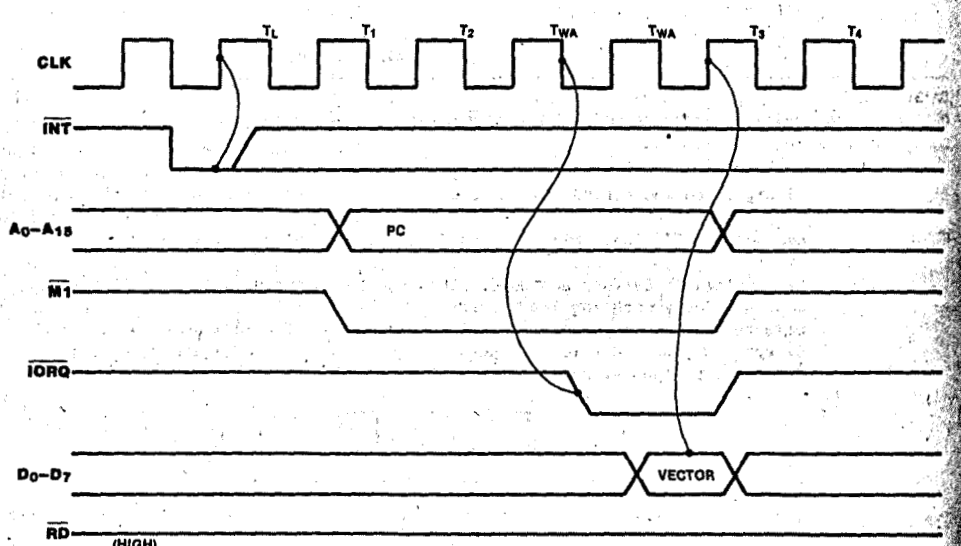


Figure 9. Peripheral Interrupt Acknowledge

Return from Interrupt Operation

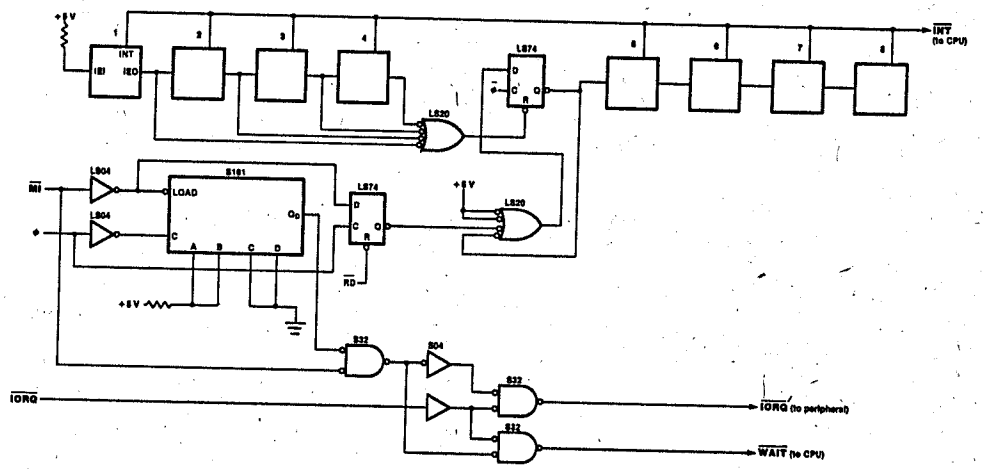
When the CPU executes an RETI instruction, the device with an interrupt under service resets its interrupt condition, provided that IEI is High. All Z80 peripheral products sample the data bus for this instruction when $\overline{M1}$ goes active along with $\overline{RD}$.

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

The Z80 CPU automatically inserts two wait states during $\overline{INTACK}$, 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 $\overline{INTACK}$ 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 $\overline{IORQ}$ goes active at the proper time. Normally, the CPU activates $\overline{IORQ}$ 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 $\overline{IORQ}$ goes active during the first wait state and the peripherals assert their vectors when $\overline{IORQ}$ becomes active, $\overline{IORQ}$ 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). $\overline{IORQ}'$ is the delayed $\overline{IORQ}$ that activates the peripheral devices.



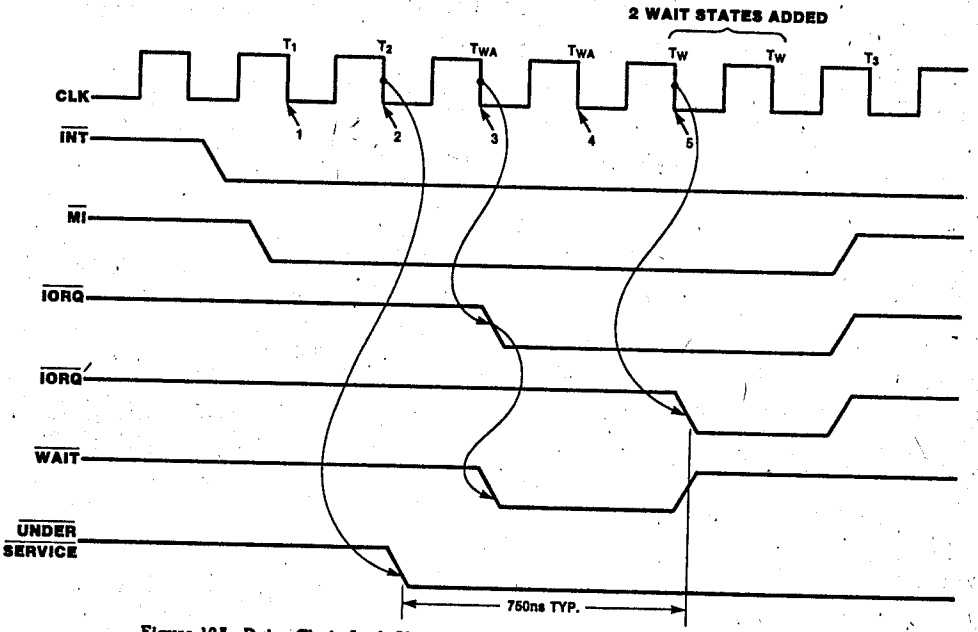


Figure 12A. Daisy Chain Look-Ahead Logic for More Than Four Peripheral Devices

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 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.

