Z80 Family Questions & Answers

This application note contains the most commonly asked questions about the Zilog Z80 Family. They are divided into following sections:

- Z80 CPU
- Z80 DMA
- Z80 PIO
- Z80 CTC
- Z80 SIO, Z80 DART

Obviously, not every questions on Z80 Family components are answered. However, this application note should give you a good feel for the Z80 Family devices. Along with the technical Manual, Product Specification and some other application notes, it should help make your Z80 design family a little easier. Also, this Application Note is applicable to the Z80 KIO and other Z80 family based Super Integration Devices.

Z80 CPU

Q: Are the Z80 CPU 6 and 8MHz clocks sensitive like their predecessors?
A: Yes, specifications for rise and fall times and clock voltage levels must be met.

Q: Can the rising edge on the CLK input affect the operation of the CPU?
A: Very much so. For NMOS devices, a negative voltage spike on any pin without back bias will forward-bias the diode that exists between the N+ material connected to the pad and p-type substrate. This action causes the injection of many electrons into the substrate. Once in the substrate, they are free to drift into any region of higher potential, which is the N+ region at Vcc of storage nodes storing a "1". Since storage holds don't store much charge (in order to minimize capacitance), these electrons in the substrate can be swept across the junction and destroy the "1" stored there. This reaction obviously affects the operation of the part.

Also, on CMOS devices, positive spikes on any pin exceeding Vcc voltage could cause "Latch-up"!

Q: What is the clock input impeadance (load)?
A: Capacitive load only (35pF max).

Q: Will Non-maskable interrupts continue occurring and executing if the NMI line pulses prior to the finish of the service routine?
A: Yes. Non-Maskable interrupts can not be disabled by user. Even though Non-Maskable Interrupts are negative edge triggered, if the input to the CPU pulses before termination of the service routine, then the service routine will begin again.

Q: How does the Non-Maskable Interrupt acknowledge cycle and RETN instruction actually work?
A: When a Non-Maskable Interrupt is acknowledged, interrupt flip-flop #1 (IFF1) is actually cleared to inhibit the acknowledgement of maskable interrupts. The state of interrupt flip-flop #2 (IFF2) is not altered. This is the only time that the contents of IFF1 and IFF2 can disagree. When the RETN instruction executes, the state of IFF2 is copied back into IFF1. This allows the state of maskable interrupts, before a Non-Maskable Interrupt, to be restored after service routine execution.

Q: How are subtraction operations performed?
A: Although the actual operation is probably a 2's complement addition, the flags are affected as if it were a logical subtraction operation.

Q: What is the setup time to recognize an NMI?
A: Through characterization of the CPU, Zilog has found that a setup time of 120nS (@ 4MHz) is required in order to assure that NMI is recognized before INT is recognized.

Q: What do the EI and DI instruction actually do?
A: Only the interrupt control flip-flops (IFF1 and IFF2) are affected by those instructions. The DI instruction will clear both IFF1 and IFF2 and prevent any further maskable interrupt from being recognized from that point on. The EI instruction will set both IFF1 and IFF2, but maskable interrupts will not be recognized till the completion of the next instruction.

Q: What is the status of the output drivers when the CPU is in a power-down situation?
A: When the CPU is without power, the output drivers appear to be in a high impedance state.

Q: How can I use the on chip refresh mechanism of the Z80 CPU to handle refreshing of 64K D-RAMs?
A: Here are some suggestions (assuming 256 cycle refresh):

1. Use an external counter to count 128 M1 cycles and toggle refresh address line A7.
2. Use external hardware to generate an NMI every 2mS and change the state of bit D7 in the R Register via software.
3. Use refresh address bit A6 to toggle the state of refresh address bit A7.

Q: Is there a method for testing hardware without removing the Z80 CPU from the socket?
A: Two methods are available:

1. Use BUSREQ to tri-state all control signals and then use external hardware to simulate the logic;
2. Remove power and ground from the CPU, all signals should go to a high-impedance.

Q: Does the CPU tristate M1 during reset?
A: No.

Q: Is Zilog going to add a 3.15 Volt current drive spec for designers using 74HCxx series of components?
A: No. There is a choice of either using 74HCTxx logic or using our CMOS Z80 CPU.

Q: If NMI is activated DURING reset, will the processor execute the NMI or address 0000h after reset goes high?
A: Since NMI input is "edge-triggered" input, if the CPU has active NMI "during" reset, CPU won't detect NMI and will execute the instruction at 0000h. If NMI goes low after RESET goes inactive, then CPU will process NMI.

Q: I've heard the CPU is a static device. Can I use the clock to single step it?
A: It's different for NMOS and CMOS.

NMOS: No, it violates the clock specs.
CMOS: Yes. You can do that.

Q: I don't seem to get the correct state of the interrupts when using the LD A,I and LD A,R instructions to read the state of IFF2. Why is this? How can I get around this?
A: On CMOS Z80 CPU, we've fixed this problem. On NMOS Z80 CPU, in certain narrowly defined circumstances, the Z80 CPU interrupt enable latch, IFF2, does not necessarily reflect the true interrupt status. The two instructions LD A,R and LD A,I copy the state of interrupt enable latch (IFF2) into the parity flag and modifies the accumulator contents (See table 7.0.1 in the Z80 CPU technical manual for details). Thus, it is possible to determine whether interrupts are enabled or disabled at the time that the instruction is executed. This facility is necessary to save the complete state of the machine. However, if an interrupt is accepted by the CPU during the execution of the instruction -- implying that the interrupts must be enabled -- the P/V flag is cleared. This incorrectly asserts that interrupts were disabled at the time the instruction was executed.

This paradox can be traced to the internal timing of the CPU. The problem is that the interrupt flip-flop (IFF2) is cleared before it is actually transferred to the P/V flag. The state of the interrupt enable latch is not copied into the parity flag until after the interrupt time, occurring during the execution of the instruction, has been accepted. Since the acceptance of the interrupt automatically clears the interrupt enable latch, the parity flag is also cleared, despite the fact that interrupts were enabled when the instruction started executing.

A neat solution to this anomaly relies on the fact that at least one item -- the old PC value -- is saved on the stack when an interrupt is accepted. The "next entry" position on the stack (the word below the address currently held in the stack pointer) may be cleared before execution of LD A,I (or LD A,R). If that zero value has changed by the time that the next instruction in the routine is executed, then an interrupt must have been accepted. This implies that interrupts were enabled, even if the state of the parity flag suggests that they were not. Of course, if the parity flag is found to be set after LD A,R (LD A,I) has been executed, there is no need to check the stack top. Interrupts are definitely enabled if the parity flag is in this state.

Two routines are listed here. Both return carry clear if interrupts are enabled, set otherwise. Both corrupt the A register; it does not contain the value in the I (or R) register on exit. The status of all flags except the carry flag are undefined on exit.

