ZILOG

PRELIMINARY TECHNICAL MANUAL

Z16C30

USC™ UNIVERSAL SERIAL CONTROLLER

Q1/92

ZILOG Z16C30 USC™ UNIVERSAL SERIAL CONTROLLER PRELIMINARY TECHNICAL MANUAL

PRELIMINARY TECHNICAL MANUAL

Z16C30 USC™

UNIVERSAL SERIAL CONTROLLER

INTRODUCTION

The Universal Serial Controller (USC) is the next-generation successor to Zilog's popular SCC family of multi-protocol serial controllers, and is recommended for new designs. Compared to the SCC family and most competing devices, the USC features more serial protocols, a 16- or 8-bit data bus, higher data rates, larger FIFOs, better support for DMA operation, and more convenient software handling. The USC can handle higher data rates because it takes its timing reference from the software-selected receive and transmit clocks and the host bus control signals, rather than a separate "bus clock" or "master clock".

FEATURES

* Two full-duplex multi-protocol serial controllers
* Supports external DMA channels with two Request and two Acknowledge lines
* Serial data rates to 20M bits/second
* 32-character transmit and receive FIFOs for each channel
* 8- or 16-bit transfers for both serial data and registers
* Flexible adaptation to various system buses
* Serial modes include Asynchronous, Bisync, SDLC, HDLC, Ethernet, 1553B, and Nine-Bit
* Two baud rate generators per channel
* Digital phase locked loop for each channel
* Carrier Detect, Clear to Send, and two Serial Clock pins for each channel
* Transmit and receive frame-length counters for each channel
* Async features include false-start filtering, stop bit length programmable by 1/16-bit steps, parity generation/checking, break generation/detection
* HDLC/SDLC features include 8-bit address checking, extended address support, 16/32 bit CRC, programmable idle state, auto preamble option, loop mode
* Sync features include 2 to 16 bit sync pattern, sync-strip option, 16/32 bit CRC, programmable idle state, auto preamble option, X.21 xmit/rcv slaving
* Automatic control character recognition in Transparent Bisync mode
* Flexible interrupt modes including interrupt acknowledge daisy chain
* High speed, low power CMOS technology
* 68 pin PLCC

LOGIC SYMBOL

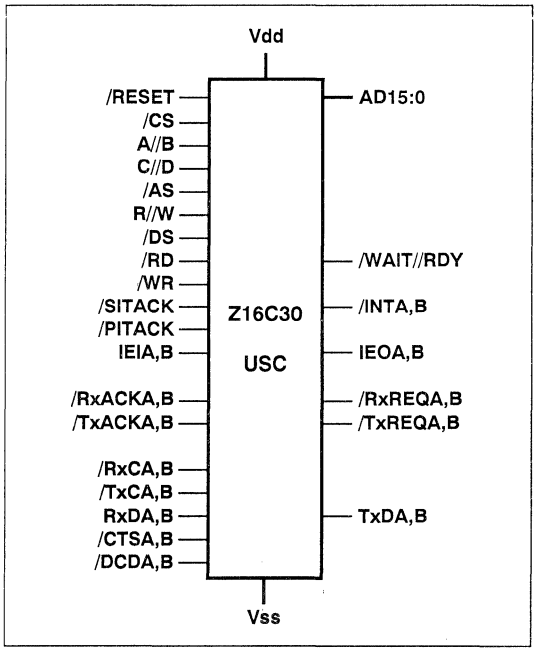
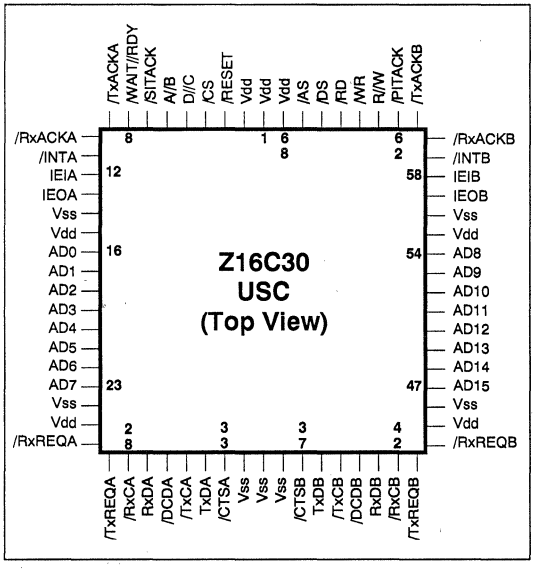


Table of Contents

PACKAGING



PIN DESCRIPTIONS

/RESET. *Reset* (input, active low). A low on this line places the USC in a known, inactive state, and conditions it so that the data, from the next write operation that asserts the /CS pin, goes into the Bus Configuration Register (BCR) regardless of register addressing. /RESET should be driven low as soon as possible during power-up, and as needed when restarting the overall system or the communications subsystem.

AD15-0. *Address/Data Bus* (inputs/3-state outputs). These lines carry data between the controlling microprocessor and the USC, and may also carry multiplexed addresses of registers within the USC. If the USC is used with an external DMA controller, these lines also carry data between the USC and system memory or the DMA controller. AD15-0 can be used in a variety of ways based on whether the USC senses activity on /AS after Reset, and on the data written to the Bus Configuration Register (BCR).

/CS. *Chip Select* (input, active low). A low on this line indicates that the controlling microprocessor's current bus cycle refers to a register in the USC. The USC ignores /CS when a low on /SITACK or /PITACK indicates that the current bus operation is an interrupt acknowledge cycle. On a multiplexed bus the USC latches the state of this pin at rising edges on /AS, while on a non-multiplexed bus it latches /CS at leading/falling edges on /DS, /RD, or /WR.

A//B. *Channel Select* (input, high indicates "channel A"). Cycles with /CS low, and /SITACK, /PITACK, and this pin both high, access registers for channel A. Cycles with /SITACK and /PITACK high, and /CS and this pin both low, access registers for channel B. The state of this line when the Bus Configuration Register is written determines "wait vs. acknowledge" operation, as described in the text. On a multiplexed bus the USC latches the state of this pin at rising edges on /AS, while on a non-multiplexed bus it latches the state at leading/falling edges on /DS, /RD, or /WR.

D//C. *Data/Control* (input, high indicates Data). A read cycle with /CS low, and /SITACK, /PITACK, and this pin high, fetches data from the receive FIFO of the channel selected by A//B, via its Receive Data Register (RDR). A write cycle with the same conditions writes data into that channel's transmit FIFO via its Transmit Data Register (TDR). Cycles with /SITACK and /PITACK high and both /CS and this pin low read or write a USC register. On a multiplexed bus the USC determines which register to access from the low-order AD lines at the rising edge of /AS. On a non-multiplexed bus it typically selects the register based on the LSBits of the serial controller's Channel Command / Address Register. On a multiplexed bus the USC latches the state of this pin at rising edges on /AS, while on a non-multiplexed bus it latches the state at leading/ falling edges on /DS, /RD, or /WR.

/AS. *Address Strobe* (input, active low). After a reset, the USC's bus interface logic monitors this signal to see if the host bus multiplexes address and data on AD15-0. If the logic sees activity on /AS before (or as) software writes the Bus Configuration Register, then in subsequent cycles directed to the USC, it captures register selection from the AD lines, A//B, and C//D on rising edges of /AS.

For a non-multiplexed bus this pin should be pulled up to +5V. If a processor uses a non-multiplexed bus, yet has an output called Address Strobe (e.g., 680x0 devices), this pin should not be tied to the output.

R//W. *Read / Write control* (input, low signifies "write"). R//W and /DS indicate read and write cycles on the bus, for host processors / buses having this kind of signalling. The USC samples R//W at each leading/falling edge on /DS.

/DS. *Data Strobe* (input, active low). R//W and /DS indicate read and write cycles on the bus, for host processors/buses having this kind of signalling. It is qualified by /CS low or /SITACK low. The USC samples R//W at each leading/falling edge on this line. For write cycles, the USC captures data at the rising (trailing) edge on this line. In read cycles the USC provides valid data on the AD lines within the specified access time after this line goes low, and this data remains valid until after the master releases this line to high.

/RD. *Read Strobe* (input, active low). This line indicates a read cycle on the bus, for host processors/ buses having this kind of signalling. It is qualified by /CS low or /SITACK low. In Read cycles the USC provides valid data on the AD lines within the specified access time after this line goes low, and this data remains valid until after the master releases this line to high.

/WR. *Write Strobe* (input, active low). This line indicates write cycles on the bus, for host processors/buses having this kind of signalling. It is qualified by /CS low. The USC captures write data at the rising (trailing) edge of this line.

/WAIT//RDY. *Wait, Ready, or Acknowledge handshaking* (output, active low). This line can carry "wait" or "acknowledge" signalling depending on the state of the A//B input during the initial BCR write. If A//B is high when the BCR is written, this line operates thereafter as a Ready/Wait line for Zilog and most Intel processors. In this mode the USC asserts this line low until it's ready to complete an interrupt acknowledge cycle, but it never asserts this line when the host accesses one of the USC registers.

If A//B is low when the BCR is written, this line operates thereafter as an Acknowledge line for Motorola and some Intel processors. In this mode the USC asserts this line low for register read and write cycles, and also when it is ready to complete an interrupt acknowledge cycle.

In either case this is a full time (totem pole) output. The board designer can combine this signal with similar signals from other slaves, by means of an external logic gate or a 3-state or open-collector driver.

/INTA,B. *Interrupt Requests* (outputs, active low). A channel drives its /INT pin low when (1) its IEI pin is high, (2) one or more of its interrupt condition(s) is (are) enabled and pending, and (3) the Under Service flag isn't set for its highest priority enabled/pending condition, nor for any higher-priority condition within the channel. The USC drives these pins high or low at all times -- they are neither 3-state nor open-drain pins.

/SITACK, /PITACK. *Interrupt Acknowledge* (inputs, active low). A low on one of these lines indicates that the host processor is performing an interrupt acknowledge cycle. In some systems a low on one of these lines may further indicate that external logic has selected this USC as the device to be acknowledged, or as a potential device to be acknowledged. The two signals differ in that /SITACK should be used for a level-sensitive "status" signal that the USC should sample at the leading edge of /AS or /DS, while /PITACK should be used for a single-pulse or double-pulse protocol. The other, unused pin should be pulled up to a high level. A channel will respond to an interrupt acknowledge cycle in a variety of ways depending on its /INT and IEI lines, as described in the text.

IEIA,B. *Interrupt Enable In* (inputs, active high). These signals and the IEO pins can be part of an interrupt-acknowledge daisy-chain with other devices that may request interrupts. If a channel's IEI pin is high **outside of** an interrupt acknowledge cycle, and one or more interrupt condition(s) is(are) enabled and pending for that channel, and the Under Service flag isn't set for the (highest priority such) condition nor for any higher-priority one within the channel, then the channel requests an interrupt by driving its /INT pin low. If a channel's IEI pin is high **during** an IACK cycle, and one or more interrupt condition(s) is(are) enabled and pending in that channel, and the Under Service flag isn't set for the (highest priority such) condition nor for any higher-priority one within the channel, then the channel forces its IEO line low and responds to the cycle.

IEOA,B. *Interrupt Enable Out* (outputs, active high). These signals and the IEI pins can be part of an interrupt acknowledge daisy chain with other devices that may request interrupts. A channel drives its IEO pin low whenever its IEI pin is low, and/or whenever the Under Service flag is set for any condition within the channel. These signals operate slightly differently during an interrupt acknowledge cycle, in that a channel also forces its IEO pin low if it is (has been) requesting an interrupt.

RxDA,B. *Received Data* (inputs, positive logic). The serial inputs for each channel.

TxDA,B. *Transmit Data* (outputs, positive logic). The serial outputs for each channel.

/RxCA,B. *Receive Clock* (inputs or outputs). These signals can be used as a clock input for any of the functional blocks within each channel. Or, software can program a channel so that this pin is an output carrying any of several receiver or internal clock signals, a general purpose input or output, or an interrupt input.

/TxCA,B. *Transmit Clock* (inputs or outputs). These signals can be used as a clock input for any of the functional blocks within each channel. Or, software can program a channel so that this pin is an output carrying any of several transmitter or internal clock signals, a general purpose input or output, or an interrupt input.

/RxREQA,B. *Receive DMA Request* (inputs or outputs). These pins can carry a low-active DMA Request from each channel's receive FIFO. If DMA transfers aren't used for a channel's receiver, its RxREQ pin can be used as a general-purpose input or output or as an interrupt input.

/TxREQA,B. *Transmit DMA Request* (inputs or outputs). These pins can carry a low-active DMA Request from each channel's transmit FIFO. If DMA transfers aren't used for a channel's transmitter, its TxREQ pin can be used as a general-purpose input or output or as an interrupt input.

/RxACKA,B. *Receive DMA Acknowledge* (inputs or outputs). If an external "flyby" DMA controller is being used for a channel's received data, this pin carries the low-active Acknowledge signal from the DMA controller. If DMA transfers aren't used for a channel's receiver, or if the DMA controller uses two-cycle rather than flyby operation, that channel's RxACK pin can be used as a general-purpose input or output.

/TxACKA,B. *Transmit DMA Acknowledge* (inputs or outputs). If an external "flyby" DMA controller is being used for a channel's transmit data, this pin carries the low-active Acknowledge signal from the DMA controller. If DMA transfers aren't used for a channel's receiver, or if the DMA controller uses two-cycle rather than flyby operation, that channel's TxACK pin can be used as a general-purpose input or output.

/DCDA,B. *Data Carrier Detect* (inputs or outputs, active low). Software can program a channel so that this signal enables / disables its receiver. In addition or instead, software can program a channel to request interrupts in response to transitions on this line. The pins can also be used as simple inputs or outputs.

/CTSA,B. *Clear to Send* (inputs or outputs, active low). Software can program a channel so that this signal enables / disables its transmitter. In addition or instead, software can program a channel to request interrupts in response to transitions on this line. The pins can also be used as simple inputs or outputs.

Vcc, Vss. *Power and Ground.* The inclusion of seven pins for each power rail insures good signal integrity, prevents transients on outputs, and improves noise margins on inputs. The USC's internal power distribution network requires that all these pins be connected appropriately.