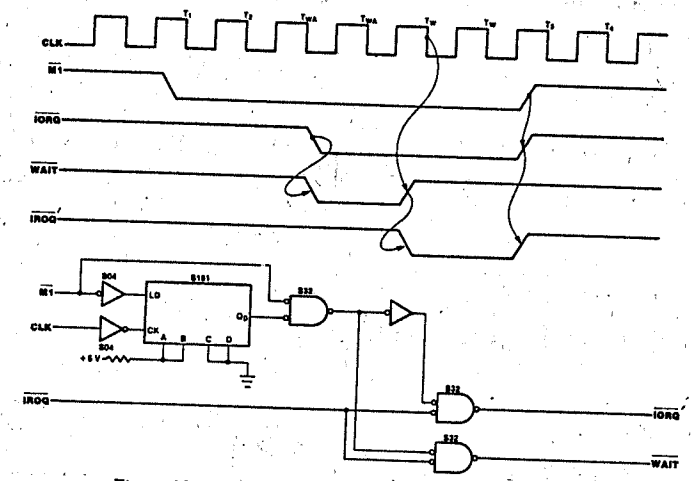
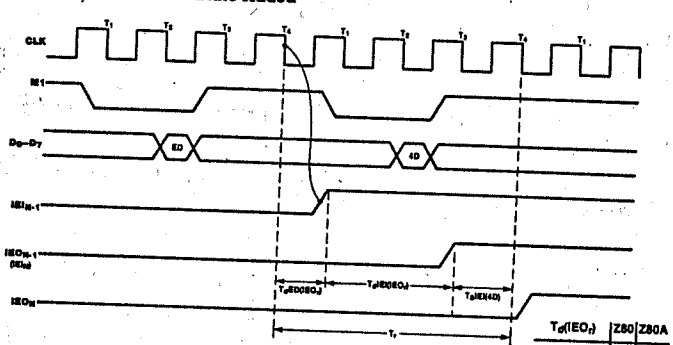


Figure 13. Wait State Logic for Interrupt Acknowledge Cycle. Counter Preset Value Should Be 5-n, Where n = # Wait State Added



T_dIEO_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.

$$T_r \geq T_dED(IEO_r) + \underbrace{T_dIEI(IEO_r) * |N-2|}_{\text{for N-2 devices}} + T_sIEI(4D)$$

$T_dED(IEO_r)$ = 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.

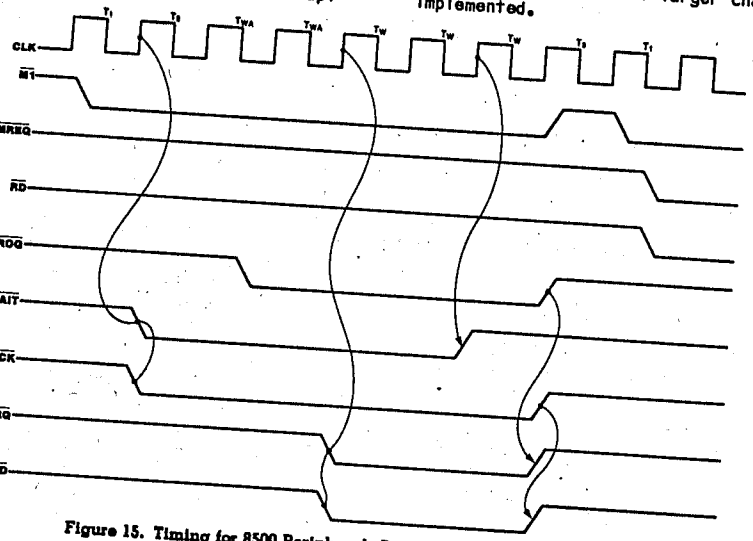


Figure 15. Timing for 8500 Peripherals During Interrupt Acknowledge.

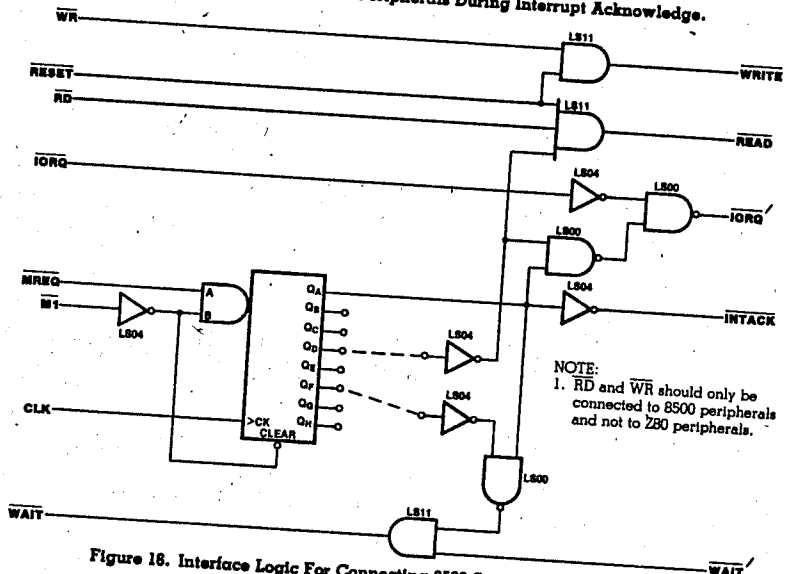
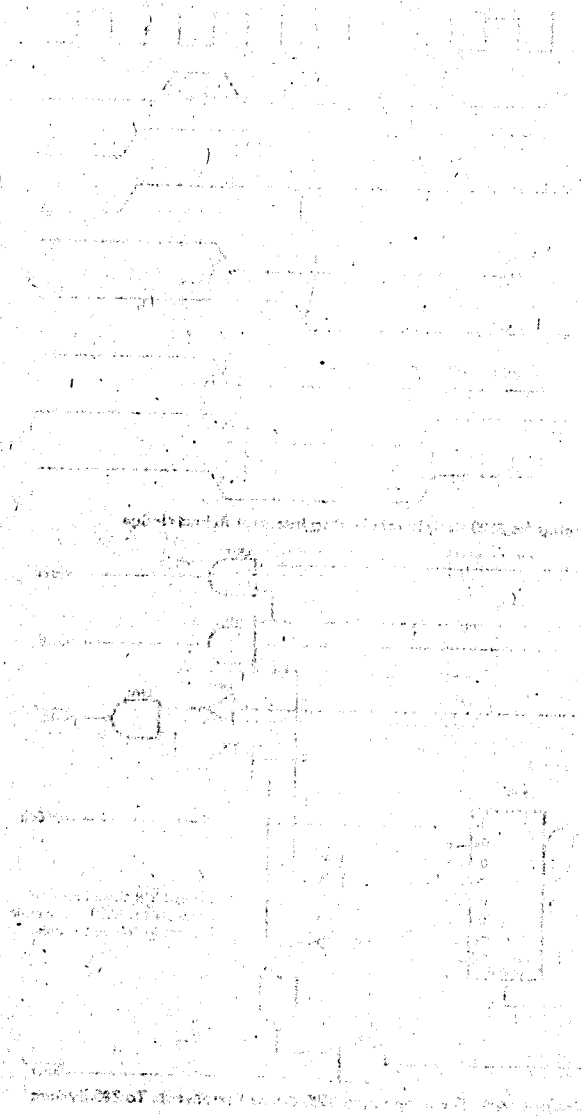


Figure 16. Interface Logic For Connecting 8500 Series Peripherals To Z80 System

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 $\overline{NMI}$ occurs during the RESET operation, the CPU executes one instruction after the RESET condition and before acknowledging the $\overline{NMI}$. Processing then continues as usual.