The first routine may be loaded anywhere in memory except "page zero" -- 0000h to 00FFh. This small restriction comes about because the routine checks only the most significant byte of the "next" stack entry. This byte will be non-zero after an interrupt has occurred if and only if the routine itself is not on page zero. The second routine tests both bytes of the "next" entry and, therefore, overcomes this restriction.

Caution, these routines presume that the service routine for any acceptable interrupt will re-enable interrupts before it terminates. This is almost always the case. They may not return the correct result if an interrupt service routine, which does not re-enable interrupts, is entered after the execution of LD A,I (or LD A,R).

Listing 1: This routine may not be loaded in page zero (0000h to 00FFh).

```
GETIFF:
        XOR     A       ;C flag, acc. := 0
        PUSH    AF      ;stack bottom := 00xxh
        POP     AF      ;Restore SP
        LD      A,I     ;P flag := IFF2
        RET     PE      ;Exit if enabled
        DEC     SP      ;May be disabled.
        DEC     SP      ;Has stack bottom been
        POP     AF      ;overwritten ?
        AND     A       ;If not 00xxh, INTs were
        RET     NZ      ;actually enabled.
        SCF             ;Otherwise, they really are
        RET             ;disabled.
        END
```

Listing 2: This routine may be loaded anywhere in memory.

```
GETIFF:
        PUSH HL         ;Save HL contents
        XOR     A       ;C flag, acc. := 0
        LD      H,A     ;HL := 0000h
        LD      L,A
        PUSH    HL      ;Stack bottom := 0000h
        POP     HL      ;Restore SP
        LD      A,I     ;P flag := IFF2
        JP      PE,
                POPHL   ;Exit if isn't enabled
        DEC     SP      ;May be disabled.
        DEC     SP      ;Let's see if stack bottom
        POP     HL      ;is still 0000h.
        LD      A,H     ;Are any bits set in H
        OR      L       ;or in L ?
        POP     HL      ;Restore old contents.
        RET     NZ      ;HL <> 0 : isn't enabled.
        SCF             ;Otherwise, they really are
        RET             ;disabled.
POPHL:
        POP     HL      ;Exit when P flag is
        RET             ;set by LD A,I
        END
```

Q: Are all of the Z80 control lines internally synchronized?

A: The inputs in question are INT, NMI, BUSREQ, WAIT, and RESET. In the past, it seems that some of our customers have assumed that those inputs are totally asynchronous with respect to the system clock (i.e. no setup time required). Zilog's official position on this topic is as follows.

All asynchronous inputs to the Z80 family CPUs should be externally synchronized with the CPU clock. The required synchronization is specified by the setup and hold times for asynchronous inputs to the CPU. The synchronization is automatically provided for by the Z80 Family peripherals that are capable of driving the asynchronous inputs to the CPU.

In the Z80 CPU Technical Manual (Pages 70 and 72, footnote B), it is stated that "All control signals are internally synchronized so that they are totally asynchronous with respect to the clock." This statement should be amended to say "When interfacing the Z80 CPU to the Z80 family peripherals, the interface control signals are internally synchronized with the system clock by the peripherals themselves. When interfacing to the Z80 CPU with other devices, these control signals should be synchronized with respect to the system clock." Note that the former statement has been removed from the data book and the CPU product specification, but has not been removed from the technical manual yet.

The basis for the synchronization of the input control signals is the potential for the occurrence of a phenomenon called a "meta-stable state". The details of the meta-stable state are complex, but the concept is fairly simple. A meta-stable state occurs in bi-stable logic devices at the interface between an asynchronous and synchronous environment. All two-state logic devices spend some finite amount of time in the "linear region" (between the logic state of one and zero). The length of time spent in the linear region depends upon the switching speed of the device. If a synchronous system samples asynchronous inputs at the precise point in time that it passes through the linear region, the output of the sampling logic may spend time in an undefined logic state (the meta-stable state). The settling time to a valid logic state is proportional to the inverse exponential of the speed of the switching devices. More importantly, if the device in the meta-stable state is connected to several other bi-state devices in the system, the possibility exists for each of these bi-state devices to interpret the non-binary (or meta-stable) input differently. The final result can be an undefined or unpredictable state for a sequential state machine such as the Z80 CPU.

There are several points that should be remembered concerning these asynchronous inputs:

E. V. JARAVIA

1. All interfaces between synchronous and asynchronous system that use clocked bi-stable devices are subject to the "meta-stable" phenomenon.
2. The probability of occurrence of a meta-stable state is directly proportional of the frequency of changes in the state at the interface and inversely proportional to the exponential of the switching speed of the devices used.

Q: How to interface the Z80 CPU to a 8259 using Mode 0 interrupt?

A: The Z80 CPU's interrupt mode "Mode 0" is the mode which maintains the "software compatibility" with the 8080, it is NOT fully compatible.

In this interrupt mode, during INTACK cycle, the Z80 CPU fetches the data on the bus as an "instruction" and executes it, like the 8080. However, from the hardware stand point, it's not true.

The 8080 generates three INTA pulses during the interrupt acknowledge cycle while the Z80 CPU generates only one INTACK signal (which can be decoded from M1 and RD).

This system works fine if you are not using the 8259 and put "RST" (restart) instruction onto the bus during the Interrupt Acknowledge cycle, which is a one byte instruction.

However, if you want to use the 8259 with the Z80 CPU, you'll have a problem. That is:

> The 8259 expects three INTA pulses but the Z80 CPU generates only one INTACK cycle.

The best way to solve the problem is "simulating an 8080 interrupt acknowledge cycle" - which means generating a total of three "INTA" pulse for the 8259 from the Z80 CPU's interrupt acknowledge cycle by external logic. Following figure (Figure 1.) is the one example of the implementation.

This circuit works as follows (Assume that the instruction sent by the 8259 is "CALL" instruction):

On interrupt acknowledge cycle, the decoded INTA signal is sent as an INTA pulse for the 8259 and at the same time sets the LS74 to indicate that an interrupt acknowledge cycle has started.

On the following memory read cycle for the jump address on the call instruction, this circuit generates two additional INTA pulses for the 8259 and also masks off the read signal for the memory to avoid bus contention problems.

On the following write cycle, WR signal resets the LS74 to indicate that the interrupt acknowledge cycle is completed.

By using this circuit, you can use the 8259 with Z80 CPU.

Z80 DMA

Q: Does DMA recognize only 8-bit I/O addresses?

A: The DMA device does not care whether the I/O addresses or memory addresses are 8-bit or 16-bit. The Z80 DMA can address just as many I/O locations as it can memory locations.

Q: What is the importance on the placement of the "LOAD" commands?

A: The "LOAD" command only loads the contents of the source address register into the source address counter. The contents of the destination address register are automatically loaded into the destination address counter the first time the destination address gets incremented or decremented.