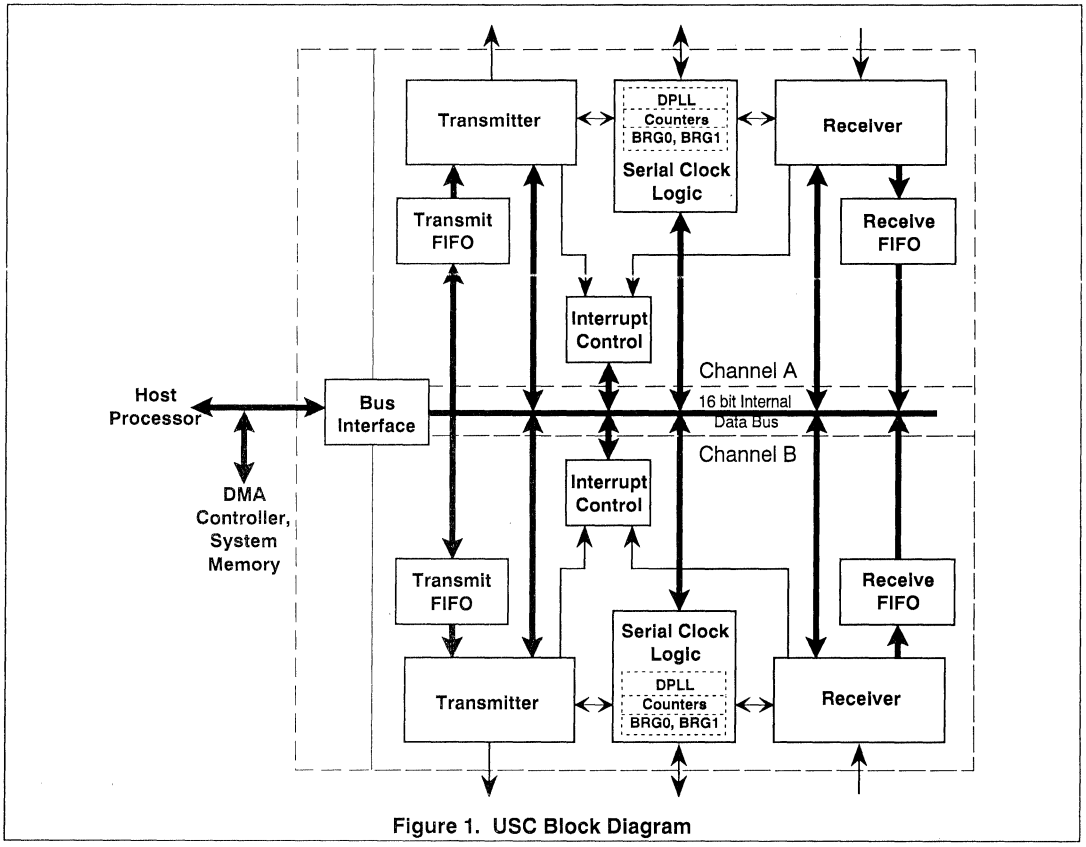


Figure 1. USC Block Diagram

DEVICE STRUCTURE

Figure 1 shows the basic structure of the USC. The Bus Interface module stands between the external bus pins and an on-chip 16-bit data bus that interconnects the other functional modules. It includes several flexible bus interfacing options that are controlled by the Bus Configuration Register (BCR). The BCR is automatically the destination of the first write cycle from the host processor to the USC after a Reset. After that it is no longer accessible to the host software.

Either the host processor or an external DMA channel can write transmit data into a channel's Transmit First-In, First-Out (FIFO) memory. At any time, a Transmit FIFO can be empty or can contain from 1 to 32 characters to be transmitted. Characters written into the FIFO automatically "migrate to its other end", where they become available to the Transmitter.

While the host processor can itself write data into the Transmit FIFOs, it's more efficient to use external Transmit DMA channels to fetch the data. The host can program a USC channel so that its Transmitter will trigger its DMA controller to fill its FIFO at varying degrees of FIFO "emptiness". Selecting this point involves balancing the probability and consequences of "underrunning" the transmitter, against the overhead for the DMA channel to acquire control of the host bus more often.

The serial Transmitters take characters from the Transmit FIFOs and convert them to serial data on the TxD pins. While this function is conceptually simple, the USC supports many complex serial protocols, which increases the complexity of the Transmitters dramatically. Depending on the serial mode selected, the Transmitters may do any of the following in addition to parallel-serial conversion: start, stop, and parity bit generation, calculating and sending CRCs, automatic generation of opening and closing flags, encoding the serial data into any of several formats that guarantee transitions and carry clocking with the data, and/or controlling transmission based on the CTS pin.

In general, the functions of the Receivers are the inverse of those of the Transmitters: they monitor the serial data on the RxD pin, organize it according to the serial mode selected by the software, and convert the data to parallel characters that they place in the Receive FIFOs. Again, there is more to the process than just serial-parallel conversion. Depending on the serial mode the Receivers may have to detect and synchronize start bits, check parity and stop bits, calculate and check CRCs, detect flag, abort and idle sequences, recognize control characters including transparency considerations, decode the serial data and clock extraction using any of several encoding schemes, and/or enable and disable reception based on the DCD input pin. The Receivers' checking functions generate several status bits associated with each character, that accompany the characters through the Receive FIFOs.

The Receive FIFOs can hold up to 32 characters and their associated status bits. As the receivers write entries into their FIFOs, the entries automatically "migrate to the output side" where they become available to either the host processor or external Receive DMA channels. As on the transmit side, the Receive FIFOs include detection logic for various degrees of "fullness". Separate thresholds control the point at which a channel starts requesting its DMA channel starts to refill its FIFO, and at which a channel requests an interrupt. Besides the main Receive FIFOs, each channel has a 4-entry Frame Status FIFO for status related to entire frames rather than individual characters.

While the host processor can access data from the Receive FIFOs, it's more efficient to use external Receive DMA channels to transfer the data directly into buffer areas in memory.

Each channel includes a Serial Clocking Logic section that creates the clocking signals for the channel's Transmitter and Receiver. Software can program the clocking logic to do this in various ways based on one or two external clock(s) for each channel. Each channel also includes a Digital Phase Locked Loop (DPLL) circuit that can recover clocking from encoded data on RxD.

There's also an Interrupt Control section in each channel, that gathers the various "request" lines from the Transmitter and Receiver, and takes care of requesting host interrupts and responding to host interrupt-acknowledge cycles or to software equivalents. Interrupt operation depends on the data written to the Bus Configuration Register and on several registers in the Receiver and Transmitter. There are a separate set of interrupt pins for each channel so that external logic can control their relative priority.

DOCUMENT STRUCTURE

This Chapters in this manual provide the first-time reader with a staged and gradual introduction to the USC. Chapter 2 discusses interfacing the part to typical processor or backplane buses. Chapter 3 discusses how to interface the USC "on the serial side", including the various flexibilities and options available in doing so. Chapter 4 talks about the many serial protocol capabilities of the part; many readers won't be familiar with all the protocols described, but each reader should know the basics of those needed by his or her application. Chapter 5 talks about how to interface external Direct Memory Access (DMA) channels to the USC. Chapter 6 deals with interrupts. Finally, Chapter 7 pulls together certain aspects of writing software for the USC and serves as a central programming reference.

This manual is structured according to the USC's major internal blocks and various aspects of their operation, rather than as a list and description of each of its registers. The various registers and fields are covered in conjunction with the facilities that they report on and control. Chapter 7 then reviews the general programming model and includes a concise description of each register bit and field for quick reference.

The actual timing parameters and electrical specifications of the IUSC are given in the companion publication *USC Product Specification.*

We at Zilog hope that this newly structured manual will make the USC more easily understandable and accessible. Naturally, it's impossible to write at the right level for all readers; newcomers will find some parts hard going, while experts will undoubtedly tire of full explanations of matters that "everyone knows". Our target audience is neither newcomers nor experts, but midway between: working engineers with some datacom background.

About Test Modes

Each USC channel includes a Test Mode Control Register (TMCR) and a Test Mode Data Register (TMDR) that Zilog uses to help test the device and ensure that customers receive only fully functional units. In some cases these registers might be useful to help hardware and software developers solve a knotty problem. On the other hand, this manual is big enough without including subjects of use to only a fraction of its readers. If you are interested in using the test modes, contact a Zilog sales office for the forthcoming volume *USC Family Test Modes.*

2. Bus Interfacing

The USC can be used in systems with various microprocessor or backplane buses. Its flexibility with respect to host bus interfacing derives from its Bus Configuration Register (BCR), from on-chip logic that monitors bus activity before software writes the BCR, and from certain other registers in the serial channels. This section describes how to use these facilities to interface the USC to a variety of host microprocessors and buses.

Multiplexed / Non-Multiplexed Operation

One important distinction among buses is whether they include separate sets of lines for addresses and for data, or whether the same set of lines carries both addresses and data. On a multiplexed bus, the USC captures addressing at rising edges on /AS. If this signalling is the same as that used on the host bus (as with a Zilog 16C0x), then the USC's /AS pin can be directly connected to the corresponding bus signal. Figure 2 shows such a system.

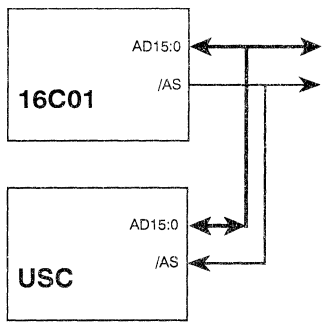


Figure 2. Simple Multiplexed System

An 80x86-based system differs only in that the processor's ALE signal has to be inverted to produce /AS for the USC.

Figures 3 and 4 illustrate two ways to interface the USC to a non-multiplexed host bus. Figure 3 includes minimum hardware but requires the software to write the register address into the USC each time it is going to access a register. In this mode the USC's /AS pin should be pulled up to ensure a constant high logic level. Figure 4 includes drivers to sequence the low-order bits of the host address onto the USC's AD lines, and logic to synthesize a pulse on the /AS pin. This interfacing method has the advantage that software can directly address the USC's registers.

The USC monitors the /AS pin from the time the /RESET pin goes high until the software writes the Bus Configuration Register. If it sees /AS go low at any point in this period, then after the software writes the BCR, the USC captures the state of the low-order AD lines, A//B, C//D, and /CS, at each rising edge of /AS. If /AS remains high, software may have to write each register address into the Channel Command/ Address Register (CCAR) before reading or writing a register. (If the host bus only includes 8 data lines, AD13-8 can carry register addresses.)

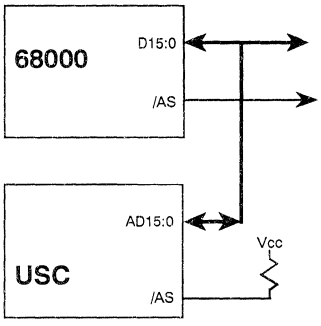


Figure 3. Simple Interface to Non-Multiplexed Bus

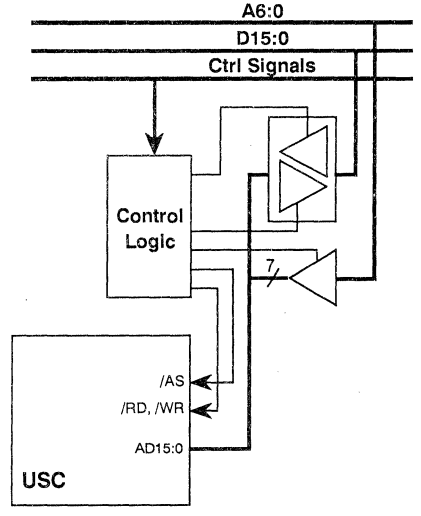


Figure 4. User-Friendly Interface to Non-Multiplexed Bus

Read/Write Data Strobes

Another difference among host buses is the way that read and write cycles are signalled and differentiated. Figures 5 and 6 show two standard methods supported by the USC. In Figure 5, the bus includes separate strobe lines for read and write cycles, commonly called /RD and /WR. In Figure 6, the bus includes a data strobe line, /DS, that goes low for both read and write cycles, and a R//W line that differentiates read cycles from writes. The USC includes pins for all four of these signals. The two that match up with host bus signals should be connected to those signals. The two unused pins should be pulled up to a high level.

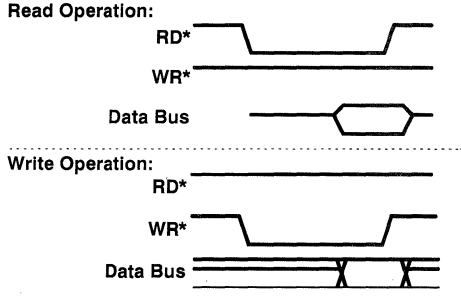


Figure 5. /RD and /WR Signalling

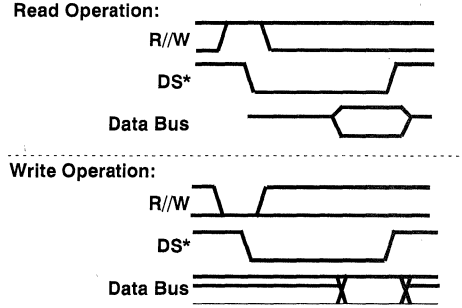


Figure 6. R//W and /DS Signalling

There is no programmable option for the distinction between /RD-/WR and R//W-/DS operation. The USC simply responds to either pair of lines, which is why it's important to pull up the unused pair. Also, the USC doesn't demand that the R//W line remain valid throughout the assertion of /DS. It captures the state of R//W at the leading/falling edge of /DS, so that R//W need only satisfy setup and hold times with respect to this edge.

Only one among the bus signals /DS, /RD, /WR, and /PITACK may be active at a time. This prohibition also includes /RxACKA, /RxACKB, /TxACKA, and /TxACKB when these pins are used as DMA acknowledge signals. (Chapter 5 covers DMA interfacing including the "ACK" signals, and Chapter 6 describes the USC's interrupt features including /PITACK). If the USC detects more than one of these inputs active simultaneously, it enters an inactive state from which the only escape is via the /RESET pin.

Bus Width

Another major difference among host buses is the number of data bits that can be transferred in one cycle. Software can configure the USC to transfer 16 bits at a time, in which case it is still possible to transfer 8 bits when this is necessary or desirable. Or, software can restrict operation to transferring only 8 bits at a time, on the AD7-0 pins. This leaves the AD15-8 pins unused: another BCR option allows them to carry register addresses. The latter option allows software to directly address USC registers even on a non-multiplexed bus, without having to write an address into the USC before it accesses a register.

ACK vs. WAIT Handshaking

The final major difference among host buses involves the nature of the handshaking signals that slave devices use for speed-matching with masters. Figure 7 illustrates the three variations in common use. In the first, which we'll call Wait signalling, if a master selects a slave and the slave cannot capture write data or provide read data within the time allowed to keep the master operating at full speed, it quickly (combinatorially) drives a Wait output low, and then returns it to high when it's ready to complete the cycle. Slave Wait outputs that are open-collector or open-drain can be tied together for a negative logic wired-Or function, and/or a logic gate can be used to negative-logic OR (positive-logic AND) separate Wait lines to produce the /WAIT input to the master (e.g., to the processor).

In the second scheme, "Acknowledge" signalling, all slaves must respond when the master directs a cycle to them, by driving an Acknowledge signal (sometimes called /DTACK) low to allow the master to complete the transfer, and keeping it low until the master does so. As with the previous scheme, slave Ack outputs that are open-collector or open-drain can be tied together for a negative logic wired-Or function, and/or a logic gate can be used to negative-logic OR separate Ack lines to produce the Acknowledge input to the master.