Using the Z80® SIO In Asynchronous Communications

ZiLOG

Application Note

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.

rotocol

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.

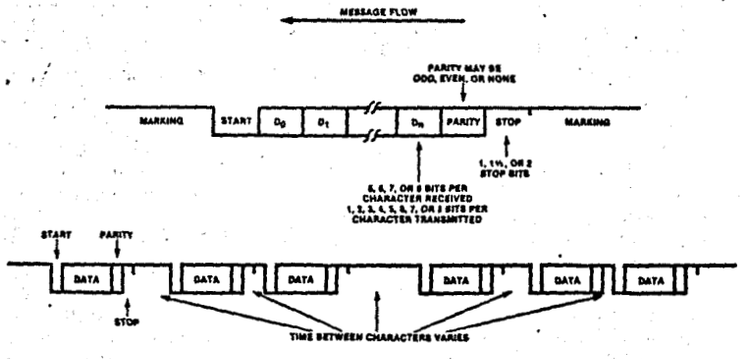


Figure 1. Asynchronous Data Format

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*

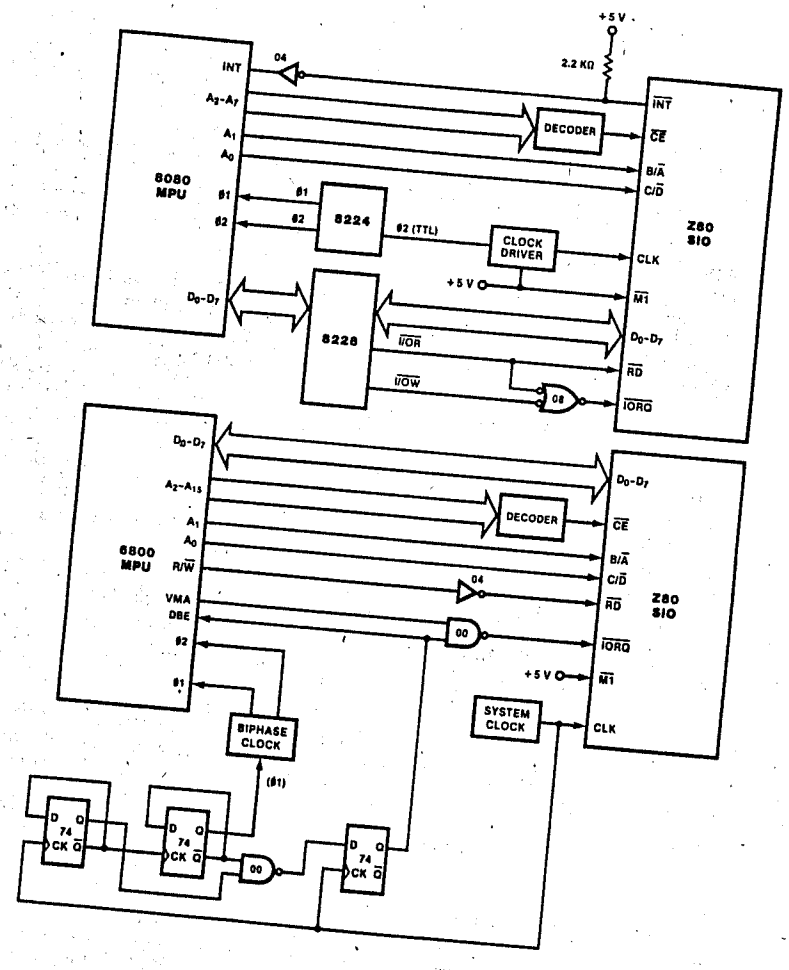


Figure 2. SIO Hardware Interfacing

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 I 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.

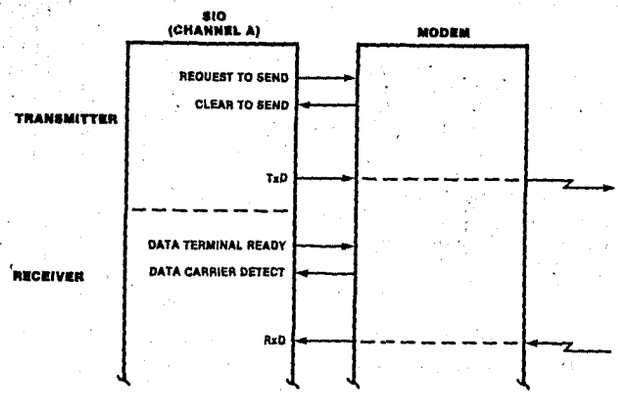


Figure 3. Modem Control (Single Channel)

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.

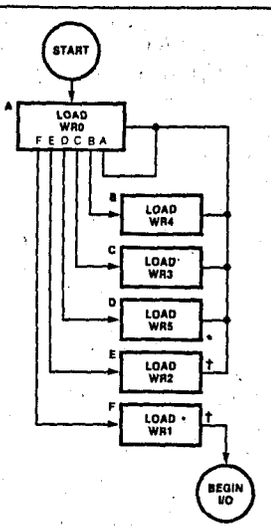


Figure 4. Typical Initialization Sequence (One Channel)

26-0003-0343

26-0003-0344

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.

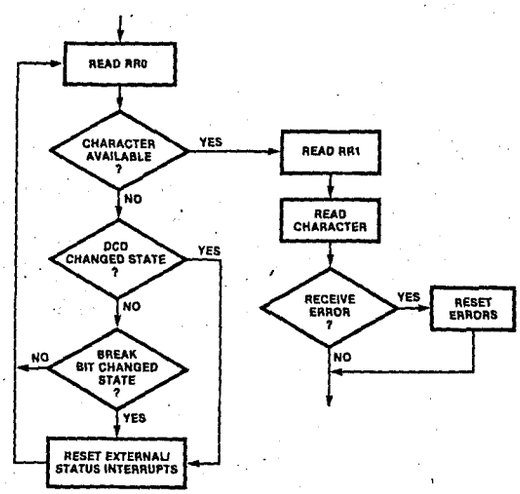


Figure 5. Polled Receive Routine

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.