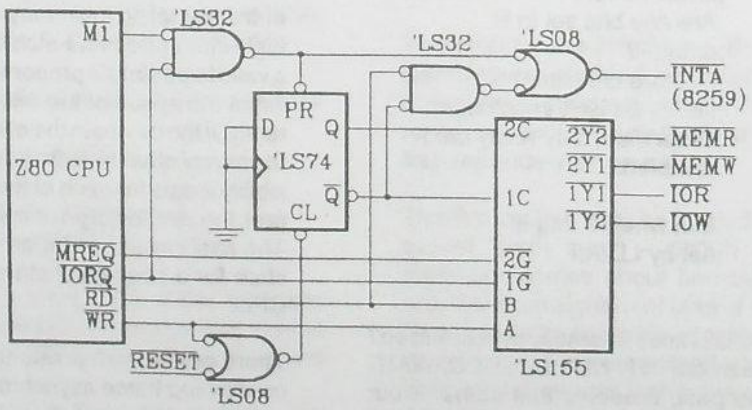

Figure 1. Z80 CPU to 8259 interface example

Q: When using the variable timing modes, are there any constraints in setting up the two ports that the user should be aware of?

A: Yes. When using the early cycle end timing feature of the DMA, it is strongly recommended that both ports be initialized with the same timing constraints.

Q: Is there any way to reset the DMA besides the RESET command and power-down?

A: With the CMOS DMA: On 44-pin PLCC package, there is a newly added "hardware reset pin" on pin 12 (This pin is left open on NMOS PLCC). Also, we've added special functions to the M1 signal line that allows you to reset C-MOS DMA. During an active M1 signal, without an active RD or IORQ, the DMA is reset. This feature is the same as that with Z80 PIO.
With NMOS, the only way to reset the DMA is by reset command. Actually, the RESET command can only reset the DMA if the CPU has control of the bus. if the DMA has control of the bus there is no way to reset it other than powering down the system (or the DMA).

Q: How long does power need to be removed from the DMA for an internal reset to occur?

A: Zilog tests the power-on reset circuit at 10mS. If the user is going to remove power from the DMA, ZIlog recommends that it be done with the CLK input high.

Q: What limitations are not specified in the data book?

A: For NMOS DMA, when using the DMA in BURST mode with 2 cycle timing, an extra transaction is generated at the end of the burst.
For CMOS DMA, we've fixed all limitations.

Q: How can I use the DMA to transfer a page of information but do it one line at a time and wait between lines? (Printer application)

A: Operate the DMA in the Burst mode and use the printer I/O READY line to control DMA. Program the DMA for auto-restart mode to transfer the same "page" area continuously. When the printer is unable to accept a "line", the DMA will allow the CPU to control the bus.

Z80 PIO

Q: When using a port of the PIO in bit mode (mode 3), can any of the bits, programmed as outputs, affect the interrupt conditions set for recognizing inputs?

A: While it is undocumented, it is possible that the state of the bits programmed as outputs could be used as satisfying conditions for the mode 3 interrupt equation. It is recommended that all bits not needed for the interrupts be masked off.

Q: Can the PIO be programmed to provide a 16-bit input port and an 8-bit bidirectional port at the same time?

A: Yes. but there are some major concerns in doing it. Remember that when Port A is programmed into the bidirectional mode (mode 2), the handshake lines from Port B are used as input handshake lines for Port A. Some confusion occurs within the PIO if Port B is also programmed into the input mode (mode 1) and tries to use the handshake lines. A combination of software and hardware can be used to insure that data will not change until both ports can be read.

Q: Is the PIO port protected against hysteresis?

A: No.

Q: Do you have to strobe data into the port for proper mode 1 operation?

A: Yes, if you want to generate interrupts for mode 1 operation. If you only want to read the port data, then the STB input can be held low to make the input data latches transparent.

Q: How can I get Port B interrupt in Mode 3 and Port A interrupt in Mode 2?

A: You can get them, but it can cause severe interrupt conflicts if you choose that option. Port B interrupts are used by Port A in bidirectional mode for receive data interrupts. To prevent interrupt conflicts, Port B interrupts should be enabled, but all bits of Port B should be masked from affecting the interrupts. In the PIO Technical Manual it states that, "the same interrupt vector will be returned for a Mode 3 interrupt on Port B and an input interrupt during Mode 2 operation of Port A" (Section 5.3).

Q: Can the PIO control register be written while the PIO IUS bit is set?

A: Yes. But it is a safer programming practice to program the device after the RETI command.

Q: The on-chip power-on reset does not always work properly. How can I get around this?

A: Use the external hardware reset condition. Activate M1 for a minimum of two clock cycles without activating either RD of IORQ.

Q: When using the PIO in Mode 2, a 55h is written to the port. On the port side, an 0AAh is storobed into the port via BSTB. When the processor reads the data port, the 55h is read back instead of the 0AAh. Why and How?

A: The only way that the system can read the same data that it wrote into the PIO was if the ASTB signal was active (place the 55h onto the port bus) and the BSTB went active to strobe it into the data register. Suggest that system logic inhibit both strobe signals from becoming active during the same time.

E. V. JARAVIA

Q: On which clock edge is the PIO reset (with M1 active and IORQ and RD inactive)?
A: The actual reset function will take place when the M1 signal goes low active (must have been active a minimum of 2 clock cycles).

Q: Can the PIO catch pending interrupts while interrupts are disabled?
A: Yes. Enabling the interrupts allow the interrupt daisy chain to function and the interrupt under service flip-flops to be set.

Q: A question came in concerning how the Z80 PIO handled its interrupts. Is the PIO capable of storing pending interrupts or must an interrupt be serviced and cleared (via either RESET or RETI) before another interrupt can be accepted?
A: It seems that the Z80 PIO interrupt structure is designed so that pending interrupts can be stored. There are to caveats to watch for in this however. The only way to store a pending interrupt is while another one is under service, and only one pending interrupt can be stored. Be aware that if you are operating in Modes 0, 1 or 2, the transition of the STB signal can cause new data to be latched into the input data register and generate a pending interrupt. Be sure that any previous data can be read from the PIO before any new data is strobed in.

The storage for pending interrupts is only one deep. This means that a second interrupt condition cannot be stored if the first one has not been acknowledged.

Q: Does an interrupt mask word have to follow the interrupt control word (assuming bit 4 was set) if the PIO is not programmed for Mode 3 operation?
A: Yes. Follow the interrupt control word with a dummy write to reset the PIO's write control logic.

Q: How can you get two PIOs to talk with each other in Mode 2 operation?
A: Suggest using ARDY1 and BRDY2 to generate a strobe pulse for ASTB1 and BSTB2. Same setup could be used for ARDY2 and BRDY1 and for ASTB2 and BSTB1. The logic basically consists of a 74LS123 (one shot) and a 74LS08 (AND gate). The ARDY1 and BRDY2 signals are ANDed together and supplied as B-TRG for generating ASTB1 and BSTB2. The ARDY2 and BRDY2 signals are ANDed together and supplied as the A-TRG for generating BSTB1 and ASTB2. The A-TRG for generating ASTB1 and BSTB2 is always low (grounded). In this manner, the port control signals are used to set the priority for strobe signal generation.