In the third scheme, "Ready" signalling, all slaves must respond when the master directs a cycle to them, by driving a Ready signal high to allow the master to complete the transfer, and keeping it high until the master does so. This scheme differ from Wait signalling in the default state of the handshaking signal between cycles (high for Wait signalling, low for Ready). It has similar timing as Ack signalling, but differs in the polarity of the handshaking signal. With Ready signalling, the board designer must include a logic gate to positive-logic OR the various slaves' Ready lines to produce a composite Ready input for the bus master(s).

The USC supports Acknowledge and Ready signalling for all cycles, and Wait signalling for interrupt acknowledge cycles. The USC register access times should be short enough to avoid the need for Wait signalling on all but the fastest processors. The board designer can combine the USC's /WAIT//RDY output with similar signals from other slaves, by means of an external logic gate or (for Acknowledge and Wait) an external 3-state or open-collector driver.

The next section describes how software can select which way the USC drives its /WAIT//RDY pin, depending on the address at which it writes the Bus Configuration Register (BCR).

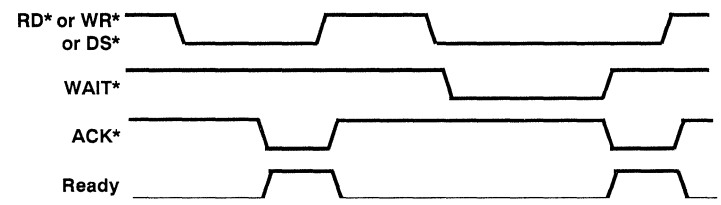


Figure 7. A Fast and Slow Cycle, with Three Kinds of Handshaking

The Bus Configuration Register (BCR)

The BCR is a 16-bit register having the format shown in Figure 8. All the bits in the BCR reset to zero. If the host processor handles 16-bit data, and the data bus between it and the USC is at least 16 bits wide, then the software's initial access to the USC should be a 16-bit write. This write can be to any address that activates the /CS pin; the data will be placed in the BCR. If the host can only write bytes to the USC, all data should be transferred on the AD7-0 pins. In such a system, pull-down resistors should be attached to the AD15-8 pins to ensure the state of these lines during the BCR write. (AD15 may want to be pulled up instead of down, as described in the section on the SepAd bit below.)

The following paragraphs describe the significance of the various bits and fields in the BCR. Besides these data bits, the USC captures the state of the A//B pin when the software writes the BCR. It uses this captured state after the BCR write, such that if A//B was low, it drives the /WAIT//RDY pin as an "acknowledge" (or inverted "ready") signal during register accesses and interrupt acknowledge cycles, while if A//B was high, it drives the pin as a "wait" signal during interrupt acknowledge cycles only. Therefore, software should program the BCR at an address that corresponds to the kind of slave-to-master handshaking used on the host bus.

SepAd (Separate Address; BCR15): this bit should only be written as 1 with 16Bit=0. This combination conditions the USC to use AD7-0 for data and to take register addressing from AD13-8. In this mode the USC takes the Upper/Lower byte indication (U//L) from AD8 and the register address from AD13-9.

With this interfacing technique, the BCR must be written at an address such that AD13-8 are low/zero. Further, AD15 must be high/one and AD14 must be low/zero when software writes the BCR. The designer can ensure this by connecting AD15 and AD14 to more-significant address lines and writing the BCR at an appropriate address. Alternatively, the designer can ensure this by connecting a pull-up resistor to AD15 and a pulldown resistor to AD14.

This mode is useful with a non-multiplexed bus, to avoid making the software write a register address to CCAR before each register access. In this mode the USC captures the state of AD13-8 on each leading/falling edge on /DS, /RD, or /WR. But software can still program SepAd=1 (with 16Bit=0) when the USC has detected early activity on /AS. In this case the USC captures addressing from AD13-8 on each rising edge of /AS, rather than from the low-order AD lines as would be true with SepAd=0.

SepAd	Reserved												16Bit	2Pulse IACK	SRight A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 7. The USC's Bus Configuration Register (BCR)

16Bit (BCR2)**:** this bit should be written as 1 when the host data bus is 16 bits wide (or wider). Writing this bit as 0 has two effects: it restricts the host to using byte transfers on AD7-0 when reading and writing the USC's registers, and it makes the USC ignore the state of the "B//W" signal or bit for register accesses. This bit also controls whether "implicit" accesses to the CCAR, TDR, and RDR are 8 or 16 bit wide.

2PulseIACK (Double-Pulse Interrupt Acknowledge; BCR1): software should program this bit to 0 if the /PITACK pin isn't used or if it carries a single pulse when the host processor acknowledges an interrupt, or to 1 if /PITACK carries two pulses when the host processor acknowledges an interrupt. (The latter mode is compatible with certain Intel processors.)

SRightA (Shift Right Addresses; BCR0): this bit is significant only for a multiplexed bus -- the USC ignores it for a non-multiplexed bus. If SRightA is 1, the USC captures register addressing from the AD6-0 pins and ignores the AD7 pin. In this mode, AD0 carries the Upper/Lower byte indication (U//L), AD5-1 carry the actual register address, and AD6 carries the Byte/Word indication (B//W). If SRightA is 0, the USC captures addressing from AD7-1 and ignores AD0. It takes U//L from AD1, the register address from AD6-2, and B//W from AD7. This bit has no effect on the use of the S//D and D//C pins.

SRightA would be 0 when using the USC as an 8-bit peripheral on a 16-bit bus, which isn't likely to be a common application. Some sections of this manual assume that SRightA is 1.

All other bits in the BCR are reserved and should be programmed as 0. If the processor can only write bytes to the USC, software can only write the 8 LSBits of the BCR, on the AD7-0 lines. In this case, the state of AD15-8, when software writes the BCR, must be ensured by connecting these pins to pulldown resistors, or, if SepAd=1, to host address lines.

Register Addressing

Tables 1 and 2 show the names and addresses of the addressable registers in the USC, in address and alphabetical order. As already noted, the device can take register addresses from any of several sources:

(1) from the AD6-0 lines as latched at the rising edge of /AS, assuming SRightA (BCR0) is 1,
(2) from the AD13-8 lines as latched at the rising edge of /AS, /DS, /RD, or /WR, and/or
(3) from the least significant 7 bits of the Channel Command/Address Register (CCAR), namely the B//W, RegAddr, and U//L bits/fields. (Figure 9 shows the CCAR.)

The Tables assume that SRightA (BCR0) is 1. The RegAddr column in the Tables reflects the state of AD5-1, AD13-9, or CCAR5-1 as applicable.

If "16Bit" (BCR2) is 1, the state of AD6, AD14, or CCAR6 selects between a 16-bit transfer (if 0/low) and an 8-bit transfer (if 1). If "16Bit" is 0, the USC ignores AD6, AD14, or CCAR6 (as applicable). Note that the values in the "8-bit data" columns of Tables 1 and 2 include the B//W bit 1 for both direct and indirect addressing, as is required on a 16-bit bus. When 16Bit (BCR2) is 0 these address values can be used as shown, or 64 lower like the addresses shown in the "16-bit data" columns.

For 8-bit transfers on either an 8- or 16-bit bus, the state of AD0, AD8, or CCAR0 selects the less-significant 8 bits of the register (if 0/low) or the more-significant 8 bits if 1/high. In this regard the USC is "little Endian" like Intel microprocessors. For 16-bit transfers, AD0, AD8, or CCAR0 must be 0/low.

The Direct Address columns of the Tables assume:

(1) SRightA (BCR0) is 1,
(2) the processor's multiplexed AD6-0 lines are connected to AD6-0, or its A5-0 lines are connected to AD13-8, depending on SepAd (BCR15),
(3) the D//C pin is grounded, and
(4) the processor's A7 line is connected to A//B.

If your design differs from these assumptions, register addressing will be different from that shown in the Direct Address columns.

RTCmd					RT Reset	RTMode		Chan Load	B//W	RegAddr					U//L
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 9. The Channel Command/Address Register (CCAR)

The USC provides certain "implicit addressing" features that are intended mainly to make indirect addressing more convenient for host software. Two notes indicated in the Tables relate to these features:

(Note 1): If no other source of addressing applies, that is, if the USC considers the bus non-multiplexed because:

1) it saw no activity on /AS after Reset,

2) the SepAd bit (BCR15) is 0,

3) D//C is low, and

4) CCAR5-0 are all zero,

then the USC assumes a reference to CCAR. If 16Bit (BCR2) is 1, it assumes a 16-bit access, while 16Bit=0 it assumes an access to CCAR7-0.

(Note 2): If D//C is high for a write operation, the USC assumes a write to the Transmit Data Register (TDR), while if D//C is high for a read, it provides data from the Receive Data Register (RDR). For both Reads and Writes, if 16Bit (BCR2) is 1 the USC assumes a 16 bit access, while if 16Bit=0 it assumes an access to the less-significant byte.

(For a 16-bit bus, this means that software can neither write a byte to the TDR/TxFIFO nor read a byte from the RDR/RxFIFO using an address that makes D//C high. Instead, software must provide the explicit address of the LSbyte of the TDR/RDR, either directly or by writing it to the CCAR.

Register Name	Acronym	Reg Addr	CCAR6-0 Indirect Address or Channel B Direct Address		Channel A Direct Address:	
			16-bit data	8-bit data	16-bit data	8-bit data
Channel Command / Address	CCAR	00000	0 / 0 (note 1)	64,65 / 40,1 (note 1)	128 / 80 (note 1)	192,3 / C0,1 (note 1)
Channel Mode	CMR	00001	2 / 2	66,7 / 42,3	130 / 82	194.5 / C2,3
Channel Command / Status	CCSR	00010	4 / 4	68,9 / 44,5	132 / 84	196,7 / C4,5
Channel Control	CCR	00011	6 / 6	70,1 / 46,7	134 / 86	198,9 / C6,7
Test Mode Data	TMDR	00110	12 / 0C	76,7 / 4C,D	140 / 8C	204,5 / CC,D
Test Mode Control	TMCR	00111	14 / 0E	78,9 / 4E,F	142 / 8E	206,7 / CE,F
Clock Mode Control	CMCR	01000	16 / 10	80,1 / 50,1	144 / 90	208,9 / D0,1
Hardware Configuration	HCR	01001	18 / 12	82,3 / 52,3	146 / 92	210,1 / D2,3
Interrupt Vector	IVR	01010	20 / 14	84,5 / 54,5	148 / 94	212,3 / D4,5
Input / Output Control	IOCR	01011	22 / 16	86,7 / 56,7	150 / 96	214,5 / D6,7
Interrupt Control	ICR	01100	24 / 18	88,9 / 58,9	152 / 98	216,7 / D8.9
Daisy-Chain Control	DCCR	01101	26 / 1A	90,1 / 5A,B	154 / 9A	218,9 / DA,B
Miscellaneous Interrupt Status	MISR	01110	28 / 1C	92,3 / 5C,D	156 / 9C	220,1 / DC,D
Status Interrupt Control	SICR	01111	30 / 1E	94,5 / 5E,F	158 / 9E	222,3 / DE,F
Receive Data (Read only; TDR for Write)	RDR	1x000	32 / 20 (note 2)	96/60 or 97/61 (note 2)	160 / A0 (note 2)	224/E0 or 225/E1 (note 2)
Receive Mode	RMR	10001	34 / 22	98,9 / 62,3	162 / A2	226,7 / E2,3
Receive Command / Status	RCSR	10010	36 / 24	100,1 / 64,5	164 / A4	228,9 / E4,5
Receive Interrupt Control	RICR	10011	38 / 26	102,3 / 66,7	166 / A6	230,1 / E6,7
Receive Sync	RSR	10100	40 / 28	104,5 / 68,9	168 / A8	232,3 / E8,9
Receive Count Limit	RCLR	10101	42 / 2A	106,7 / 6A,B	170 / AA	234,5 / EA,B
Receive Character Count	RCCR	10110	44 / 2C	108,9 / 6C,D	172 / AC	236,7 / EC,D
Time Constant 0	TC0R	10111	46 / 2E	110,1 / 6E,F	174 / AE	238.9 / EE,F
Transmit Data (Write only; RDR for Read)	TDR	1x000	48 / 30 (note 2)	112/70 or 113/71 (note 2)	176 / B0 (note 2)	240/F0 or 241/F1 (note 2)
Transmit Mode	TMR	11001	50 / 32	114,5 / 72,3	178 / B2	242,3 / F2,3
Transmit Command / Status	TCSR	11010	52 / 34	116,7 / 74,5	180 / B4	244,5 / F4,5
Transmit Interrupt Control	TICR	11011	54 / 36	118,9 / 76,7	182 / B6	246,7 / F6,7
Transmit Sync	TSR	11100	56 / 38	120,1 / 78,9	184 / B8	248,9 / F8,9
Transmit Count Limit	TCLR	11101	58 / 3A	122,3 / 7A,B	186 / BA	250,1 / FA,B
Transmit Character Count	TCCR	11110	60 / 3C	124,5 / 7C,D	188 / BC	252,3 / FC,D
Time Constant 1	TC1R	11111	62 / 3E	126,7 / 7E,F	190 / BE	254,5 / FE,F