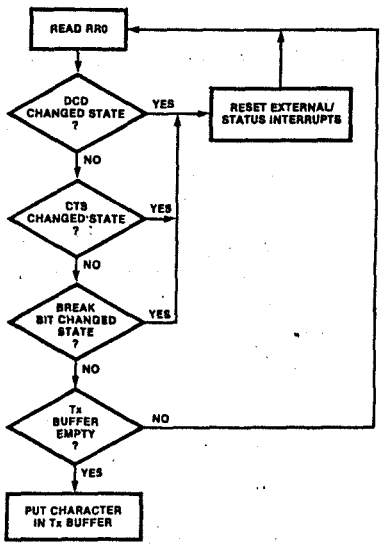


Figure 6. Polled Transmit

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)

C/$\overline{D}$	D_7
1	0
1	0
1	1
1	0
1	1
1	0
1	1

Reset and Error Sequences

In the re
low, we tre
Data Carri
sequence"
RR0 to refl
the pins. T
the Reset E
mand and
The comma
register WF

D_7	D_6
0	0

Permits the sta

This comr
for such thi
4-6 of regist
occurs and

Receive and Transmit Routines

Now we w
of the recei
preceding d
Reception."

The framir
on a charac

Initialization (Continued)

Bits sent to the SIO									Effects and Comments
$C/\overline{D}$	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
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)

C/D	Bits sent and received 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.

C/D	Bits sent and received 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)

Th
inter
each
WR1
an ir
the t
chan
rupt.
prior
prior
Chan
3-1
Chan
to 1
Chan
to 1
Chan
3-1
Chan
3-1
Chan
to 0
Chann
to 0(
Chann
3-1 t
For
tor hac
Affects
interru
transiti
(bits 3-
an inte
the inte
tained

Initialization

In ge
illustrat
six step
After cc
the sam
necessa
of Chan
register
to be en
interrup
Now l
example
device t
characte
Enables
provide
We dc
registers
Technic
bit assig
Input/Ou
at the en

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.

	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 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).

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

Hardware Considerations

Questions and Answers.

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 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
```

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, $\overline{M1}$ should be tied High and no equivalent to an interrupt acknowledge should be issued.

Q: How can the Wait/Ready ($\overline{W/RDY}$) signal be used by the CPU in asynchronous I/O?

A: The $\overline{W/RDY}$ signal is most commonly used in Block Transfer Mode with a DMA, and this use is described in the *Z80 DMA Technical Manual*. However, $\overline{W/RDY}$ may be directly connected to the Z80 CPU $\overline{WAIT}$ 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 $\overline{W/RDY}$ line. The block I/O instructions decrement a byte counter to determine when I/O is complete.

Q: Can the $\overline{SYNC}$ 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.

SECTION 6

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

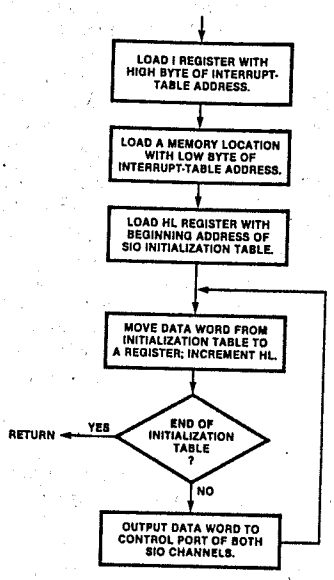


Figure 7. Interrupt-Driven Initialization Routine

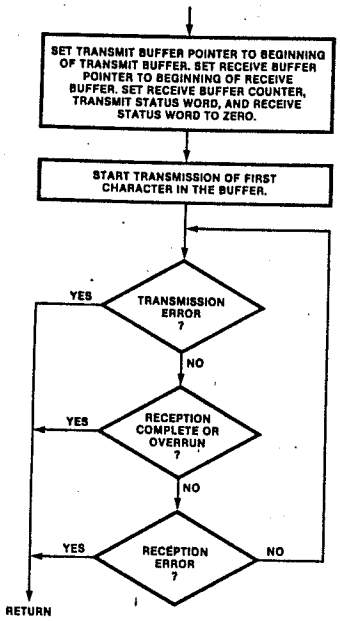


Figure 8. Interrupt-Driven Transmit Routine

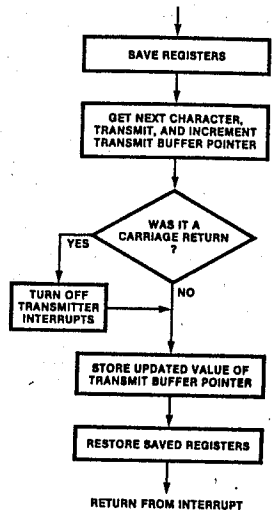


Figure 9. Transmitter Buffer Empty Interrupt Routine

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.

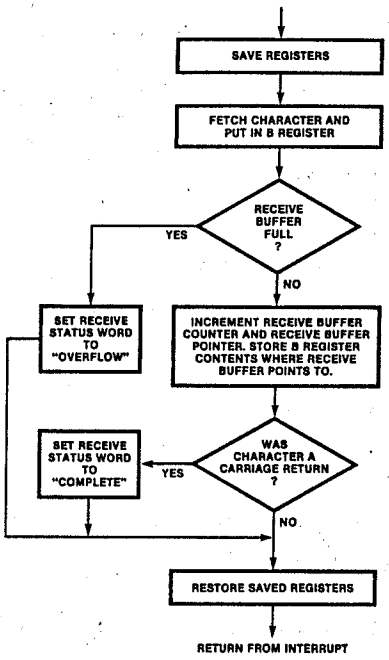


Figure 10. Receive Character Interrupt Routine

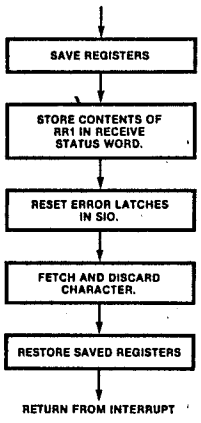


Figure 11. Special Receive Condition Interrupt Routine

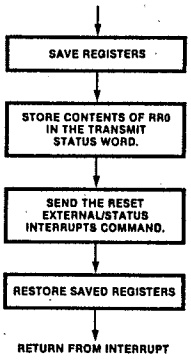


Figure 12. External/Status Interrupt Routine

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,SIOADData
            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,SIOADData
            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,SIOADData
            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
```