Please refer to Figure 2 and Table 1.

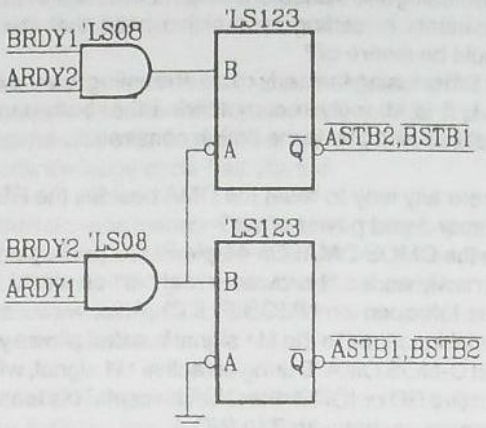

Figure 2. I/F circuit example

ARDY1	BRDY2	BRDY1	ARDY2	ASTB1-BSTB1	ASTB2-BSTB1	Direction
0	0	0	0	1	1	
0	0	0	1	1	1	
0	0	1	1	1	1	
0	0	1	1	1	0	2→1
0	1	0	0	1	1	
0	1	0	1	1	1	
0	1	1	0	1	1	
0	1	1	1	1	0	2→1
1	0	0	0	1	1	
1	0	0	1	1	1	
1	0	1	0	1	1	
1	0	1	1	1	0	2→1
1	1	0	0	0	1	1→2
1	1	0	1	0	1	1→2
1	1	1	0	0	1	1→2
1	1	1	1	0	1	1→2

Table 1. Truth Table for strobe signal generation

Q: How can the PIO be reprogrammed without having pending interrupts locking the system?
A: Try the following procedure.

1. Disable CPU interrupts;
2. Disable interrupt in the PIO;
3. Clear any pending interrupts within the PIO by using the interrupt control word with bit D4 set;
4. Reprogram the PIO as desired; and
5. Re-enable CPU interrupts.

Q: The PIO generates false interrupts during the programming sequence. What can cause this?
A: This symptom is almost always the result of a programming error. Depending upon the details of the problem, there are several solutions.

1. The interrupts should be enabled last in the initialization sequence. The Interrupt Control Word should be written with interrupts disabled so that the logical interrupt equation should be set (Mode 3). Finish the initialization with the Interrupt Enable Control WOrd (83H) to enable the interrupts.

2. The STB and RDY signals should be in a defined state. A transition on the STB input could cause a pending interrupt to be stored and executed as soon as interrupts are enabled.

3. A change in bit pattern (while in Mode 3) may cause an interrupt. A defined state for external inputs is recommended for power-up sequences.

Q: How can I get around the fact that only one "bit set" can be detected at a time in the OR bit mode?
A: One possibility would be to "mask" that bit during the interrupt service routine. Another possibility is to use external hardware to "mask" the bit.

Q: How can I get the PIO to give me interrupts on both transition of an input signal (Mode 3) ?
A: One method to use would require the PIO to be reprogrammed with a different logic equation during the interrupt service routine for the first transition. When the second transition occurs, then a new interrupt can be generated and the logic equation could be set back to its original state. Another method would require the use of external hardware to change the state of interrupting bit. Some possible logic could be to use an output port along with an exclusive-or (XOR) gate to control the state of the input bits.

Z80 CTC

Q: How does the software reset command to the CTC affect the rest of the CTC's operation?
A: The data book and the technical manual differ in the information that is presented on this subject. A software reset command stop the counter from counting any further. In order to start the counter again, a new time constant must be loaded into the time constant register. All bits in a mode control word will cause the operation of the CTC to be affected.

Q: What is the maximum frequency of the counter?
A: If external input is synchronized to the system clock, it's half that of the CLK (system clock) input. If it's not, 1/3 of the system clock.

Q: Are there any other uses for the CTC besides counting and timing?
A: Yes, the CTC makes a very nice interrupt controller for the Z80 bus. By programming the counter for a terminal count of one and defining the transition of the trigger, you can interface non-vectored interrupting devices onto the Z80 bus.

Q: How can I have control over an individual counter so that it cannot be started, stopped, and started again?
A: Use an external gate to qualify the clock input to the counter.

Q: When does the time constant (from the time constant register) get loaded into the down-counter?
A: On the first down count ---- ? verify.

Q: The CTC product specification states that no additional wait states (other than the automatic wait state inserted by the CPU) are allowed in the I/O cycles. Why?
A: It is not that the the wait states aren't allowed, it is just that they don't accomplish anything. The data will arrive at a particular time for the read cycles, and the internal write strobe is generated as a result of the clock edges that will be available. During the read cycle, it is possible that an improper value of the down-counter could be released onto the bus if additional wait states were added (the counter could change in the middle of the read operation).

Z80 SIO

This section contains the most commonly asked questions about the Zilog SIO. They are divided into following groups:

- Features
- Registers
- Interrupt
- Modem control signals
- Enable & Disable Tx & Rx, Auto enable mode
- Questions around DMA
- Internal timings
- External interface
- Asynchronous mode of operation
- Synchronous mode
- Questions about SDLC mode

Features

Q: What is the maximum data rate of the SIO?
A: 1/5 of the system clock rate. So it is 1.6Mb/s max for 8MHz version.

Q: What are the differences between Z80 SIO/0, /1 , /2 and /4?
A: The differences between those four devices is "a combination of Channel B Modem signals". In fact, the SIO die itself has 41 pins internally. But a 40 pin DIP package has only "40 PINs", so we made three kinds of SIO's:

1. Z80 SIO/0: Have all channel B modem signals, exceptTxCB and RxCB, bonded together internally.

E.V. JARAVIA

2. Z80 SIO/1: Lacks "DTRB".
3. Z80 SIO/2: Lacks "SYNCB".

For PLCC packages we are only offering "SIO/4", which covers all, since PLCC has 44 pins to bond out all signals.

Q: What are the differences between the SIO and Z80 DART?
A: The Z80 DART (Dual Asynchronous Receiver/Transmitter) is the device which only supports asynchronous mode of operation. The functionality, internal architecture and AC/DC characteristics are identical to the SIO in asynchronous mode. Also, pin assignment of it is identical to Z80 SIO/0 with the exception of one signal name. The "SYNC" pin on SIO/0 is "RI" (Ring Indicator) on SIO/0, but the functionality is exactly the same as the SIO/0 in asynchronous mode.

Registers

Q: How do you read the status registers?
A: Reads from RR0 (Read Register 0) are accomplished by simply doing a read from the SIO. Reads from RR1 or RR2 are accomplished by writing a register pointer to the SIO (WR0) and then doing a read operation.

Q: What happens when you read an empty FIFO?
A: You will read the last character in the buffer.