Table 1. USC Registers, In address order

Register Name	Acronym	Reg Addr	CCAR6-0 Indirect Address or Channel B Direct Address		Channel A Direct Address:	
			16-bit data	8-bit data	16-bit data	8-bit data
Channel Command / Address	CCAR	00000	0 / 0 (note 1)	64,65 / 40,1 (note 1)	128 / 80 (note 1)	192,3 / C0,1 (note 1)
Channel Command / Status	CCSR	00010	4 / 4	68,9 / 44,5	132 / 84	196,7 / C4,5
Channel Control	CCR	00011	6 / 6	70,1 / 46,7	134 / 86	198,9 / C6,7
Channel Mode	CMR	00001	2 / 2	66,7 / 42,3	130 / 82	194.5 / C2,3
Clock Mode Control	CMCR	01000	16 / 10	80,1 / 50,1	144 / 90	208,9 / D0,1
Daisy-Chain Control	DCCR	01101	26 / 1A	90,1 / 5A,B	154 / 9A	218,9 / DA,B
Hardware Configuration	HCR	01001	18 / 12	82,3 / 52,3	146 / 92	210,1 / D2,3
Input / Output Control	IOCR	01011	22 / 16	86,7 / 56,7	150 / 96	214,5 / D6,7
Interrupt Control	ICR	01100	24 / 18	88,9 / 58,9	152 / 98	216,7 / D8.9
Interrupt Vector	IVR	01010	20 / 14	84,5 / 54,5	148 / 94	212,3 / D4,5
Miscellaneous Interrupt Status	MISR	01110	28 / 1C	92,3 / 5C,D	156 / 9C	220,1 / DC,D
Receive Character Count	RCCR	10110	44 / 2C	108,9 / 6C,D	172 / AC	236,7 / EC,D
Receive Command / Status	RCSR	10010	36 / 24	100,1 / 64,5	164 / A4	228,9 / E4,5
Receive Count Limit	RCLR	10101	42 / 2A	106,7 / 6A,B	170 / AA	234,5 / EA,B
Receive Data (Read only; TDR for Write)	RDR	1x000	32 / 20 (note 2)	96/60 or 97/61 (note 2)	160 / A0 (note 2)	224/E0 or 225/E1 (note 2)
Receive Interrupt Control	RICR	10011	38 / 26	102,3 / 66,7	166 / A6	230,1 / E6,7
Receive Mode	RMR	10001	34 / 22	98,9 / 62,3	162 / A2	226,7 / E2,3
Receive Sync	RSR	10100	40 / 28	104,5 / 68,9	168 / A8	232,3 / E8,9
Status Interrupt Control	SICR	01111	30 / 1E	94,5 / 5E,F	158 / 9E	222,3 / DE,F
Test Mode Control	TMCR	00111	14 / 0E	78,9 / 4E,F	142 / 8E	206,7 / CE,F
Test Mode Data	TMDR	00110	12 / 0C	76,7 / 4C,D	140 / 8C	204,5 / CC,D
Time Constant 0	TC0R	10111	46 / 2E	110,1 / 6E,F	174 / AE	238.9 / EE,F
Time Constant 1	TC1R	11111	62 / 3E	126,7 / 7E,F	190 / BE	254,5 / FE,F
Transmit Character Count	TCCR	11110	60 / 3C	124,5 / 7C,D	188 / BC	252,3 / FC,D
Transmit Command / Status	TCSR	11010	52 / 34	116,7 / 74,5	180 / B4	244,5 / F4,5
Transmit Count Limit	TCLR	11101	58 / 3A	122,3 / 7A,B	186 / BA	250,1 / FA,B
Transmit Data (Write only; RDR for Read)	TDR	1x000	48 / 30 (note 2)	112/70 or 113/71 (note 2)	176 / B0 (note 2)	240/F0 or 241/F1 (note 2)
Transmit Interrupt Control	TICR	11011	54 / 36	118,9 / 76,7	182 / B6	246,7 / F6,7
Transmit Mode	TMR	11001	50 / 32	114,5 / 72,3	178 / B2	242,3 / F2,3
Transmit Sync	TSR	11100	56 / 38	120,1 / 78,9	184 / B8	248,9 / F8,9

Table 2. USC Registers, in alphabetical order

The RDR and TDR have certain other special characteristics:

1. They are actually "the read and write sides of" the same register location. The USC ignores the state of AD4, AD12, or CCAR4 (as applicable) whenver the rest of the address indicates an access to TDR or RDR. For simplicity Tables 1 and 2 show RDR at the lower address and TDR at the higher one.

2. The MSBytes of RDR and TDR should never be read or written alone, only as part of a 16-bit access. On a Zilog 16C0x or Motorola 680x0 system, use direct addresses 97 or 113 (61 or 71 hex) for channel B, and 225 or 241 (E1 or F1 hex) for channel A, to select the LSByte for byte transfers. On an Intel-based system, use the addresses 96, 112, 224, or 240 (60, 70, E0, F0 hex) correspondingly, to select the LSByte for byte transfers.

The direct, indirect, and implicit addressing features of the USC interact in several ways. For example, CCAR can always be used to select a register for a subsequent access to the CCAR address. This is true whether or not the USC detected activity on /AS after Reset, and regardless of the state of SepAd (BCR15).

The flowchart of Figure 10 shows the complete process by which the USC determines which register to access when a host processor cycle asserts /CS and one of /RD, /WR, or /DS.

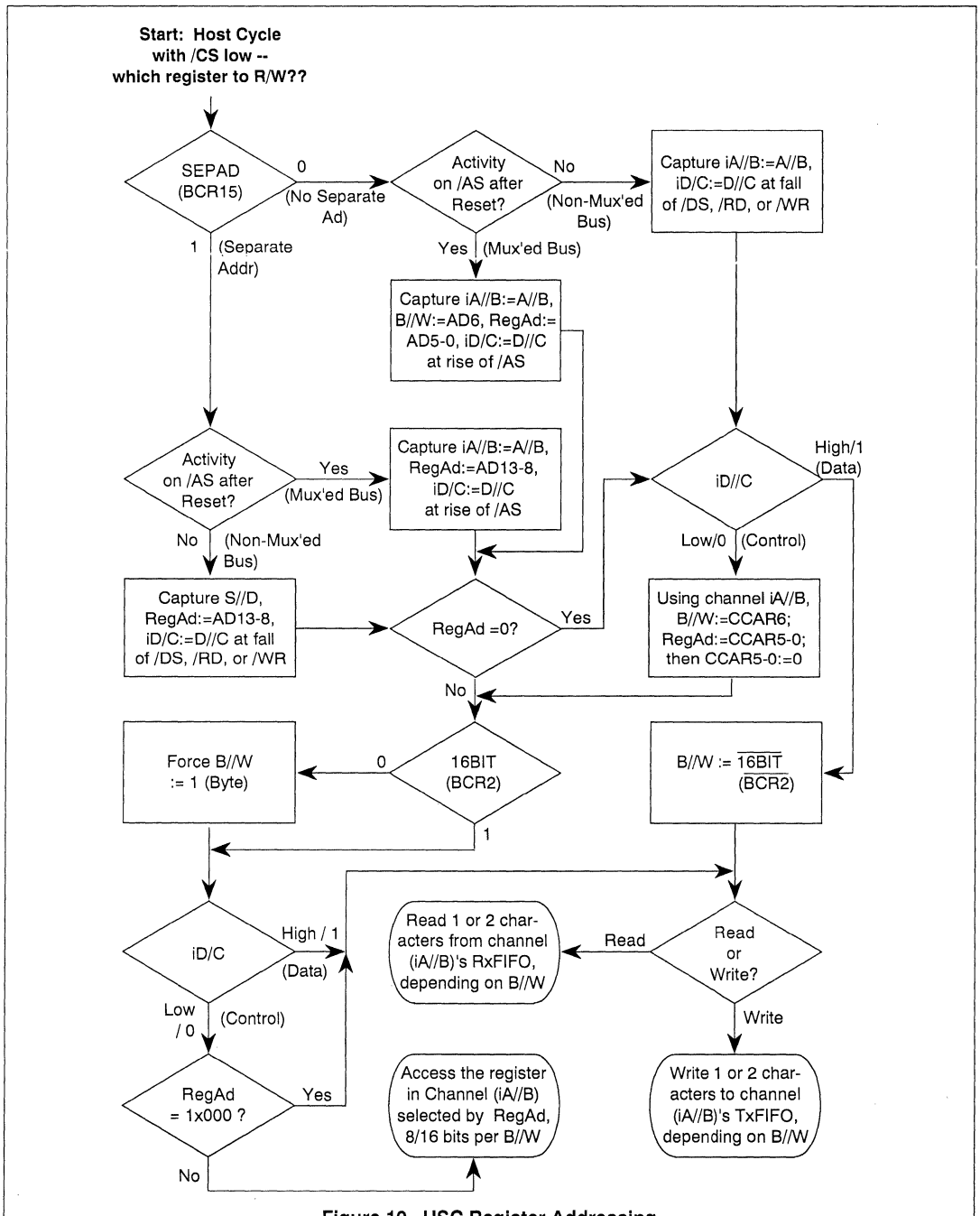


Figure 10. USC Register Addressing

Byte Ordering

Various microprocessors differ on the correspondence between byte addresses and how bytes are arranged within a 16- or 32-bit value. The Zilog Z80 and most Intel processors use what's sometimes called the "Little-Endian" convention: the least significant byte of a word has the smallest address, and the most significant byte has the largest address. The Zilog 16C0x and Motorola 680x0 processors are "Big-Endian": they store and fetch the MSByte in the lowest-addressed byte, and the LSByte from the highest address.

Two commands in the RTCmd field of the Channel Command / Address Register (CCAR15-11) allow the USC to be used with either kind of processor. The "Select D15-8 First" and "Select D7-0 First" commands control the byte ordering within a 16-bit transfer of serial data, and apply to DMA and processor accesses to RDR and TDR.

Register Read and Write Cycles

Figures 11 through 14 show the waveforms of the signals involved when the host processor reads or writes a USC register. Separate drawings are included for the signalling on a bus with multiplexed addresses and data, and for a bus with separate address and data lines. On the other hand, since waveforms get pretty boring after the first few, several things have been done to minimize the number of figures.

1. The cases of separate read and write strobes, vs. a direction line and a common data strobe, have been combined by labelling the strobe traces as "/DS or /RD" and "/DS or /WR". The direction line R//W is shown in the figures, but a note reminds the reader that its state doesn't matter with /RD and /WR.

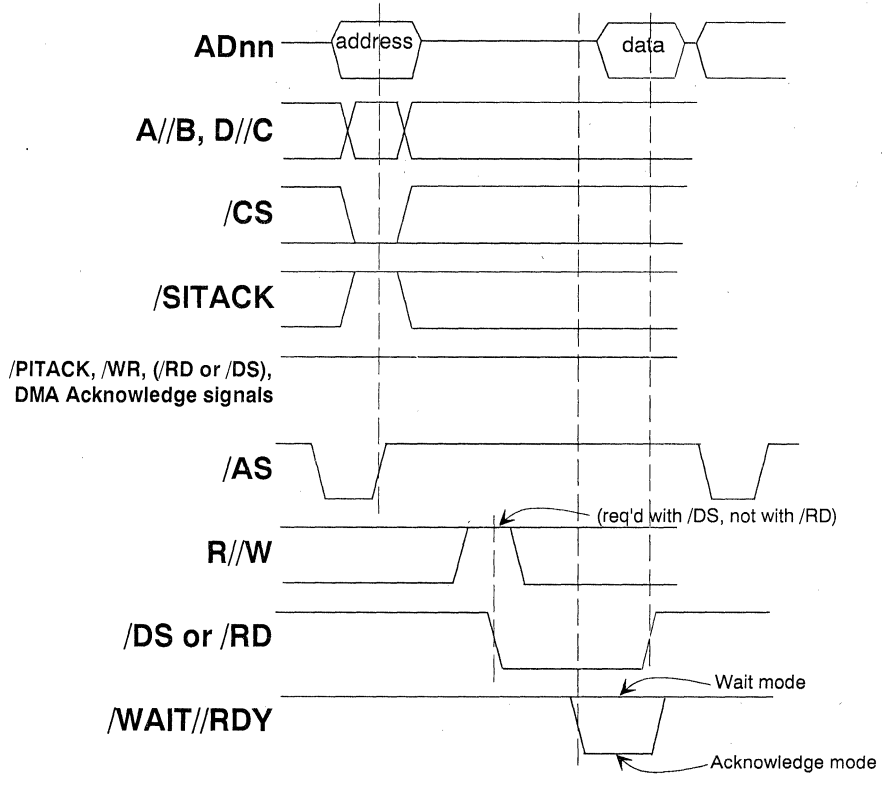


Figure 11. A Register Read Cycle with Multiplexed Addresses and Data

2. The difference between "wait" and "acknowledge" signalling is handled by showing the /WAIT//RDY trace as "maybe or maybe not" going low, with appropriate labelling. (The USC never asserts a "Wait" indication during a register access cycle.)

Chapter 5 covers details of DMA cycles initiated by an external DMA controller, while Chapter 6 covers interrupt acknowledge cycles.

The actual timing parameters and electrical specifications of the USC are given in the companion publication *USC Product Specification.*