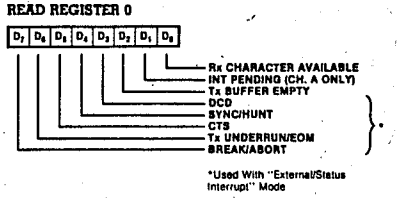
External/Status Routine (see Figure 12)

```
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

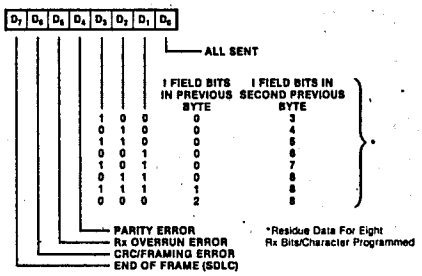
Read Register Bit Functions

READ REGISTER 0



*Used With "External/Status Interrupt" Mode

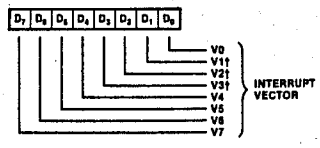
READ REGISTER 1†



*Residue Data For Eight Rx Bits/Character Programmed

†Used With Special Receive Condition Mode

READ REGISTER 2

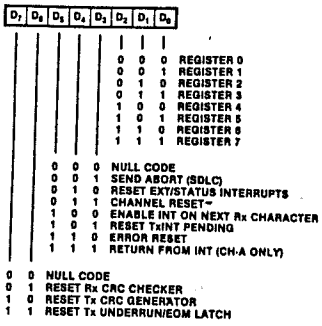


†Variable if "Status Affects Vector" is Programmed

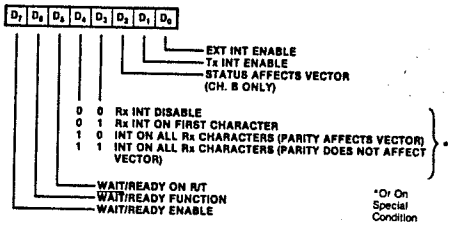
Appendix C

Write Register Bit Functions

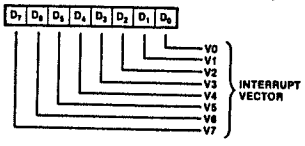
WRITE REGISTER 0



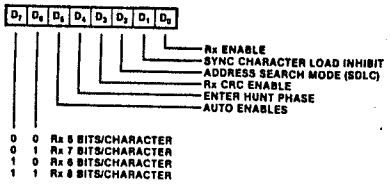
WRITE REGISTER 1



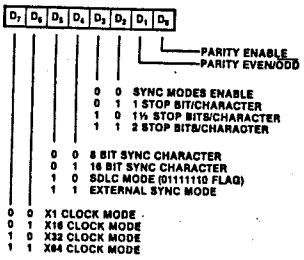
WRITE REGISTER 2 (CHANNEL B ONLY)



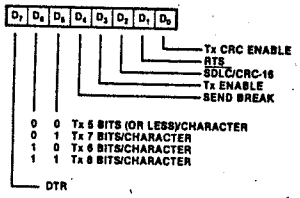
WRITE REGISTER 3



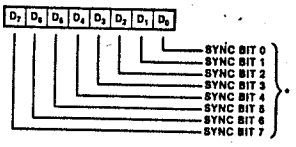
WRITE REGISTER 4



WRITE REGISTER 5

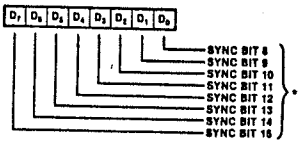


WRITE REGISTER 6



*Also SDLC Address Field

WRITE REGISTER 7



*For SDLC It Must Be Programmed to "01111110" For Flag Recognition

Using the Z80 SIO With SDLC

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.

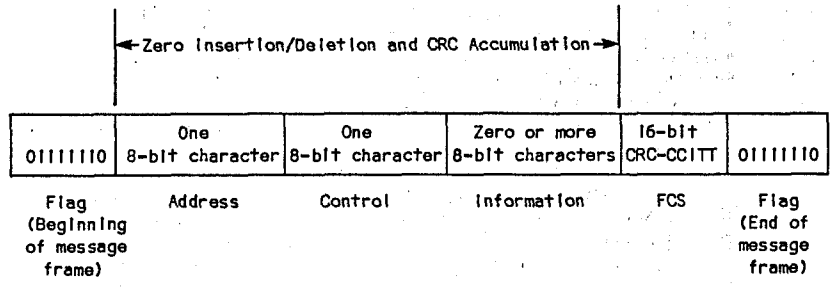


Figure 1. A Typical SDLC Message Frame Format

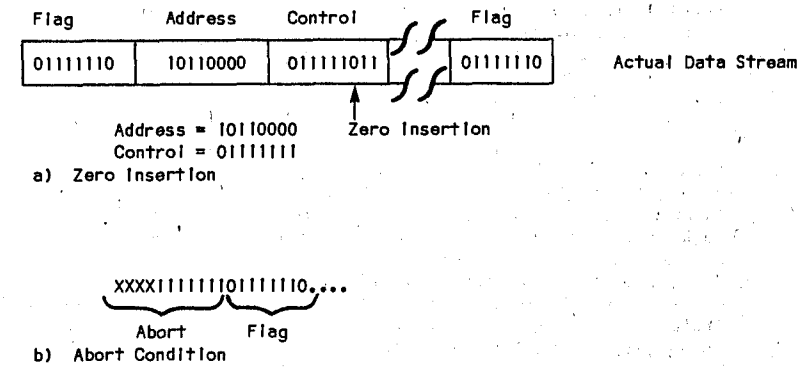


Figure 2. Bit Patterns for Various Line Conditions

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 (01111110111111101111110...), it can only transmit flag characters without shared 0s (01111110011111100111111O...).

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.

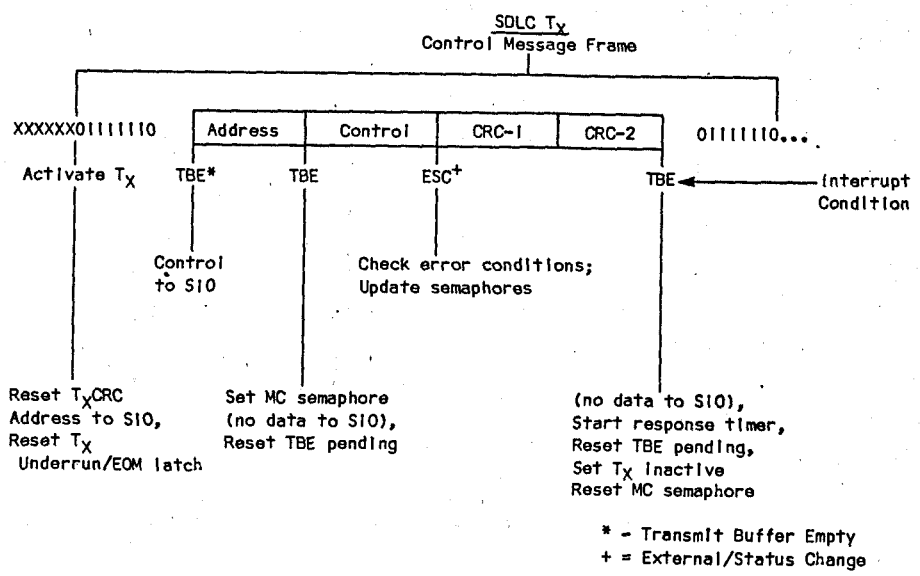


Figure 3. A Typical Transmit Control Frame Sequence

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.

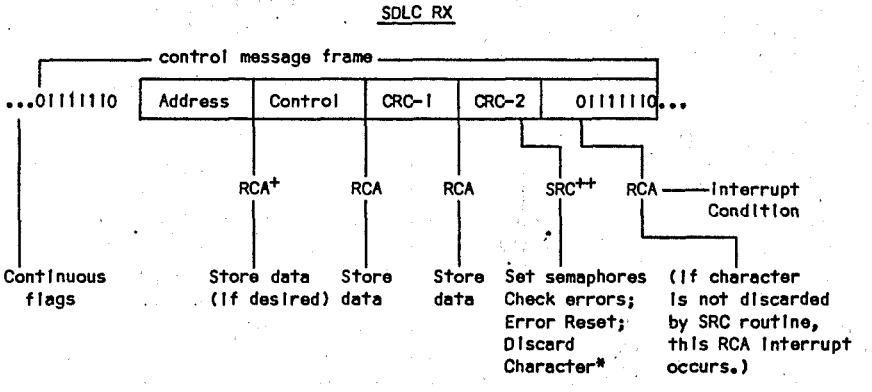
When the last data character (the control byte) has been shifted out of the SIO, the T_X underrun/EOM latch is set because the SIO buffer was not loaded with a character on the previous TBE interrupt. As a result, an External/ Status Change (ESC) interrupt occurs and the SIO begins transmitting the FCS 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 Tx inactive semaphore
   Reset Tx CRC
   Data to SIO
      (Address field byte)
   Reset Tx 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 Tx 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 Rx CRC error semaphore
   Else If (Rx overrun)
      Set Rx 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 is have already passed. Figure 6 shows the abort/idle line timing and the interrupts resulting from the line changes.

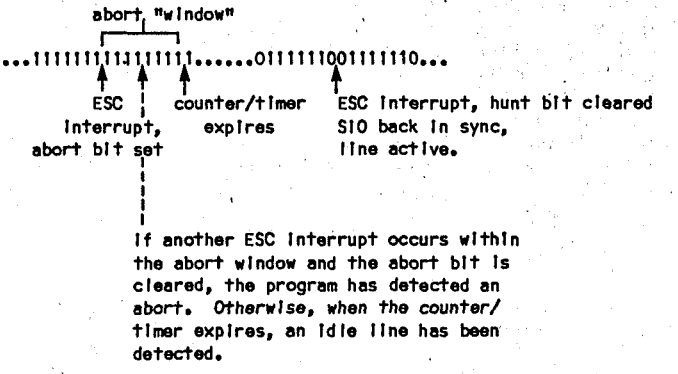


Figure 6. Abort/Idle Line Conditions

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  ;[01]      01-21-81/MDP            INITIAL CREATION
                    4
```

```
                        5  ;       THIS PROGRAM SENDS ADDRESS 9EH, CONTROL 19H,
                        6  ;       AND DATA 81H CONTINUOUSLY USING THE Z80 VECTORED
                        7  ;       INTERRUPT MODE.  THE SIO IS INITIALIZED TO USE
                        8  ;       SDLC WITH THE BAUD RATE CLOCK SUPPLIED BY
                        9  ;       HARDWARE INTERNAL TO THE SYSTEM.
                       10
                       11  ;       EQUATES
                       12
                       13  ADDRESS:          EQU     9EH         ;ADDRESS FIELD
                       14  CTRL:     EQU     19H                 ;CONTROL FIELD
                       15  DATA:     EQU     81H                 ;INFORMATION FIELD
                       16  MSGLEN:   EQU     1                   ;MESSAGE LENGTH
                       17  RAM:      EQU     2000H               ;RAM ORIGIN
                       18  RAMSIZ:   EQU     1000H               ;RAM SIZE
                       19  SIODA:    EQU     0                   ;SIO PORT A DATA
                       20  SIOCA:    EQU     SIODA+1             ;SIO PORT A CTRL
                       21  SIODB:    EQU     SIODA+2             ;SIO PORT B DATA
                       22  SIOCB:    EQU     SIODB+1             ;SIO PORT B CTRL
                       23  CIOC:     EQU     8                   ;CIO PORT C
                       24  CIOB:     EQU     CIOC+1              ;CIO PORT B
                       25  CIOA:     EQU     CIOC+2              ;CIO PORT A
                       26  CIOCTL:   EQU     CIOC+3              ;CIO CTRL PORT
                       27  BAUD:     EQU     9600                ;ASYNC BAUD RATE
                       28  RATE:     EQU     BAUD/100
                       29  CIOCNT:   EQU     9216/RATE
                       30  LITE:     EQU     0E0H                ;LIGHT PORT
                       31  RSPCNT:   EQU     100                 ;RESPONSE TIMER VALUE
                       32
                       33  ;       SIO PARAMETERS