Q: How do you avoid an overrun in the receiver FIFO?
A: The receive buffer must be read before the recently received data character on the serial input is shifted into the receive data FIFO. This FIFO is three bytes deep. Thus, if the buffer is not read, the fifth character that just arrived caused an overrun condition. There is no set or reset bit to disable the buffering.

Q: When the FIFO gets locked due to an error condition, can it still receive?
A: The SIO continues to receive until an overrun error occurs.

Q: When does the FIFO buffer lock on an error condition?
A: The receive data FIFO gets locked when the following receiver interrupt modes are selected:

- Receive interrupt on Special condition only.
- Receive interrupt on First character or Special condition.

In both of these modes the special condition interrupt occurs after the character with the special condition has been read. The error status has to be valid when read in the service routine. The special condition locks the FIFO and guarantee that the DMA will not transfer any character until the special condition has been serviced.

Q: When a special condition occurs due to parity error, will a receive interrupt for that byte still be generated?
A: No. In the case of Receive interrupt on Special condition only mode, the interrupt will not occur until after the character with the special condition is read. In the case of Receive interrupt on First character or Special condition mode, the interrupt is generated on every character whether or not it has a special condition.

Q: What is the function of the Error FIFO?
A: The Error FIFO buffers the error conditions status bits for each of the received characters.

Q: When should the status in RR1 be checked?
A: Always read RR1 before reading the data.

Q: What information is contained in the Error FIFO?
A: End of frame, CRC/Framing error, Receive overrun error and Parity error. These are all contained in RR1 as well. The other status offered in RR1 is not part of an Error FIFO.
The Overrun and Parity error bits are held in the FIFO until they are reset by issuing the Error Reset Command. They will not be overwritten by new error information.

Q: How many register pointers does the SIO have?
A: The SIO has one for each channel. So it's possible to set the pointers for each channel first, then accessing each channel's register afterward. But it's not recommended, since program readability gets worse.

Interrupt

Q: What are the various Interrupting conditions?
A: The SIO can generate interrupts from the receiver, Transmitter and External/status for each channel (6 sources). This is a list of all conditions that could possibly generate an interrupt (one channel only listed):

Transmitter:	Transmit Buffer Empty
Receiver:	Receiver Character Available, Parity Error, Framing Error, Receive Overrun Error
External/Status: (Transition on DCD)	CTS, Sync/Hunt, Transmit, Underrun/EOM, Break/Abort Detection

Q: Can the IP bits be set while the SIO is servicing other interrupts?

A: Yes. If the interrupting condition has a higher priority than the interrupt currently being serviced it will cause another interrupt, thus nesting the interrupt service.

Q: How many levels of pending interrupts are there and how does the internal daisy chain operate?

A: Each possible source of an interrupt (6 possible) has one level of pending interrupts. The internal daisy chain operates in the same manner as would an external daisy chain.

Q: Does the RETI Instruction reset any status register?

A: No.

Q: If the CPU does not have the Return From Interrupt sequence (RETI instruction on the Z80 CPU), how may the SIO be informed of the completion of interrupt handling?

A: This may be done by writing the Return From Interrupt command (38h) to WR0 in Channel A of the SIO.

Q: Can the IUS bits be accessed?

A: No.

Q: When do IUS bits get set?

A: The IUS bits will be set during an interrupt acknowledge cycle on the falling edge of RD.

Q: When responding to an Interrupt, can you have the following sequence:

Int Ack, Disable INT, RETI, Clear interrupt condition?

A: No. The correct sequence is : Int Ack, Disable INT., Reset.

Q: Will enabling Interrupt after a transition on the Sync/Hunt bit cause Interrupt to occur?

A: No. External/Status Interrupt should be enabled before the transition occurs.

Note: It is advisable to execute the Reset Ext/Status Interrupt command in advance, so that the status of RR0, bit D4 reflects the current condition.

Q: Why is the Reset/Status Interrupt command recommended to be used several times in SIO setup?

A: Because many of the status bits that reflect interrupting conditions are latched bits and need to be reset to reflect current status rather than what may have occurred due to earlier Interrupts (changes in state).

Q: Will the SIO continue to request interrupt if the condition has not been satisfied?

A: Yes. There are several methods that can be used to clear the interrupt conditions. If it is a transmitter interrupt, then the transmitter must either be loaded with data or the Reset Transmit Interrupt Pending command must be issued. If the interrupt is for External/Status, then the Reset External/Status Interrupt command must be issued. If the interrupt is for a receive character being available, then the receive character must be read. If the interrupt is for an error condition, then the Error Reset command must be given.

Q: What conditions cause the transmit IP to be set?

A: Either the buffer empty or the flag after CRC is being loaded.

Q: How do the external/status bits affect the interrupts?

A: The external/status interrupt structure is affected by bits D7-D3 of RR0. These bits can be "reset" by either a hardware reset, a channel reset, or by the Reset External/Status command. The first status change on any one of the five bits after the reset will cause an interrupt to be issued and also will cause all five status bits to be latched. The latching effect is caused whether or not External/Status interrupts are enabled. If the current status at the time of reset is different than the latched status, then another Interrupt request is generated immediately. To clear the interrupt structure, two resets are necessary. The configuration of the SIO can change the definition of some of these signals. If the state of the bit changes across definition boundaries, an interrupt can be generated. Issue the Reset External/Status Interrupts command after definition. To process an external/Status interrupt, the Reset External/Status Interrupts command must be issued after reading these status bits and before the RETI.

Q: Can you use the SIO without an interrupt acknowledge cycle sequence (Z80 CPU)?

A: Reset the responsible interrupt pending bit (IP). The INT line will follow the IP bit.

Q: If the CPU can be interrupted but cannot be used with vectored interrupts, how should processing be done?

A: Immediately after being interrupted, proceed in a manner similar to polling the SIO for both receive and transmit. Alternatively, the Status affects vector bit (Bit D2 in WR1) may be set and a 0 byte placed into the interrupt vector register(WR2 in channel B). Then, the contents of the interrupt vector register can be used to determine the cause of the interrupt and the channel on which the interrupt occurred. This is queried by reading register RR1 of channel B. Also, IEI is tied high and M1 is tied high. No equivalent to an interrupt acknowledge is issued.

E.V. JARAVIA

Q: When interfacing the SIO to the CPU other than the Z80 CPU, is it possible to assert M1 and IORQ at the same time as the Interrupt acknowledge cycle to simulate Z80 timing?
A: The SIO requires "Internal daisy chain settle time" even if you don't have devices other than the SIO on the interrupt daisy chain. The period for that purpose is "M1 is active but IORQ is inactive", and is at least 100nS (for 4MHz clock ; Parameter # 16, IEI-IEO delay time).

Modem control signals

Q: What is the state of the transmitter output when data is no longer available in the following modes?

a) Asynchronous?
b) Synchronous
c) SDLC?