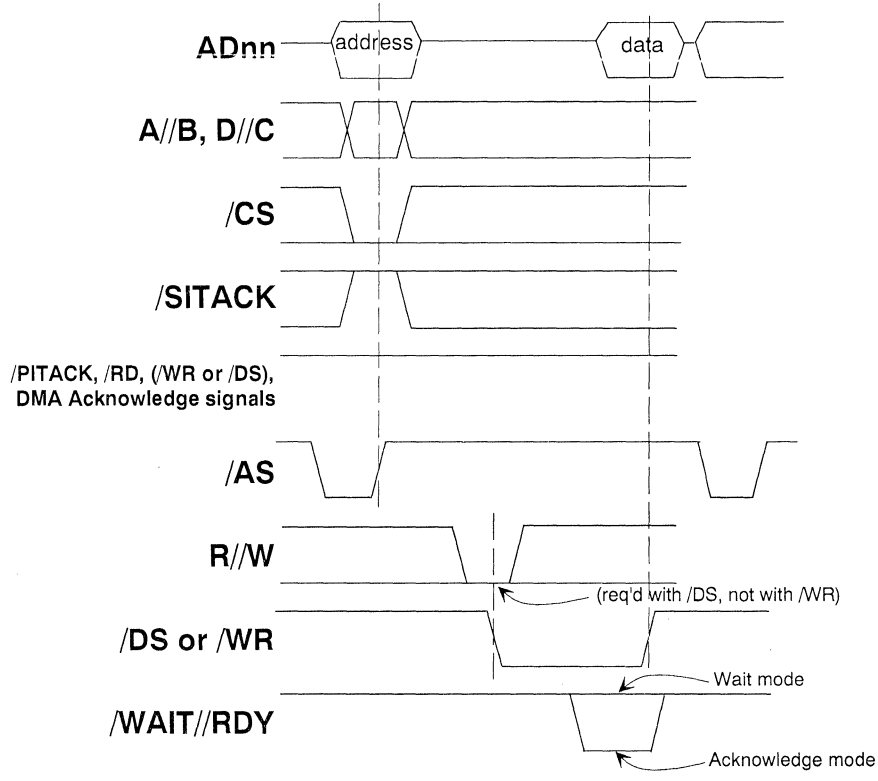


Figure 12. A Register Write Cycle with Multiplexed Addresses and Data

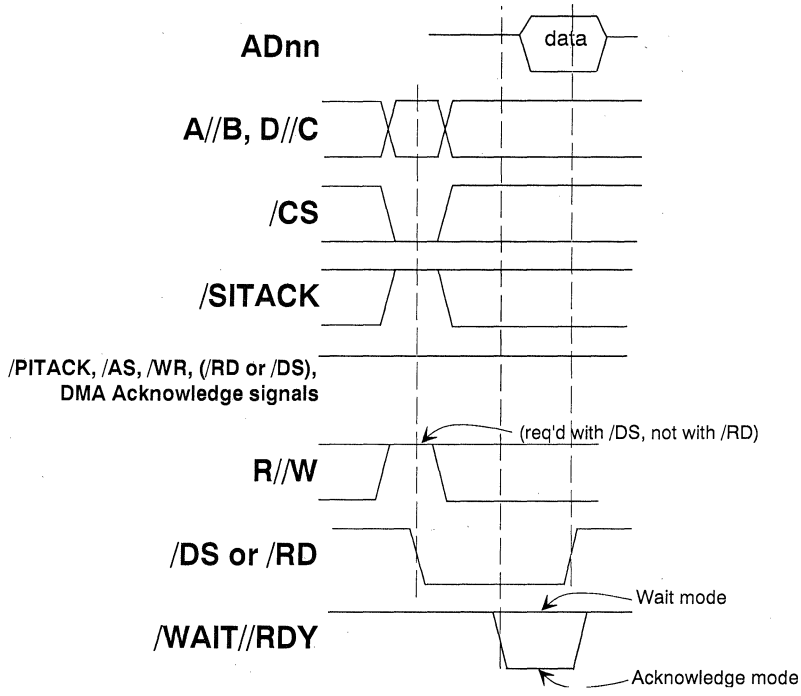


Figure 13. A Register Read Cycle with Non-Multiplexed Data Lines

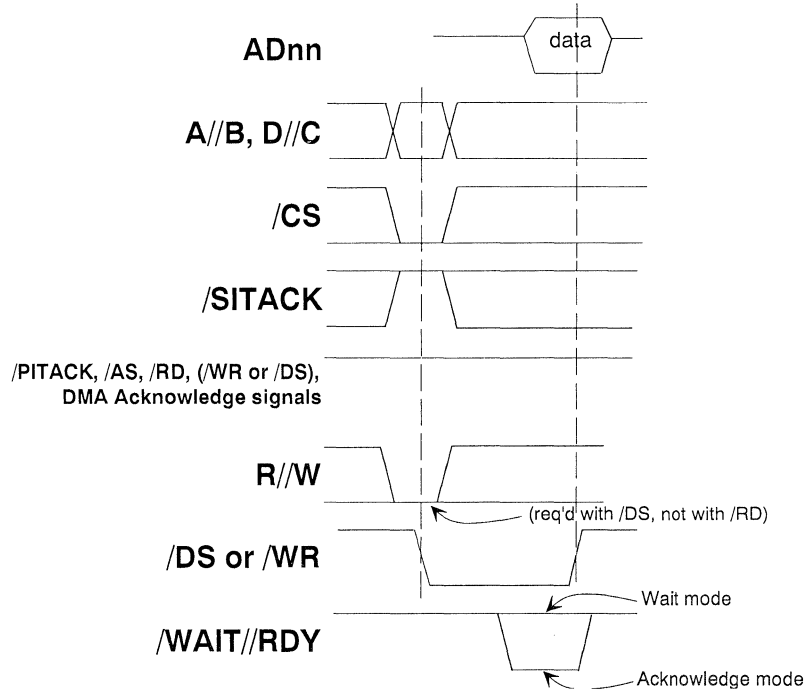


Figure 14. A Register Write Cycle with Non-Multiplexed Data Lines

3. Serial Interfacing

The USC includes several serial interface options and features that promote its usefulness in various kinds of applications. It allows a variety of *clocking schemes,* and will do *serial encoding and decoding* for NRZI and Biphase formats that carry clocking information with the serial data. The USC further supports such decoding with an on-chip *Digital Phase Locked Loop* circuit. Finally, it also provides *I/O lines* that can be connected to modem control and status signals, to other control and status lines related to the serial link, or even to input and/or output signals that aren't related to the serial link at all.

Transmit and Receive Clocking

The USC's Receiver and Transmitter logic have separate internal clock signals that we'll call RxCLK and TxCLK. In most of the USC's operating modes, the Receiver samples a new bit on RxD once per cycle of RxCLK, and the Transmitter presents a new bit on TxD for each cycle of TxCLK. One exception is asynchronous mode, in which RxCLK and TxCLK run at 16, 32, or 64 times the bit rate on RxD and TxD respectively. The other exception involves Biphase-encoded serial data, for which the Receiver samples RxD on both edges of RxCLK, and the Transmitter may change TxD on both edges of TxCLK.

Figure 15 shows how RxCLK and TxCLK can be derived in several different ways. This flexibility is an important part of the USC's ability to adapt to a wide range of applications.

In the simplest case, external logic derives clocks indicating bit boundaries, and software programs the channel to take RxCLK directly from the /RxC pin and TxCLK directly from the /TxC pin. When a channel uses such external clocking for synchronous operation with "NRZ" data, it samples a new bit on the RxD pin on each rising edge on /RxC, and presents each new bit on the TxD pin on the falling edge of /TxC.

It is often desirable to vary the bit rates for transmission and reception by programming the USC, rather than by means of off-chip hardware. To provide for this, each channel includes independent means by which high-speed clocking on /RxC or /TxC can be divided down to almost any desired bit rate.

CTR0 and CTR1

There are two separate 5-bit counters called CTR0 and CTR1 in each channel of a USC, comprising the "first stage" of the channel's clock-generation logic. Figure 16 shows the Clock Mode Control Register. Its **CTR0Src** and **CTR1Src** fields (CMCR13-12 and CMCR15-14 respectively) control whether each counter runs and whether it takes its input from the /RxC or /TxC pin:

CTRnSRC	CTRn clock source
00	CTRn disabled
01	Reserved (disabled)
10	CTRn input = /RxC pin
11	CTRn input = /TxC pin

Figure 17 shows the Hardware Configuration Register. Its **CTR0Div** field (HCR15-14) controls the factor by which CTR0 divides its input to produce its output:

CTR0Div	CTR0 operation
00	CTR0 output = input / 32
01	CTR0 output = input / 16
10	CTR0 output = input / 8
11	CTR0 output = input / 4

There were not enough register bits to allow a separate 2-bit "CTR1Div" field. If the **CTR1DSel** bit in the Hardware Configuration Register (HCR13) is 0, the CTR0Div field determines the factor by which both CTR1 and CTR0 divide their inputs to produce their outputs. If CTR1DSel is 1, the DPLLDiv field in the Hardware Configuration Register (HCR11-10) determines the factor by which both CTR1 and the DPLL divide their inputs to produce their outputs. In either case, the channel interprets the selected 2-bit field as shown above for CTR0Div.

The output of either counter can be used directly as RxCLK and/or TxCLK. It can be used as the input to either of two Baud Rate Generators called BRG0 and BRG1, and it can be routed to the /RxC or /TxC pin.

The Baud Rate Generators

There are two 16-bit down counters called BRG0 and BRG1 in each channel of a USC; they form the "second stage" of the channel's clock-generation logic. The **BRG0Src** and **BRG1Src** fields in the Clock Mode Control Register (CMCR9-8 and CMCR11-10 respectively) control each BRG's input:

BRGnSRC	BRGn clock source
00	CTR0 output
01	CTR1 output
10	/RxC pin
11	/TxC pin

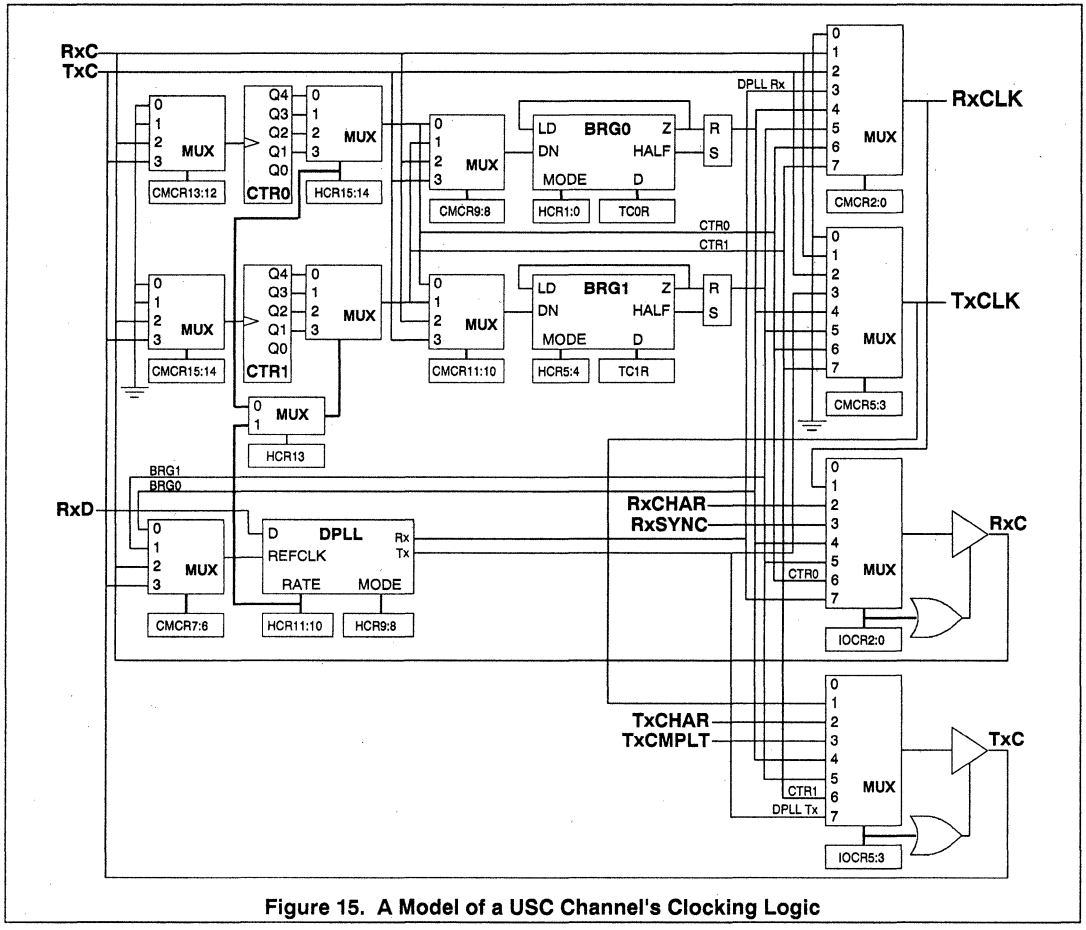


Figure 15. A Model of a USC Channel's Clocking Logic

CTR1Src	CTR0Src	BRG1Src	BRG0Src	DPLLSrc	TxCLKSrc	RxCLKSrc
15 14	13 12	11 10	9 8	7 6	5 4 3	2 1 0

Figure 16. The Clock Mode Control Register (CMCR)

CTR0Div	CTR1 DSel	CVOK	DPLLDiv	DPLLMode	TxAMode	BRG1S	BRG1E	RxAMode	BRG0S	BRG0E
15 14	13	12	11 10	9 8	7 6	5	4	3 2	1	0

Figure 17. The Hardware Configuration Register (HCR)

Each of the two Time Constant registers (**TC0R** and **TC1R**) contains a 16-bit starting value for the corresponding BRG down-counter. Zero in a Time Constant Register makes a BRG's output clock identical with its input clock; a value of one makes a BRG divide its input clock by two, and so on -- the all-ones value makes a BRG divide its input clock by 65,536 to produce its output clock. This flexibility of dividing by any value means that a channel can derive almost any baud rate from almost any input clock, unlike some competing devices that constrain the system designer to use specified crystal or oscillator values and constrain the available speeds to certain commonly-used baud rates.

The **BRG0E** and **BRG1E** bits in the Hardware Configuration Register (HCR0 and HCR4 respectively; the "E" in the names is for "Enable") control whether each Baud Rate Generator runs or not. A 0 in one of these bits inhibits/blocks down-counting by the corresponding BRG, keeping the current value in the down counter unchanged despite transitions on the selected input clock. A 1 in one of these bits enables the corresponding BRG to count down in response to input clock transitions.

When a Baud Rate Generator counts down to zero, it sets the **BRG0L/U** or **BRG1L/U** bit in the Miscellaneous Interrupt Status Register (MISR1 or 0). Once one of these bits is set, it stays set until software writes a 1 to the bit, to "unlatch" it".

A BRG may or may not continue to operate after counting down to zero, depending on the **BRG0S** or **BRG1S** bit in the Hardware Configuration Register (HCR1 or HCR5 respectively; the "S" stands for "Single cycle"). A 0 in BRGnS causes BRGn to reload the TCn value automatically and continue operation, while BRGnS=1 makes BRGn stop when it reaches 0.

Software can (re)load the value in the Time Constant register(s) into one or both BRG counters by writing a Load TC0, Load TC1, or Load TC0 and TC1 command to the RTCmd field of the Channel Command / Address Register (CCAR15-11), as described in the *Commands* section of Chapter 4. These commands also restart a BRG that's in Single Cycle mode and has counted down to zero and stopped.

The **TC0RSel** bit in a channel's Receive Interrupt Control Register (RICR0) and the **TC1RSel** bit in its Transmit Interrupt Control Register (TICR0) control what data the channel provides when software reads the TC0R and TC1R register addresses. If a TCnRSel bit is 0, the channel returns the time constant value last written to TCn. When a 1 is written to a TCnRSel bit, the channel captures the current value of the BRGn counter into a special latch, and thereafter returns the captured value from this latch when software reads the TCn address. Note that in order to obtain a series of relatively current values of a running BRGn, software has to write a 1 to the TCnRSel bit just before each time it reads the TCn location.

The output of either Baud Rate Generator can be used as RxCLK and/or TxCLK. It can be used as the reference clock input to the Digital Phase Locked Loop (DPLL) circuit, and it can be output on the /RxC or /TxC pin.

When a Baud Rate Generator isn't used to make a serial clock, software can use it for other purposes such as protocol timeouts, and can program the channel to request an interrupt when it counts down to zero. Chapter 6 covers interrupts in detail, but to use BRG interrupts software should write 1's to the BRG1 IA bit and/or BRG0 IA bit in the Status Interrupt Control Register (SICR1 and/or SICR0), as well as to the MIE and Misc IE bits in the Interrupt Control Register (ICR15 and ICR0).

Introduction to the DPLL

There is one Digital Phase Locked Loop (DPLL) circuit in each channel of a USC; it represents the "third stage" of the channel's clock-generation logic. The DPLL is a 5-bit counter with control logic that monitors the serial data on RxD. The DPLLSrc field of the Clock Mode Control Register (CMCR7-6) controls which signal the DPLL uses as its nominal or reference clock:

DPLLSrc	DPLL reference clock
00	BRG0 output
01	BRG1 output
10	/RxC pin
11	/TxC pin

The DPLLDiv field of the Hardware Configuration Register (HCR11-10) determines whether the DPLL divides this reference clock by 2, 16, or 32 to arrive at its nominal bit rate, as follows:

DPLLDiv	Nominal DPLL clock
00	reference clock / 32
01	reference clock / 16
10	reference clock / 8
11	Reserved (/4 for CTR1)

The 11 value cannot be used for DPLL operation, but if the DPLL isn't used, software can program this value, together with a 1 in the CTR1DSel bit (HCR13), to operate CTR1 in "divide by four" mode.