                       34
                       35  SIOWR0:   EQU     0
                       36          CHRES:    EQU     18H         ;CH. RESET CMD
                       37          ESCRES:   EQU     10H         ;ESC RESET CMD
                       38          TBERES:   EQU     28H         ;TBE RESET CMD
                       39          RETIA:    EQU     38H         ;RETI CH. A
                       40          ENINRX:   EQU     20H         ;ENAB. INT. NEXT RX
                       41          SRCRES:   EQU     30H         ;SRC RESET CMD
                       42          RCRCRE:   EQU     40H         ;RX CRC RESET CMD
                       43          TCRCRE:   EQU     80H         ;TX CRC RESET CMD
                       44          EOMRES:   EQU     0C0H        ;EOM RESET CMD
                       45
                       46  SIOWR1:   EQU     1
                       47          WREN:     EQU     80H         ;WAIT/RDY ENABLE
                       48          RDY:      EQU     40H         ;READY FUNCT.
                       49          WRONR:    EQU     20H         ;WAIT/RDY ON RX
                       50          RXIFC:    EQU     8           ;RX INT. FIRST CHAR
                       51          RXIAP:    EQU     10H         ;RX INT. ALL + PARITY
                       52          RXIA:     EQU     18H         ;RX INT. ALL
                       53          SIOSAV:   EQU     4           ;STATUS AFFECTS VECT.
                       54                                        ;(CH.B ONLY)
                       55          TXI:      EQU     2           ;TX INT. ENABLE
                       56          EXTI:     EQU     1           ;EXT. INT. ENABLE
                       57
                       58  SIOWR2:   EQU     2                   ;(CH.B ONLY)
                       59
                       60  SIOWR3:   EQU     3
                       61          RX8:      EQU     0C0H        ;RX 8 BITS
                       62          RX6:      EQU     80H         ;RX 6 BITS
                       63          RX7:      EQU     40H         ;RX 7 BITS
                       64          RX5:      EQU     0           ;RX 5 BITS
                       65          AUTOEN:   EQU     20H         ;AUTO ENABLES
                       66          HUNT:     EQU     10H         ;HUNT MODE
                       67          RXCRC:    EQU     8           ;RX CRC ENABLE
                       68          ADSRCH:   EQU     4           ;ADDR SEARCH
                       69          SYNINH:   EQU     2           ;SYNC LOAD INHIBIT
                       70          RXEN:     EQU     1           ;RX ENABLE
                       71
                       72  SIOWR4:   EQU     4
                       73          X64:      EQU     0C0H        ;64X CLOCK
                       74          X32:      EQU     80H         ;32X CLOCK
                       75          X16:      EQU     40H         ;16X CLOCK
                       76          X1:       EQU     0           ;1X CLOCK
```

```
LOC   OBJ CODE M STMT SOURCE STATEMENT
                    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
```

```
LOC   OBJ CODE  M STMT SOURCE STATEMENT
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
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
```

```
012F  D303          295            OUT     (SIOCB),A
0131  C9            296            RET
                    297  SETDVR:
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
                   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+SYNC6N
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

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.

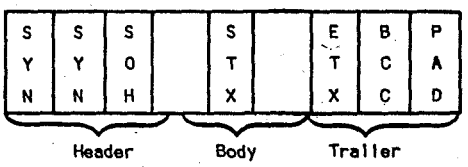


Figure 1. Basic Message Block Format for Bisync Protocol

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.

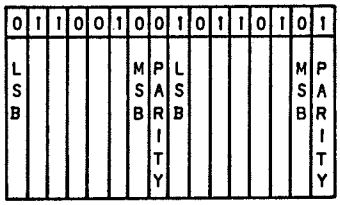


Figure 2. Odd VRC.
Number of 1s should be odd.

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.

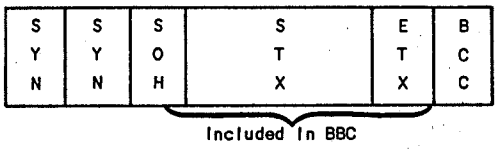


Figure 3. Characters Included in BBC

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.

Timing in an Interrupt-Based System with the Z80® CTC

ZiLOG

Application Note

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.

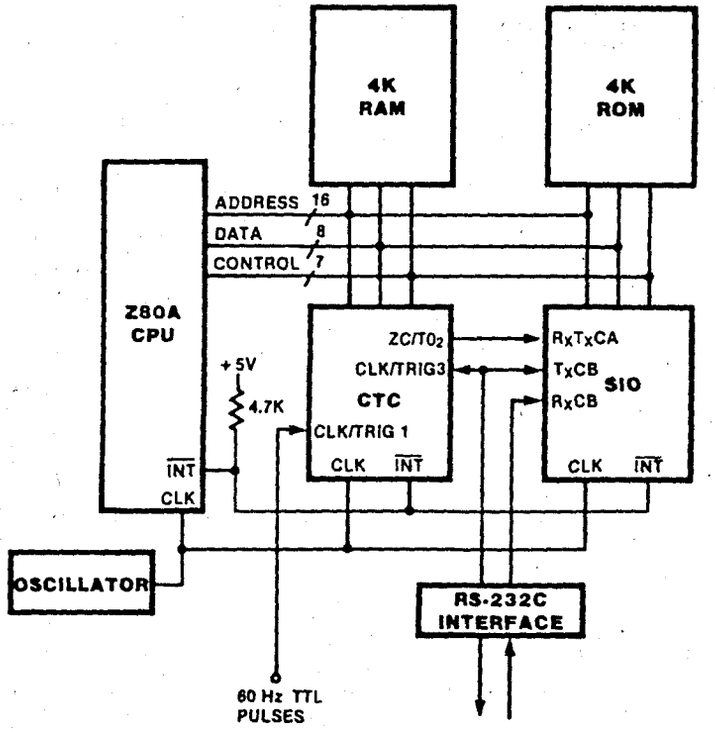


Figure 1. Z80A System Block Diagram

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.

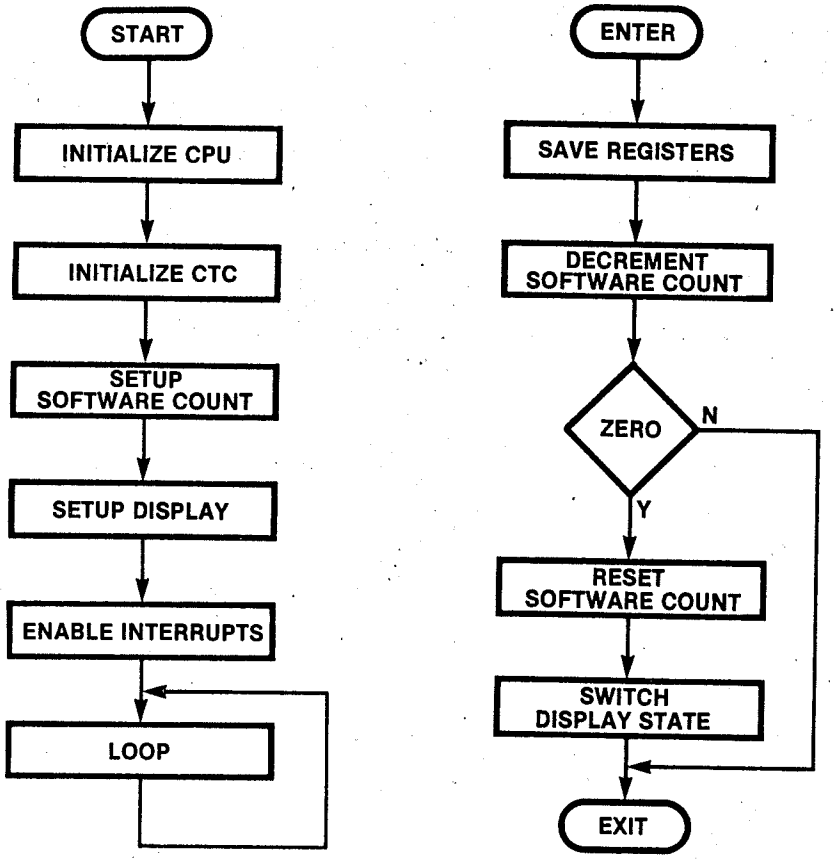
In the example shown, the interrupt interval is set to 8.33 ms, which is provided by the CTC with a 3.6864 MHz input clock, 256 prescaler value, and a time constant value of 120. The CTC interrupt service routine uses a software count of 120 to maintain a one-second system time interval. Each time the 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

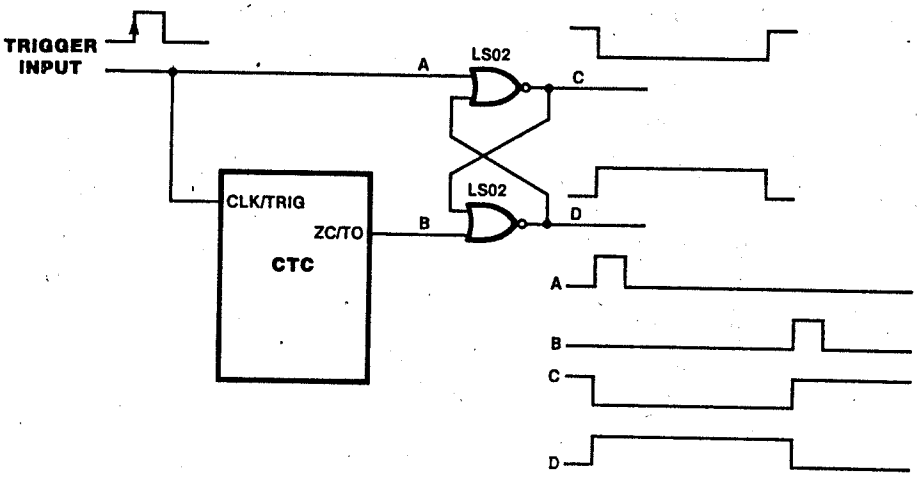


Figure 3. Monostable Multivibrator Using the Z80 CTC