A: a) In asynchronous modes, the transmitter goes into a marking state whenever all data has been sent.
b) In the synchronous mode, the SIO will send out 16 bits of CRC (2 bytes; if programmed and the Transmitter underrun/EOM Latch has been reset) followed by the appropriate number of Sync character. The line will then continue to idle sync characters.
c) In the SDLC mode, the SIO will send out 16 bits of CRC (2 bytes ; if programmed and the Transmitter underrun/EOM Latch has been reset) followed by the SDLC flag character (7Eh). The line will then continue to idle SDLC flag characters.

Q: What is the delay time for RTS/ to TxD?
A: Two Tx clocks for asynchronous and synchronous, 7 Tx clocks for SDLC.

Q: What is the delay time for the transmit buffer empty to RTS/?
A: Two Tx clocks for asynchronous gate delays for synchronous and SDLC.

Q: Does the frequency of the CTS or DCD signals have any adverse affects on the External/Status Interrupt? (even if auto enable is not programmed)?
A: Since every transition locks the External/Status latches, you could get constant interrupts (if External/Status Interrupt are enabled) or constant status latches.

Q: Is it possible to deactivate the DTR output without reprogramming WR5?
A: Only by resetting the channel or chip.

Q: Can you gate data by stretching the receive clock?
A: You can hold the clock until you have valid data. There are no maximum specs on the RxC period, and the edges are used to sample the data. If there are no edges, no data is sampled.

Enable&Disable Tx&Rx, Auto enable mode

Q: What happens to the character being assembled if the receiver becomes disabled?
A: Assembly of a character stops immediately and the character is lost.

Q: What happens to the characters already in the receive FIFO if the receiver becomes disabled?
A: They will remain in the FIFO until they are either read by the CPU or DMA, or until the channel is reset.

Q: When Auto enable bit is set, will DCD & CTS going true cause an Interrupt?
A: Interrupt will occur only on transition of DCD & CTS since both are edge triggered if WR1,D0 is set for Ext. Int enable.
However, since these are latched conditions in Status Register RR0 (D3 & D5), current status must only read after issuing Reset Ext/Status Interrupt command.

Q: In the auto enable mode, what happens when CTS goes inactive (High) in the middle of transferring a byte?
A: If the Auto Enable mode is selected, the CTS pin is an enable for transmitter (Ideally, Transmitter enable bit is ANDed with the status of CTS). So when CTS is inactive, transmit stops immediately. (The data being shifted out will be sending out completely, however).

Questions around DMA

Q: Can the SIO operate with a DMA in full duplex on each channel?
A: No. The SIO has only one ready line per channel and can only operate in half duplex mode.
If full duplex operation is required under DMA control, both channels A & B need to be used ; One for transmit and one for receive.

Q: Can both channels make simultaneous DMA requests?
A: Yes.

Q: What happens when you program the SIO to interrupt on Buffer Empty and the DMA to act on Buffer Empty?
A: This would not be a wise thing to do. However the Interrupt occurs, the DMA will take over the bus before Interrupt has acknowledged. The buffer will be filled by the DMA and the Interrupt Request will go away due to a Buffer Full condition and the Interrupt Acknowledge will

occur causing bus confusion. The same thing occurs on Receive buffer empty interrupt and DMA on Receive character.

Q: How can the SIO/DMA combination be used for synchronous communications and ensure that the CRC characters are also transmitted?
A: Try the following procedure:

1. Initialize the SIO for use of the READY function with a DMA controller and then poll (or interrupt on) external/status (not transmit buffer empty).
2. Initialize DMA controller for data transfer and bus release at end-of-block. DO NOT ENABLE DMA YET!
3. Send first byte of data to SIO for transmission followed by a Reset Transmit Underrun/EOM Latch command.
4. Enable the DMA controller now (it should take control of the bus).
5. When the end-of-block is reached, the DMA controller should release the bus back to the CPU.

Q: When does the SIO terminate the READY signal?
A: The rising edge of the system clock that samples IORQ low causes READY to go inactive. The delay is specified by parameter 19 in the data sheet.

Q: When does the READY signal become active after an access to the SIO?
A: The READY signal will be inactive for a minimum of 5 clock cycles and will become active again 700 nS after CE goes inactive.

Internal timing

Q: When the transmitter is disabled, when does the TxD line go to a marking state?
A: One bit time after the last bit of the data leaves the transmit shift register.

Q: When the transmitter is empty, does status register RR0, bit D2 indicate that the buffer is now empty or that the last data in the buffer is in the process of being shifted out?
A: It indicates the buffer is now empty. The status register has nothing to do with the transmit shift register.

Q: Does the Transmit interrupt occur when Transmit Buffer is empty or Transmitter itself is empty?
A: Interrupt occurs when the Transmit Buffer is empty.

Q: How many bit times from external clock is the Transmit Buffer Empty Interrupt delayed?
A: The interrupt occurs a maximum of 9 clock periods from the Txc clock edge that causes the buffer to become empty. The exact time is highly dependent upon the mode of operation and is transparent to the user.

Q: When is the data available at the top of the FIFO?
A: Data is available after a maximum of 13 clock periods from the rising edge of RxC.

Q: What is the delay time between transmit shift register to the TxD pin?
A: Two Tx clocks for asynchronous and synchronous. Seven Tx clocks (five for zero inserter, two for internal delay) for SDLC.

Q: Does an Interrupt occur on RxC for last data bit assembled or does it occur relative to the RxC, but delayed?
A: Interrupt occurs when data is moved from the receive shift register to FIFO. The relationship of this event is relative to an external clock edge. This relationship however, is of no concern to the user. There is, however a specific delay from the external clock edge to the interrupt, caused by internal SIO logic.

External interface

Q: Can a sloppy system clock cause problems in SIO operation?
A: Yes. The specs on this system clock are very tight and must be met to prevent SIO malfunction. The specifications are:

Symbol	Description	Min	Max	Unit
VIHC	Clock "H"	Vcc-0.6	5.5	Volt
VIHL	Clock "L"	-0.3	0.45	Volt

Clock rise/Fall time = 30nS each edge
(For N-MOS, 4MHz device).

Should there be any ringing or undershoot/overshoot on the clock input, the SIO could behalf in any number of indeterminable ways.

Q: Must the system clock, fed to the SIO, have a 50% duty cycle?
A: The duty cycle doesn't have to be 50% as long as the minimum specification is met.

Q: Are input control lines to the SIO synchronized to system clocks so that garbage may exist on the buses anytime before setup requirements are satisfied?
A: Yes.

E.V. JARAVIA

Q: Since setup time for CE and IORQ may be satisfied during T2, is time T1 required?
A: If the Z80 CPU is being used, then T1 timing state is required in order to utilize the interrupt structure. (interrupt request, acknowledge, and RETI).
No, if not using the Z80.