A later section describes the operation of the DPLL in greater detail, but for now it's sufficient to note that it samples the (typically encoded) data stream on RxD to produce separate receive and transmit outputs. These outputs are synchronized to the bit boundaries on RxD, and can be used as RxCLK and/or TxCLK and/or can be routed to the /RxC or /TxC pin.

TxCLK and RxCLK Selection

The Transmitter can take its TxCLK from any of the sources described in preceding sections, under control of the **TxCLKSrc** field of the Clock Mode Control Register (CMCR5-3):

TxCLKSrc	Source of TxCLK
000	No clock (xmitter disabled)
001	/RxC pin
010	/TxC pin
011	Tx output of DPLL
100	BRG0 output
101	BRG1 output
110	CTR0 output
111	CTR1 output

Similarly, the Receiver can take its RxCLK from various sources, under control of the **RxCLKSrc** field of the Clock Mode Control Register (CMCR2-0):

RxCLKSrc	Source of RxCLK
000	No clock (receiver disabled)
001	/RxC pin
010	/TxC pin
011	Rx output of DPLL
100	BRG0 output
101	BRG1 output
110	CTR0 output
111	CTR1 output

Clocking for Asynchronous Mode

For asynchronous reception, transitions on RxCLK don't have to have any relationship to transitions on RxD. When the Receiver is searching for a start bit, it samples RxD in each cycle of RxCLK, which it divides by 16, 32, or 64 to determine the bit rate. After the Receiver finds the 1-to-0 transition at the beginning of each start bit, it counts off the appropriate number of RxCLK cycles to the middle of the bit cell. At this point it samples RxD to validate the start bit. If RxD has gone back to 1, the Receiver ignores the prior transition as line noise and goes back to searching for a start bit. If RxD is still 0, the Receiver accepts the start bit. Then it counts off 16, 32, or 64 RxCLK cycles to the middle of each subsequent bit of the character, and samples RxD at those times.

For asynchronous transmission, if a Transmitter has been idle and software then provides it with data and enables it, it drives TxD from 1 to 0 for the Start bit at the falling edge on TxCLK that follows the latter of these two steps. It applies each subsequent bit to TxD after counting off 16, 32, or 64 TxCLK cycles. When sending successive async characters, the Transmitter waits for the stop bit length programmed in the two MSBits of the TxSubMode field of the Channel Mode Register (CMR15-14), before driving TxD from 1 to 0 for a subsequent start bit. If these bits specify "shaved" operation, the Transmitter adjusts the stop bit length per the TxShaveL field of the Channel Control Register (CCR11-8).

Synchronous Clocking

Except in asynchronous operation, one cycle on RxCLK corresponds to one data bit on RxD, and one TxCLK cycle corresponds to one bit on TxD. In any of the synchronous modes, the clock used by the receiver to sample the data must be similar to the one used by the remote transmitter to send the data.

The simplest way to ensure this is to use a separate wire to send the clock from one station's transmitter to the other station's receiver. But often cost or the nature of the serial medium prevents this -- for example, you can't send a separate clock over a telephone line. In such cases it is common practise to encode the data so that serial stream also includes clocking information. For such applications, the USC can encode transmitted data and decode received data in any of several popular formats.

In addition, each channel's Digital Phase Locked Loop (DPLL) module can recover a synchronized RxCLK from the received data. While the DPLL can source TxCLK as well, such operation propagates some of the clock jitter from this station's receive path onto its transmit path, which may increase the error rate.

Stopping the Clocks

CMOS circuits like those in the USC don't draw much power compared to older technologies, but their power requirements can be reduced still further if their clock signals are stopped when the circuits don't need to operate. Most of this power savings can be obtained by having the software disable RxCLK and TxCLK by writing zeroes to the RxCLKSrc and TxCLKSrc fields (CMCR2-0 and CMCR5-3). If the Counters and Baud Rate Generators are used, power consumption is reduced further if software disables them by writing zeroes to as many as possible among CTR0Src, CTR1Src, BRG0Src, and BRG1Src (CMCR13-12, CMCR15-14, CMCR9-8, and CMCR11-10). The ultimate in power savings is obtained by having external logic stop the input clock(s) on the /RxC and/or /TxC pins.

When RxCLK is stopped, previously-received data can be read from the RxFIFO, but RxD is ignored so that no further data will arrive. A final character will be available to the software and/or the Receive DMA controller if RxCLK runs for at least three cycles after its last bit is sampled from RxD. For HDLC/SDLC this means at least 3 RxCLKs after the receiver samples the last bit of a closing Flag. For Async it means at least 3 RxCLKs after the receiver samples the stop bit of the last character.

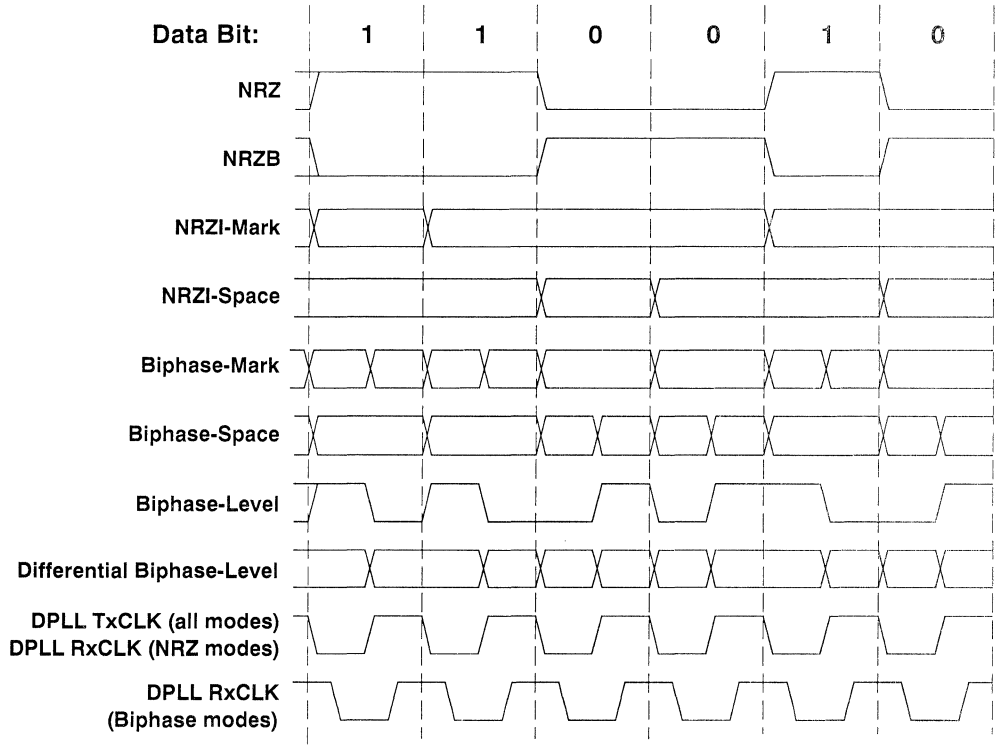
TxCLK can be stopped after the last desired bit has gone out on TxD. This is 2 or 3 TxCLKs after the last bit has left the Transmit shift register (because of the Transmit encoding logic), which in turn occurs 1 or 2 TxCLKs after the Transmitter sets the TxUnder bit (TCSR1).

Data Formats and Encoding

The USC's Transmitter and Receiver can handle data in any of the eight formats shown in Figure 18. The **RxDecode** field in the Receive Mode Register (RMR15-13) controls the format for the Receiver, and the **TxEncode** field in the Transmit Mode Register (TMR15-13) controls it for the Transmitter. The channel interprets both fields as follows:

xMR15-13	Data Format
000	NRZ
001	NRZB
010	NRZI-Mark
011	NRZI-Space
100	Biphase-Mark
101	Biphase-Space
110	Biphase-Level
111	Differential Biphase-Level

NRZ mode doesn't involve any encoding: at the start of each bit cell the transmitter makes TxD low for a 0 or high for a 1. **NRZB** mode is similar except that the transmitter and receiver invert the data: a low is a 1 and a high is a 0.



Note: No assumption is made about the starting state of the serial data in this figure. As a result, those encoding schemes that operate in terms of transitions rather than levels are shown with dual traces corresponding to their two possible starting states.

Figure 18. Data Formats / Encoding

In **NRZI-Mark** mode, at the start of each bit cell the transmitter inverts TxD for a 1 but leaves it unchanged for a 0. In **NRZI-Space** mode, at the start of each bit cell the transmitter inverts TxD for a 0 but leaves it unchanged for a 1.

None of these NRZ-type modes, by itself, guarantees transitions in the data stream. However, if the serial protocol can guarantee transitions often enough, then the DPLL can use these transitions to recover a clock from the data stream. By some method the protocol must eliminate long bit sequences without transitions in the data: successive zeroes for NRZ, NRZB, and NRZI-Mark and successive ones for NRZ, NRZB, and NRZI-Space.

For example, NRZI-Space mode matches up well with HDLC and SDLC protocols, because the Transmitter inserts a extra zero into the data stream whenever the transmitted data would otherwise produce six ones in succession. Thus, there is at least one transition every seven bit times.

The reliability of clock recovery from any kind of NRZ data stream depends on guaranteed transitions, on the transmitter's and receiver's time bases being reasonably similar/accurate, and on fairly low phase distortion in the serial medium. Such schemes have the advantage that bits can be sent at rates up to the maximum switching rate (baud rate) of the medium.

The four Biphase modes, on the other hand, provide highly reliable clock recovery and do not constrain the content of the data, but they limit the data rate to half the switching rate (baud rate) of the serial medium.

See the waveform for **Biphase-Mark** mode in Figure 18. This encoding scheme is also known as FM1. The transmitter always inverts the data at the start of each bit cell. At the midpoint of the cell it changes the data again to indicate a 1-bit, but leaves the data unchanged for a zero. In **Biphase-Space** mode (FM0) the transmitter always inverts the data at the start of each bit cell. In the middle of the cell it changes the data again for a zero-bit but leaves the data unchanged for a one-bit. In **Biphase-Level** mode (also called Manchester encoding), at the start of the bit cell the transmitter makes TxD high for a one-bit and low for a zero. It always inverts TxD in the middle of the cell. In **Differential Biphase Level** mode, at the start of each bit cell the transmitter inverts TxD for a zero but leaves it unchanged for a one. It always inverts TxD in the middle of the cell.

More About the DPLL

While the Transmitter and Receiver must be programmed for the particular serial format to be used, the DPLL only needs to know the general category of encoding on RxD, in the **DPLLMode** field of the Hardware Configuration Register (HCR9-8):

DPLLMode	DPLL Operation/Decoding
00	DPLL disabled
01	Any NRZ mode
10	Biphase-Mark or -Space
11	Either Biphase-Level mode

In any of the NRZ modes, transitions on RxD occur only at the boundaries between bit cells. The DPLL synthesizes a clock having falling edges at bit cell boundaries and rising edges in the middle of the cells. The Transmitter changes TxD on falling edges of TxCLK and the Receiver samples data on rising edges of RxCLK.

In the Biphase-Mark and Biphase-Space encodings, there is always a transition at the boundaries between active data bits, and there may or may not be a transition at the center of each bit cell. The DPLL generates a receive clock having its falling edge 1/4 of the way through the bit cell, and its rising edge at the 3/4 point. The Receiver determines each data bit from the state of RxD at rising edges of RxCLK and checks for "missing clocks" around falling edges. The DPLL generates a Transmit clock that is the same as in NRZ modes. The Transmitter complements the state of TxD at each falling edge of TxCLK, and may or may not change TxD at rising edges, depending on the current data bit.

In the Biphase-Level and Differential Biphase-Level encodings, there is always a transition at the midpoint of each active data bit, and there may or may not be transitions at the boundaries between bit cells. The DPLL generates clocks as for Biphase-Mark and -Space, but must know the difference between those modes and these to do so. The Receiver determines each data bit from the state of RxD at falling edges of RxCLK and checks for "missing clocks" around rising edges. The Transmitter may or may not change TxD at falling edges of TxCLK, depending on the current data bit. It always inverts TxD at rising edges.

RCCF Ovflo	RCCF Avail	Clear RCCF	DPLL Sync	DPLL 2Miss	DPLL 1Miss	DPLLEdge		On Loop	Loop Send	Resrvd	TxResidue			/TxACK	/RxACK
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 19. The Channel Command/Status Register (CCSR)

The DPLL does not include logic to track the clock frequency of the remote end in a long-term manner. Rather it is a counter that is affected by transitions on RxD, and uses the reference clock to make bit clocking that is more or less synchronized to these transitions. Figure 19 shows the USC's Channel Command/Status Register. Its **DPLLEdge** field (CCSR9-8) provides further control over DPLL operation. For most applications, this field should be 00, in which case the DPLL resynchronizes its counter on both rising and falling edges on RxD.

For NRZ applications in which one kind of edge is significantly more precise than the other, software can program the DPLLEdge field to 10 or 01, to make the DPLL ignore one kind of transition. One example of such an application is a serial bus with passive external pull-ups; in such a application, falling edges are more accurate than rising edges. If DPLLEdge is 11, the DPLL never resynchronizes -- that is, it runs freely like CTR0 and CTR1.

Because the blocking of edges by DPLLEdge affects missing clock detection as well as resynchronization, for Biphase operation DPLLEdge should always be programmed as 00.

In any NRZ mode, when the DPLL is in sync, it uses the selected nominal value (8, 16, or 32 cycles of its input clock) for counting off the next bit cell if a transition on RxD falls near the bit cell boundary. If a transition comes early it uses the nominal value minus 1 for the next cell, while if a transition comes late it uses the nominal value plus one. In /16 and /32 modes only, the DPLL uses the nominal value plus two for the next bit cell if a transition comes very late in a cell, and the nominal value minus two if a transition comes very early.

In Biphase-Mark or Biphase-Space modes, when the DPLL is in sync it ignores "data" transitions in the second and third quarters of the bit cell, and resynchronizes to "clock" transitions in the fourth and first quarters of the cell. If a clock transition falls very close to the cell boundary, the DPLL uses the nominal value (8, 16, or 32) as the length of the next bit cell. Otherwise it uses the nominal value minus one if a clock transition comes early, or the nominal value plus one if a clock transition is late.

In Biphase-Level or Differential Biphase-Level modes, when the DPLL is in sync it ignores "data" transitions in the first and fourth quarters of the bit cell, and resynchronizes to "clock" transitions in the second and third quarters of the cell. If a clock transition falls close to the middle of the cell, the DPLL uses the nominal value (8, 16, or 32) as the length of the next bit cell. Otherwise it uses the nominal value minus one if a clock transition comes early, or the nominal value plus one if the clock transition is late.

In an NRZ mode, if there's no transition in a bit cell the DPLL uses the nominal value (8, 16, or 32 clocks) as the length of the next bit cell. It also does this in Biphase modes, if there is no clock transition in a bit cell when the DPLL is in sync. In particular, in these cases the DPLL doesn't re-apply a correction from a previous bit cell.