```
                                        TEST.CTC0
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  CTC0:   EQU     12              ;CTC 0 PORT
                     12  CTC1:   EQU     CTC0+1          ;CTC 1 PORT
                     13  CTC2:   EQU     CTC0+2          ;CTC 2 PORT
                     14  CTC3:   EQU     CTC0+3          ;CTC 3 PORT
                     15  LITE:   EQU     0E0H            ;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.0FFF0H.OR.10H
                     39  INTVEC:
0010  4000           40          DEFW    ICTC0
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     (CTC0),A        ;SET CTC MODE
002B  3E78           58          LD      A,TIME
002D  D30C           59          OUT     (CTC0),A        ;SET TIME CONSTANT
002F  3E10           60          LD      A,INTVEC.AND.11111000B
0031  D30C           61          OUT     (CTC0),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.CTC0
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  ICTC0:
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   C0           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   D3E0         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
```

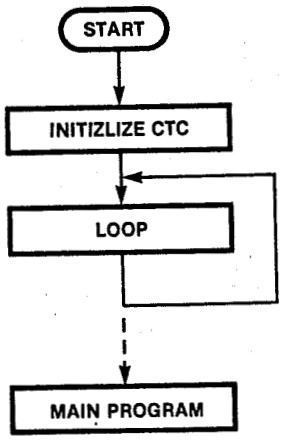


Figure 4. Software for CTC Bit Rate Generator