Q: Do wait states have to be added to provide I/O response to the SIO in non-Z80 based systems? (The Z80 adds wait states automatically)
A: No. As long as setup times as specified for the SIO are met. The SIO does not know about wait states inserted by the Z80. The Z80 puts in wait states in order to match the Z80 SIO setup times.

Q: What pins are noise sensitive and should be strapped to avoid strange interrupts?
A: The Ext Sync pin, and any Ext status pin that is not used. Also, all inputs are sensitive to signal ringing and undershoot problems.

Q: Is M1 required if no Interrupts are used in the SIO?
A: No. M1 should then be tied high.

Q: Can you use the Ready output for an Interrupt request?
A: Yes, for byte move action in or out and Respond to Interrupt. However, it is not recommended to use Ready for Interrupt with the CPU.

Q: How long must RD and the other control signals remain active?
A: Although RD and IORQ are latched internally, they must remain active for a minimum of two system clock periods.

Q: Are there any timing specifications for "Access recovery time"?
A: No.

Asynchronous Mode

Q: Why are there different Clock factors?
A: These clock factors enable the SIO to sample the center of the data cell. In the X16 mode, the SIO divides the bit cell into 16 counts and samples on count 8.

Q: For asynchronous mode of operation, must the clock rates selected be the same for Receiver and Transmitter?
A: No. However, the multiplier for both RxC & TxC must be the same because of internal logic.

Q: When running in the Async mode, is it necessary to use the X16 clock scalar?
A: No, X1 synchronization can be selected but the user must maintain data synchronization with the clock. The start bit detection logic does not work in X1 mode and the 1.5 stop bit cannot be used in X1. In other words, X1 Mode for Async mode is NOT asynchronous mode, its a "clocked serial channel".

Q: What does the SIO recognize as the Break character ?
A: A character of all zeros including stop bits (indicating a framing error).

Q: When attempting to detect a break condition by sensing the break/abort status bit, is it necessary to enable External/Status interrupts?
A: No. The External/Status latches work regardless of whether or not the external/status interrupts are enabled. This can be confusing because once the latches are strobed, the status in RR0 is frozen until a Reset External/Status Interrupt command is issued. If you desire the true current status, issue this command before reading RR0.

Q: Can a break sequence be sent for a fixed number of character periods?
A: Yes. Break is continuously transmitted as logic 1 by setting bit 4 of WR5. You can then send characters to the transmitter as long as the break level persists. A Break signal rather than the characters sent is transmitted, but each bit of each character sent will be clocked as if it were transmitted. The All sent bit, bit 0 of RR0, is set to 1 when the last bit of a character is clocked for transmission. This may be used to determine when to reset bit 4 of WR5 and stop the Break signal.

Q: If a Break sequence is initiated by setting bit 4 of WR5, will any character in the process of being transmitted, be completed?
A: No. Break is effective immediately when bit 4 of WR5 is set. The "all sent" bit in RR1 should be monitored to determine when it is safe to initiate a Break sequence.

Q: When using the SIO only in Asynchronous mode, can the SYNC pin have any use?
A: It may be used as a general purpose input. For example, by connecting it to a modem ring indicator, the status of that ring indicator can be monitored by the CPU.

Q: How can the SIO be used to transmit characters containing fewer than 5 bits?
A: First, set bit 6 and 5 in WR5 to indicate that five or fewer bits per character will be transmitted. The SIO then determines the number of bits to actually transmit from the data byte itself. The data byte should consist of zero or more 1s, three zeros, and the data to be transmitted. Thus, beginning the data byte with 1111001 will cause only the last bit to be transmitted.

Contents of data bytes(D=arbitrary value)

	D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	1	1	0	0	0	d
2	1	1	1	0	0	0	d	d
3	1	1	0	0	0	d	d	d
4	1	0	0	0	d	d	d	d
5	0	0	0	d	d	d	d	d

Synchronous Mode

Q: Can you cause interrupts on CRC error bit (RR1;D6) changes?

A: No. The CRC error status is not one of the special receive conditions. Perhaps, explanation of cyclic redundancy block checking and how the SIO operates for CRC is relevant.

CRC

Cyclic Redundancy Checking is a method of checking for errors in serial data transmission. It is also known as the polynomial error code check. The polynomial is an algebraic function used to create a constant from the message bit pattern. This constant, generated and accumulated in both the Transmitter and Receiver, is used to divide the binary numeric value of the character. The quotient is discarded and the remainder added to the next character, which again is divided. This continues until the last character, when the remainder is transmitted to the receiver for comparison with the Receiver's remainder. An equal comparison indicated no errors, while an unequal comparison indicates an error in transmission.

SIO-CRC

The SIO contains CRC generation and checking in the Transmitter and Receiver.
It allows for either of two polynomials to be used.

a) $X^{16}+X^{15}+X^2+1$: Called CRC-16, generally used in synchronous communication.

b) $X^{16}+X^{12}+X^5+1$: Called CRC-CCITT, generally used in SDLC communication and also recommended by the CCITT.

CRC Error Check

Status register RR1 which contains error conditions, allocates bit D6 for CRC error status. Since CRC checking is a continuous process and takes place character by character and intermediate results are shifted into the Receive Error FIFO continuously, bit D6 of RR1 is continuously updated because it is not latched.

However, checking the status of this bit at any intermediate point in time in the middle of a transmission is meaningless. It must be remembered that the result of a CRC check is valid only on completion of a message. Also, in most cases, bit D6 will usually be a "1" in the middle of a message since most serial bit combinations result in a non-zero CRC. Unless it is at completion of a message transmission.

The SIO does not generate an Interrupt for CRC Error Status.

Q: Suggest a hardware way to count or determine when the 16 or 20 bit times have passed before the CRC check is valid in BiSync mode?

A: Allow two "buffer full" interrupts to occur to determine that 16 bit times have elapsed, or have an external clock count 20 bit times.

Q: How do you read the CRC error status bit when receiving data in the bisync mode?

A: This is one possible method.
After two CRC bytes have been received and read, wait for the next receive character interrupted and stop CRC accumulation, then read the next received character. After the next character is interrupted, disable the receiver and read the status byte.

Q: In switched carrier Bisync application, the clock may go away before CRC calculate is complete since only one pad will be received. How can valid CRC be ensured?

A: SIO spec requires at least two pads for a valid CRC check in Bisync mode.

Q: Is CRC enabled automatically after first data in a non-SDLC node?

A: Only if it is programmed to be so.

Q: Are Sync patterns (or flags) included in CRC?

A: SDLC - No.
Yes for Bisync - CRC must be turned on/off as required or Sync will be included in CRC.

Q: In synchronous mode, does CRC get stripped from data?

A: Not normally, but it is possible if the CRC byte happens to match the contents of WR6 and the sync character load inhibit feature is enabled.
Otherwise, SIO won't delete the CRC bytes from the data stream.

Q: What is the proper sequence for a valid reading of the CRC error status bit?
A: To check the CRC error status and read the CRC bytes. The following sequence is recommended because of delays in Receive logic and the time at which EOM Interrupt occurs.