In Biphase modes, the **CVOK** bit in the Hardware Control Register (HCR12) controls whether the Receiver flags a single code violation as an error. If CVOK=0, it sets the DPLL1Miss bit for a single code violation as described below. If CVOK=1, it doesn't report a single code violation in DPLL1Miss; use this setting when the protocol includes single code violations as normal occurrences, as in the 1533B mode that's described in Chapter 4. Regardless of CVOK, code violations in two consecutive bit cells set the DPLL2Miss and DPLLDSync L/U bits and de-synchronize the DPLL.

After software sets up the DPLL, three bits in the Channel Command/Status Register (CCSR) provide the operating interface. The logic enters a "fast sync mode" when software writes a 1 to the **DPLLSync** bit (CCSR12), or in a Biphase mode when it detects two consecutive missing clocks. In this mode, the next RxD transition (that's allowed by the DPLLEdge field) resynchronizes the DPLL counter and puts the DPLL "back in sync".

The DPLLSync bit in the Channel Command/Status Register (CCSR12) reads as 1 if the DPLL is in sync. The **DPLL2Miss** bit (CCSR11) reads as 1 if the DPLL is in a biphase mode and has detected missing clocks in two consecutive bit cells. The **DPLL1Miss** bit (CCSR10) reads as 1 if the DPLL is in a biphase mode, the CVOK bit (HCR12) is 0, and the DPLL has detected a missing clock in at least one cell. Once DPLL2Miss or DPLL1Miss is 1, it continues to read that way until software writes a 1 to it.

Writing a 0 to any of DPLLSync, DPLL2Miss, or DPLL1Miss has no effect on the DPLL logic.

The channel sets the **DPLLDSync L/U** bit when it loses sync in a Biphase mode. This bit is similar to DPLL2Miss in that once it's set, it stays that way until software writes a 1 to the bit to "unlatch" it. Chapter 6 explains how to program a channel so that it interrupts the host processor when it sets DPLLDSync.

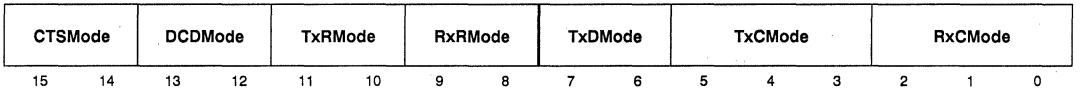


Figure 20. The Input/Output Control Register (IOCR)

The RxD and TxD Pins

In some sense these are the most important pins on a USC. Typically they carry the serial input to the Receiver and the serial output of the Transmitter respectively. Figure 20 shows the I/O Control Register. Its **TxDMode** field (IOCR7-6) allows software to control the function of TxD:

TxDMode	Function of the TxD pin
00	Totem-pole Transmitter output
01	High-impedance state
10	Low output
11	High output

Software can use the ability to drive TxD low to generate a Break condition in Asynchronous applications. The duration of such a Break is fully under software control.

The ability to put the TxD pin in a high-impedance state allows software to use the USC in "serial bus" schemes that include multiple senders on the same signal line. (But note that the TxDMode field resets to 00, so that the channel drives TxD after a Reset until the software programs TxDMode to 01.) The ability for direct programmable control over the TxD pin allows software to "bit-bang" unusual/occasional serial protocol requirements, while keeping the USC's full power for more standard and everyday communications.

The **RTMode** field of the Channel Command/Address register (CCAR9-8) controls the relationship between the Transmitter and the Receiver and thus between the TxD and RxD pins. It is encoded as follows:

RTMode	Operation
00	Normal operation: the Transmitter and Receiver are completely independent.
01	Echo mode: the state of the RxD pin is copied directly onto the TxD pin. Data from the Transmitter is ignored.
10	Pin Controlled Local Loop: the data from the TxD pin, as determined by the TxDMode field (IOCR7-6), is routed to the Receiver rather than the data from RxD. If TxDMode specs TxD as high impedance, the Receiver can take its input from a remote source via TxD rather than RxD.
11	Internal Local Loop: the data from the Transmitter is routed to the Receiver rather than the data from RxD, regardless of the setting of the TxDMode field (IOCR7-6).

Edge Detection and Interrupts

Software can program each channel to detect rising and/or falling edges on the /CTS, /DCD, /TxC, /RxC, /TxREQ, and /RxREQ pins, and to interrupt when such events occur. Figure 21 shows that the Status Interrupt Control Register (SICR) includes separate Interrupt Arm (IA) bits for rising and falling edges on each of these pins. (Chapter 6 describes the USC's interrupt features in detail.) A 1 in one of these bits makes the channel detect that kind of edge, while a 0 makes it ignore such edges. This edge detection and interrupt mechanism operates without regard for whether the various pins are programmed as inputs or outputs in the I/O Control Register (IOCR).

When a channel detects an edge that's enabled in the SICR, it records the event in an internal "edge detection latch" for that input. This latch is not directly accessible in the USC's register map. Instead, as shown in Figure 22, the Miscellaneous Interrupt Status Register (MISR) includes two bits for each of these six pins, one called a "Latched/Unlatch" or L/U bit, and the other being a "data bit" that has the same name as the pin itself.

A hardware or software Reset sequence clears all the L/U bits to zero. While the L/U bit for a pin is 0, the associated data bit reports and tracks the state of the pin in a "transparent" fashion, with a 1 indicating a low and a 0 indicating a high.

Whenever a pin's L/U bit is 0 and its internal edge-detection latch is set, the channel sets the L/U bit to 1, clears the detection latch, and sets the I/O Pin Interrupt Pending (IOP IP) bit. IOP IP can be read and cleared (and if necessary set) in the Daisy Chain Control Register (DCCR1). Chapter 6 describes how the I/O Pin Enable and Master Interrupt Enable bits determine whether the IP bit actually results in an interrupt request to the processor.

RxCDn IA	RxCUp IA	TxCDn IA	TxCUp IA	RxRDn IA	RxRUp IA	TxRDn IA	TxRUp IA	DCDDn IA	DCDUp IA	CTSDn IA	CTSUp IA	RCC Under IA	DPLL DSync IA	BRG1 IA	BRG0 IA
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 21. The Status Interrupt Control Register (SICR)

RxCL/U	/RxC	TxCL/U	/TxC	RxRL/U	/RxREQ	TxRL/U	/TxREQ	DCDL/U	/DCD	CTSL/U	/CTS	RCC Under L/U	DPLL DSync L/U	BRG1 L/U	BRG0 L/U
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 22. The Miscellaneous Interrupt Status Register (MISR)

While an L/U bit is 1, the state of the associated data bit is frozen (latched). These two bits remain in this state, regardless of further transitions on the pin, until software writes a 1 to the L/U bit. This clears the L/U bit to 0 and "opens" the data bit to once again report and track the state of the pin, at least for an "instant". If one or more enabled transitions occurred while the L/U bit was set, then L/U is set again right after software writes the 1 to it.

Writing a 0 to an L/U bit has no effect, and the channel ignores data written to the "data" bits.

One mode in which software can use this logic is to read the MISR, then immediately write back what it has read. The software should then look for 1's in any and all "interesting" L/U bits, and process/handle all such changes *without rereading the MISR.* To obtain the current state of one of these pins, regardless of the L/U bit, software can write a 1 to the L/U bit and then immediately read back the MISR.

The /DCD Pin

The **DCDMode** field of the I/O Control Register (IOCR13-12) controls the function of this pin:

DCDMode	Function of the /DCD pin
00	Low-active Rx Carrier input
01	Low-active Rx Sync input
10	Low output
11	High output

When DCDMode is 00, software can handle the Carrier indication all by itself. Or, the /DCD signal can enable and disable the Receiver in hardware if software also programs the RxEnable field of the Receive Mode Register (RMR1-0) to 11. In the latter case, the Receiver starts assembling a character only when /DCD is low; if /DCD goes high during a received character, the Receiver aborts/discards it. Figure 23 shows how the required relationship between /DCD and RxD varies depends on the Receiver mode:

* for async, nine-bit, and ACV/1553B modes, /DCD should set up low to the rising edge of RxCLK after the falling edge at which the receiver first samples the start bit on RxD.
* for isochronous mode, /DCD should set up low to the rising edge of RxCLK at which the receiver samples the start bit on RxD.
* for monosync, bisync, and transparent bisync, /DCD should set up low to the rising edge of RxCLK that precedes the one at which the receiver samples the first bit of the last sync pattern before the message.
* for HDLC/SDLC mode, /DCD should set up low to the rising edge of RxCLK at which the receiver samples the ending 0 of the last Flag before the frame.

DCDMode=01 identifies the /DCD pin as an input from external sync detection logic. Software typically programs this value in conjunction with programming the RxMode field of the Channel Mode Register (CMR3-0) with 0001 for External Sync operation or 1001 for 802.3 (Ethernet) operation. For External Sync mode, external logic should drive the /DCD pin low during the RxCLK cycle after the last bit in the sync character. For 802.3 it should drive /DCD low when carrier is detected -- a figure in Chapter 4 shows that the timing relationship to RxD isn't critical but /DCD should go low with at least 58 of the 64 alternating bits that precede the frame left. The Receiver starts sampling RxD at the same rising edge of RxCLK at which it first samples /DCD low. If /DCD goes high during a received character, the Receiver completes receiving the character and transfers it to the Receive FIFO before going inactive.

Sync conditions generated internal to the channel are not output on this pin as on certain predecessor devices, but can be output on the /RxC pin as described later.

The /DCD pin can alternatively be used as a general-purpose output. To do this, simply program DCDMode to 10 to make the channel drive /DCD low, and to 11 to drive the pin high. For such an application the designer may want to connect a pull-up or pulldown resistor to the /DCD pin, because the channel will not drive the pin from the time /RESET goes low until the software programs DCDMode to 10 or 11.

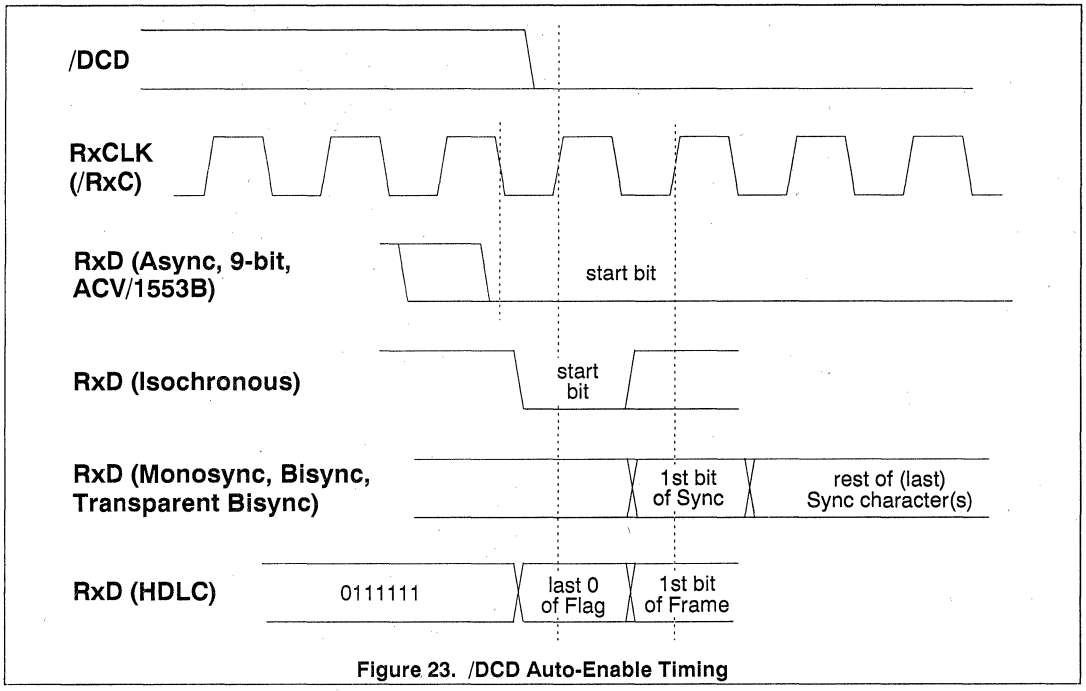


Figure 23. /DCD Auto-Enable Timing

Software can program a channel to interrupt the host processor on either or both edges on /DCD, as described in the preceding section. Typically such interrupts would be used when /DCD is an input, that is, when DCDMode is 00 or 01. Software should write a 1 to the **DCDDn IA** bit in the Status Interrupt Control Register (SICR7) to make a channel detect falling edges on /DCD, and write a 1 to **DCDUp IA** (SICR6) to make it detect rising edges.

As described in the preceding section, the **DCDL/U** bit (MISR7) is 1 if the channel has detected an enabled edge, until software writes a 1 to the bit to clear it. The /**DCD** bit (MISR6) reflects the state of the /DCD pin transparently while DCDL/U is 0, but is frozen while DCDL/U is 1. MISR6=0 indicates a high on the pin, and 1 indicates a low.

The /CTS Pin

The **CTSMode** field of the I/O Control Register (IOCR15-14) controls the function of this pin:

CTSMode	Function of the /CTS pin
0x	Low-active Clear to Send input
10	Low output
11	High output

When CTSMode is 00 or 01, software can handle the Clear to Send input all by itself. Alternatively, the /CTS input can enable and disable the Transmitter in hardware, if software writes 11 to the TxEnable field of the Transmit Mode Register (TMR1-0). In the latter case, the Transmitter will start sending a character only when /CTS is low. As shown in the following Figure, if the Transmitter is otherwise "ready to go" when /CTS goes low, the first bit active bit on TxD will begin at the falling edge of TxCLK that is 4.5 clock periods after the rising edge of TxCLK at which the Transmitter first samples /CTS low.

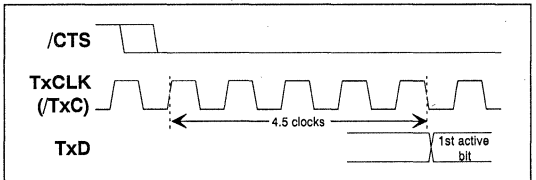


Figure 24. /CTS Auto-Enable Timing

If /CTS goes high during a transmitted character in an asynchronous mode, the Transmitter finishes sending the character before going inactive. In the same situation in a synchronous mode, the Transmitter terminates transmission immediately.

The /CTS pin can alternatively be used as a general-purpose output. To do this, simply program CTSMode to 10 to make the channel drive /CTS low, and to 11 to make it drive the pin high. For such applications the designer may want to connect a pull-up or pulldown resistor to the /CTS pin, because the channel won't

drive the pin from the time /RESET goes low until the software programs CTSMode to 10 or 11.