```
                                        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

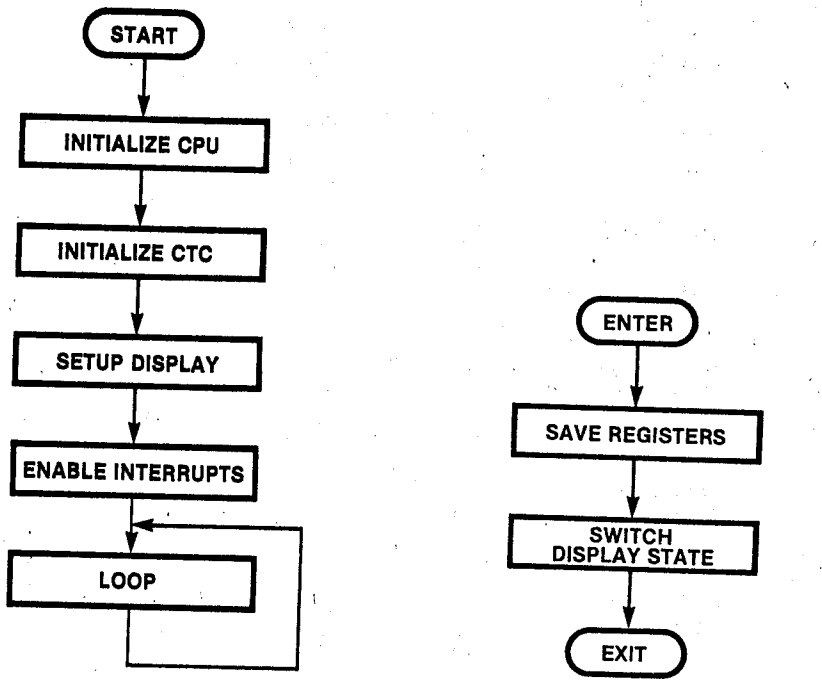
A typical computer system often uses a time-of-day clock. In the United States, the 60 Hz power line provides an accurate time base for synchronous motor clocks. A computer system can take advantage of the 60 Hz accuracy by incorporating a circuit that feeds 60 Hz square waves into a CTC channel. With 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

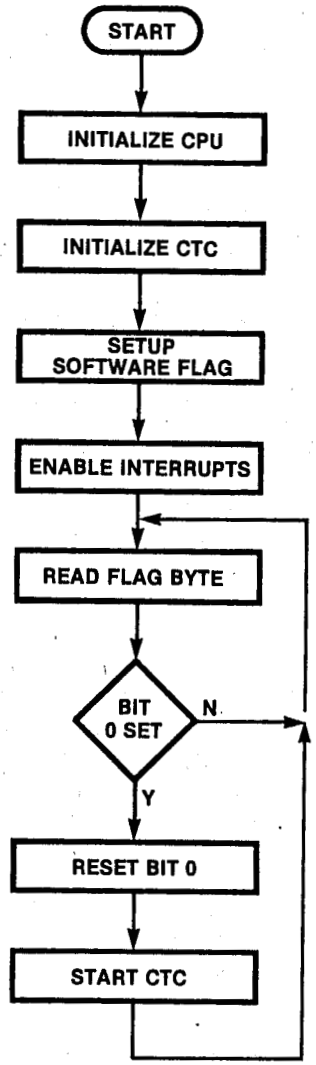
b) Interrupt Service Routine

Figure 5. Software for CTC Counter Mode

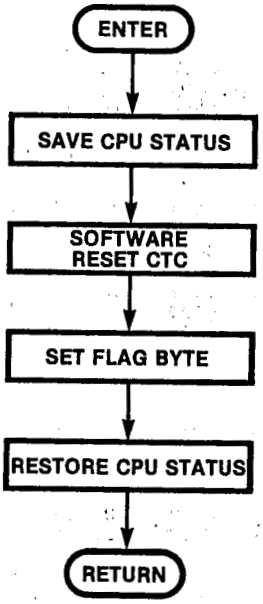
```
                                TEST.CTC1
LOC   OBJ CODE M STMT SOURCE STATEMENT

                    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.0FFF0H.OR.10H
                    36  INTVEC:
 0010   3800        37          DEFW    ICTC0
 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     (CTC0),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  ICTC0:
                    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 MO
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.