1. Interrupt, read and discard - 1st CRC byte.
2. Interrupt, read and discard - 2nd CRC byte.
3. Interrupt, read 1st pad character and discard.
4. Disable CRC
5. Interrupt, read CRC status then read 2nd pad and discard.

Q: In Monosync, is the Sync Comparison done in the Receive shift register or the Sync register?
A: Sync comparison is done in the Sync Register against the contents of WR7.

Q: For Monosync, which register contains the Sync character for comparison?
A: Write Register 7. Comparison is done in the Receive Sync Register.

Q: How does the SIO avoid losing a single sync character in the case of back-to-back Bisync messages that are separated by a single sync pad?
A: It does not. The SIO loses sync characters because the Bisync spec requires a minimum of two pad characters.

Q: Do Sync patterns (or flags) in data get stripped and still cause Interrupts?
A: All leading sync patterns (and all flags) are stripped automatically. In SDLC, sync characters (flags) will cause Interrupts if programmed to. Sync characters may or may not be stripped in Bisync depending on the state of the Sync Character Load Inhibit Bit (WR3,D1). Any data stripped from the data stream cannot cause a receive character available interrupt but may cause other interrupts (such as External/status for Sync/Hunt and special receive condition for EOM). In SDLC, programming Sync Character Load Inhibit will cause stripping of the address field and not cause Interrupts.

Q: Do interrupts occur after each Sync pattern?
A: Yes, if programmed to do so (External/Status interrupts).

Q: Do sync patterns automatically get transmitted in Bisync mode when Transmit Buffer becomes empty?
A: Yes, but the CRC bytes may be allowed to precede those sync characters.

Q: How does the SIO handle synchronous protocols which use less than 8-bit sync characters?
A: The sync character match logic within the SIO only makes comparison on 8-bit boundaries (8-bits for monosync and 16-bits for bisync). In order to match on patterns that are not integer multiples of 8-bits, the sync character must overlay the pattern stored in the sync register, or use "External Sync mode".

Q: Are sync characters subject to parity?
A: No.

Q: Assuming that there are characters available in the FIFO, what happens to them if the receiver goes into the hunt mode?
A: They will remain in the FIFO until they are either read by the CPU or DMA, or until the channel is reset.

Q: Is sync character transmission suppressed when the SIO is programmed for external sync operation?
A: Yes.

Q: How is it possible for the SIO to achieve synchronization on erroneous sync patterns (in monosync and bisync modes)?
A: The design of the SIO is such that the sync register serves as the CRC delay register after synchronization has been achieved. If the SIO goes out of synchronization or is placed into the hunt mode, the CRC delay register again becomes the sync register but it's contents are not cleared. Any data in it can be used by the comparison logic for synchronization. The best solution is to disable the receiver each time you place it in hunt mode and then re-enable it. This sequence will reset the contents of the sync register.

SDLC mode

Q: How does the SIO send CRC?
A: The SIO can be programmed to automatically send the CRC. First, write the first byte of the message to be sent. This guarantees the transmitter is full. Then, reset the Transmit Underrun/EOM latch (WR0;10h). Write the rest of the data frame. When the transmit buffer underruns, the CRC will be sent. The following table describes the action taken by the SIO for the bit oriented protocols.

Tx Underrun	Action	Comment
EOM Latch Bit	Upon Tx Underrun	
0	Send CRC+Flags	Valid Frame
1	Send Flags	Software CRC

Q: In SDLC mode, when do you get the End of Message (EOM) interrupt?
A: The EOF interrupt occurs after the 1st CRC is loaded to the transmit buffer and 2 bit times before the 2nd CRC is loaded to the buffer.

Q: How can you make sure that a flag is transmitted after CRC?

A: Use the external status End of Message (EOM) interrupt to start the CRC transmission, then enable the transmit buffer empty interrupt. When you get the interrupt, it means that the buffer is empty, a flag is loaded in the shift register, and you can send the next packet of information.

Q: When using the SIO in the SDLC mode of operation, the transmitter loses two data bits (gets shifted by two bit positions) when the last character before the closing flag is transmitted. Transmit data is looped to the receiver and CRC is not enabled?

A: The transmitter is working. The receiver actually causes the shift of two bits upon recognition of the closing flag.

Q: Why is the second CRC byte in the receiver FIFO not the full CRC byte?

A: The 2nd CRC byte read from the FIFO is not the full 8 bit byte. The transmitted byte is made up of the last two bits of the 1st CRC byte and the first 6 bits of the 2nd CRC byte. This is because of the delay in the Receive path and the point in time when the EOM Interrupt occurs causing transfer of contents of the Receive Shift Register into the FIFO.

However, since the above 2 bytes are to be discarded by the user, it does not matter.

Except in cases where users may want to include two bytes of data in place of CRC, it is important to note that the last byte will be off by two bits.

Q: In SDLC, when do you reset the CRC generator and checker?

A: The reset Tx CRC generator command should be issued when transmitter is enabled and idling (WR0). This needs to be done only once at initialization time for SDLC mode.

Q: If the SIO is idling flags and a byte of data is loaded into the transmit buffer, what will be transmitted?

A: Data takes priority over flags, and will be loaded into the shift register and transmitted.

Q: What does the SEND ABORT command do to the SDLC transmit sequence?

A: The transmission of the current character is aborted and a sequence of 8-one's are inserted into the data stream. This means that the user may see between 8 and 13 one's in the data stream because of the zero inserter. If there is data in the transmit buffer, it is destroyed and a Transmit Buffer Empty interrupt is pended.

Q: Can the SIO detect multiple aborts?

A: The SIO searches for seven consecutive 1's on the receive data line for the abort detection. This condition may be allowed to cause an external status interrupt. After these seven 1's are received, the receiver automatically enters Hunt mode, where it looks for flags. So even if more than seven 1's are received in case of multiple aborts, only the first sequence of 1's is significant.

Q: In the SDLC mode of operation, what is the relationship between the TxD output and transmitter interrupts?

A: Transmitter interrupts occur when the data from the transmit buffer is loaded into the transmit shift register. The output to the TxD pin is delayed by 6 bit times (five for the zero inserter and one for the pin delay) from the last bit leaving the shift register.

Q: Is it possible to monitor all received characters, including flags, in the SDLC mode?

A: No. if you want to monitor everything, then use the SIO in another mode of operation where the sync pattern has no special meaning.

Q: Can interrupts be generated on the idle line flag in SDLC?

A: Upon receipt of seven continuous ones, the break/abort bit will be set to indicate the abort condition. This bit will remain set until the SIO receives a zero.

Q: Does Hunt in SDLC continue until the Address Field has been recognized?

A: It does if the address search mode feature has been programmed.

Q: Does IBM SDLC specify parity?

A: No.

Q: Can the SIO include parity in SDLC mode?

A: Yes. It is appended at the end of the character.