Software can program a channel to interrupt the host processor on either or both edges on /CTS, as described in the earlier section *Edge Detection and Interrupts.* Typically such interrupts would be used when /CTS is an input, that is, when CTSMode is 00 or 01. Software should write a 1 to the **CTSDn IA** bit in the Status Interrupt Control Register (SICR5) to make a channel detect falling edges on /CTS, and write a 1 to **CTSUp IA** (SICR4) to make it detect rising edges.

As described in *Edge Detection and Interrupts,* the **CTSL/U** bit (MISR5) is 1 if the channel has detected an enabled edge, until software writes a 1 to the bit to clear it. The **/CTS** bit (MISR4) reflects the state of the /CTS pin transparently while CTSL/U is 0, but is frozen while CTSL/U is 1. MISR4=0 indicates a high on the pin, and 1 indicates a low.

The /RxC and /TxC Pins

Figure 15 (near the start of this chapter) shows each channel's options for the function of its /RxC and /TxC pins. The **RxCMode** field in the Input/Output Control Register (IOCR2-0) controls the function of /RxC:

RxCMode	Function of the /RxC pin
000	/RxC is an input
001	/RxC outputs RxCLK
010	/RxC outputs Rx character clock
011	/RxC outputs /RxSYNC
100	/RxC carries the BRG0 output
101	/RxC carries the BRG1 output
110	/RxC carries the CTR0 output
111	/RxC carries the DPLL Rx output

while the **TxCMode** field (IOCR5-3) controls the function of the /TxC pin:

TxCMode	Function of the /TxC pin
000	/TxC is an input
001	/TxC outputs TxCLK
010	/TxC outputs Tx character clock
011	/TxC outputs "Tx Complete"
100	/TxC carries the BRG0 output
101	/TxC carries the BRG1 output
110	/TxC carries the CTR1 output
111	/TxC carries the DPLL Tx output

Some of these possible outputs need further description. A channel drives its **Receive character clock** high for one RxCLK period as it transfers each character from the Receive shift register to the Receive FIFO. Similarly, it drives its **Transmit character clock** high for one TxCLK period each time it transfers a character from the Transmit FIFO to the Transmit shift register. A channel's **/RxSYNC** output goes low for one RxCLK cycle each time its Receiver recognizes a Sync or Flag sequence. The **Tx Complete** output is suitable for controlling a driver on TxD. It is low from the start of the first active bit of a sequence of one or more consecutively-transmitted characters, through the end of the last bit of the sequence. The BRG and CTR outputs are square waves. The DPLL outputs were shown earlier in this chapter.

While it's not very useful to employ a high-speed free-running clock as a source of interrupt events, for other uses of /RxC and /TxC software can program a channel to interrupt the host processor on either or both edges on these pins, as described in the earlier section *Edge Detection and Interrupts.* Typically such interrupts would be used for an input pin, that is, when RxCMode or TxCMode is 00 or 01. Software should write a 1 to the **RxCDn IA** or **TxCDn IA** bit in the Status Interrupt Control Register (SICR15 or SICR13) to make a channel detect falling edges on /RxC or /TxC, and write a 1 to **RxCUp IA** or **TxCUp IA** (SICR14 or SICR13) to make it detect rising edges.

As described in *Edge Detection and Interrupts,* the **RxCL/U** or **TxCL/U** bit (MISR15 or MISR13) is 1 if the channel has detected an enabled edge, until software writes a 1 to the bit to clear it. The **/RxC** or **/TxC** bit (MISR14 or MISR12) reflects the state of the pin transparently while the L/U bit is 0, but is frozen while the L/U bit is 1. A 0 in MISR14 or MISR12 indicates a high on the pin, and 1 indicates a low.

The /RxREQ and /TxREQ Pins

The **RxRMode** and **TxRMode** fields of the I/O Control Register (IOCR9-8 and IOCR11-10 respectively) control the function of these pins:

XxRMode	Function of /XxREQ pin
00	Input pin
01	DMA Request output (or Interrupt Request)
10	Low output
11	High output

Chapter 5 describes the DMA Request function, whereby a channel signals an off-chip DMA controller when its TxFIFO or RxFIFO reaches a programmed degree of "readiness" for DMA data transfer.

Chapter 6 suggests another use for these pins if they're not used as DMA requests, namely as interrupt request outputs that are separate from /INT. This is advantageous in a system in which the host processor and bus provide multiple interrupt request levels and the software uses them for nested interrupts. See *Using /RxREQ and /TxREQ as Interrupt Requests* in Chapter 6 for more details.

Software can program a channel to interrupt the host processor on either or both edges on these pins, as described in the earlier section *Edge Detection and Interrupts.* Typically such interrupts would be used for

an input pin, that is, when RxRMode or TxRMode is 00. Software should write a 1 to the **RxRDn IA** or **TxRDn IA** bit in the Status Interrupt Control Register (SICR11 or SICR9) to make a channel detect falling edges on /RxREQ or /TxREQ, and program **RxRUp IA** or **TxRUp IA** (SICR10 or SICR8) to 1 to make it detect rising edges.

As described in *Edge Detection and Interrupts,* the **RxRL/U** or **TxRL/U** bit (MISR11 or MISR9) is 1 if the channel has detected an enabled edge, until software writes a 1 to the bit to clear it. The **/RxR** or **/TxR** bit (MISR10 or MISR9) reflects the state of the pin transparently while the L/U bit is 0, but is frozen while the L/U bit is 1. A 0 in MISR10 or MISR9 indicates a high on the pin, and 1 indicates a low.

The /RxACK and /TxACK Pins

The **RxAMode** and **TxAMode** fields of the Hardware Configuration Register (HCR3-2 and HCR7-6 respectively) control the function of these pins:

XxAMode	Function of /XxACK pin
00	General purpose input
01	DMA Acknowledge input
10	Low output
11	High output

Chapter 5 describes the DMA Acknowledge function, whereby an off-chip DMA controller signals a USC channel that a "flyby" or single-cycle DMA operation is occurring in response to the channel's assertion of the corresponding REQ pin, and that the channel should provide data on, or capture data from, the AD pins.

The USC does not provide transition-detection, latching, or interrupt capabilities for the /RxACK and /TxACK pins as it does for most of the other signals described in this chapter. Therefore, if these pins aren't used as DMA Acknowledge inputs, they can be used either for outputs or for non-critical polled inputs.

The two LSBits of the Channel Command/Status Register (CCSR1 and CCSR0) allow software to sense the state of a channel's ACK pins if they're used as general-purpose inputs. Figure 19 (earlier in this chapter) shows the CCSR. Its **/TxACK** and **/RxACK** bits are forced to 0 unless the corresponding TxAMode or RxAMode field in the HCR is 00, in which case the bit reads back as a 0 when the /TxACK or /RxACK pin is high and 1 when the pin is low.

4. Serial Modes and Protocols

The main advantage of USC family members is that they can communicate in many different modes and serial protocols. This, in turn, makes for more flexible and capable products for Zilog's customers. This chapter describes how to set up and use the USC in its various modes of serial operation. These modes can be classified into three major categories: asynchronous, character-oriented synchronous, and bit-oriented synchronous protocols.

Asynchronous Modes

These protocols date back to when the first teletype-writers were succeeding Morse code, although there have been various changes since. Figure 25 shows how a *start bit* precedes each character in async communications, and that so-called *stop bits* separate characters. A start bit is a period of space/zero that's the same length as each following data bit. Each stop bit is a period of mark/one having a nominal minimum duration of one bit time. (The USC and other devices offer the ability to "shave" stop bits to less than a bit time.) In most forms of async, the falling edge between a stop bit and the next start bit can come any time after this minimum stop bit duration. In other words, the length of the stop bit does not have to be any particular multiple of the nominal bit time.

To handle this variability in the length of stop bits, asynchronous receivers "oversample" the received serial data at some multiple of the nominal bit frequency. Software can set up each Receiver and Transmitter to do this at 16, 32, or 64 samples/bit. When a Receiver is waiting for a start bit and successive samples reveal a falling edge, it typically samples again one-half bit time later, to validate the start bit. If the serial data is still space/zero, the receiver then samples the following data bits and stop bit at their nominal centers after that. If the hardware samples the stop bit as space/zero, the associated character is invalid or at least highly suspect.

Some async protocols check further for serial link errors by including a parity bit with each character. The transmitter generates such a bit so that the total number of 1-bits in the character is odd or even. The receiving station checks each parity bit. If it finds an incorrect one, it discards the character and/or notifies the operator(s) of the receiving and/or transmitting machine(s). But a single parity bit is not a very reliable checking method -- it can be easily deceived by errors that affect more than one bit. Few async applications use parity checking nowadays, although they may generate it just in case they find themselves talking to equipment that does. Where protection against line errors is important, some async applications may use block-oriented checking as described below for synchronous protocols.

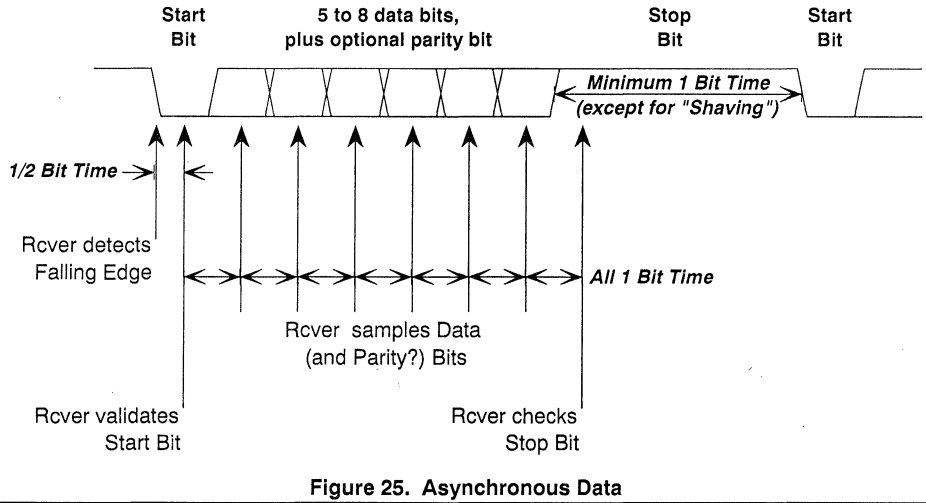


Figure 25. Asynchronous Data

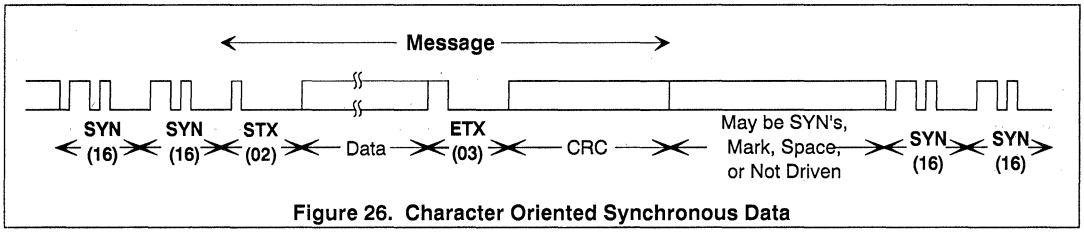


Figure 26. Character Oriented Synchronous Data

Each USC channel can handle a variety of options within "classic" async operation, plus several unique variants. In *isochronous mode,* the data format is similar to classic async, but external hardware supplies a bit-synchronized 1X clock instead of a 16x, 32x, or 64x clock. In *Nine-Bit mode,* an extra bit differentiates between "address" characters that select a particular destination on a multi-station link, and subsequent data characters. In *Code Violation* mode, a three-bit sequence that includes violations of the encoding mode replaces the start bit that precedes each character. (A primary use of Code Violation mode is to implement MIL-STD-1553B.)

Character Oriented Synchronous Modes

These protocols came into use after async, in an effort to get better line utilization by eliminating start and stop bits. In sync modes, characters follow one another directly on the serial link, each consisting of an agreed-upon number of bits and each bit having the same nominal length. Since bits and characters occur at regular intervals, the datacom hardware can typically handle higher bit rates because it doesn't have to oversample as in typical async applications. This effect combines with having fewer bits per character, to make synchronous operation substantially faster than async.

In sync modes, "special" characters divide the data into "messages". Figure 26 shows how the transmitter sends some minimum number of agreed-upon "sync characters" between messages. When a synchronous receiver begins to receive a message, it typically starts in a "search mode" in which it samples successive bits into its serial-to-parallel shift register. It does this until the last N bits match a defined sync pattern. Then the Receiver enters a mode in which it simply captures each succeeding group of bits as a character.

Most sync protocols require the receiving station to validate the sync pattern match. It can do this by checking whether the next character is another sync, an agreed-upon "start of message" character, or perhaps one of a small set of such characters. This validation can be done by software or by hardware.

Almost all character-oriented synchronous protocols also define one or more characters, or sequences of characters, to mark the end of a message. Instead of (or sometimes besides) parity checking on each character, synchronous protocols will typically include a checking code covering most or all the characters in each message. The transmitter accumulates and sends this code before or after the end-of-message character or sequence. Early sync protocols used a Longitudinal Redundancy Character (LRC) that was simply the parallel Exclusive Or of the characters in the message. Newer protocols use various kinds of Cyclic Redundancy Checking (CRC) which offer greater reliability in exchange for a somewhat more involved method of computation. Either kind of message checking can be computed by either hardware or software at the Transmitter and Receiver. The USC channels can automatically generate and check various kinds of CRCs.

Synchronous applications vary considerably in terms of the line state between messages. In half-duplex operation, each station typically stops driving the line after the end of a message. The other side then starts driving it to "turn the line around". In full-duplex point-to-point environments, a transmitter may send a stream of repeated Sync or Idle characters between messages. This maintains synchronization between itself and the remote receiver as to character boundaries. This avoids the need to send several sync characters before the start of the next message, when it becomes available for transmission. In other full-duplex environments, the line may be maintained at a constant Mark or Space between messages.

While many modes have several variants, the top level of each channel's control hierarchy includes the following character-oriented synchronous modes. In *Monosync mode,* the hardware transmits or matches a sync character of eight bits or less. Software must handle further receive-sync validation. In *Bisync mode* the hardware transmits or matches a minimum of two sync characters. The two can be the same or different codes, each of eight bits or less. *Transparent Bisync mode* is similar to Bisync mode except that the prefix character Data Link Escape (DLE) precedes control characters. This allows the transmission of arbitrary "binary" data without conflict with the various control characters. *Slaved Monosync mode* applies only to the Transmitter, making it operate in conformance with the X.21 standard, such that it sends characters in byte-synchronism with those received. *External Sync*

mode applies only to the Receiver, and leaves all sync-detection and framing control to external circuitry. An input signal simply enables the Receiver to assemble characters from the RxD line.

The final character-oriented synchronous mode of the USC channels provides basic facilities for *IEEE 802.3* (Ethernet) operation. At the start of a frame, the Transmitter generates, and the Receiver detects, a preamble consisting of alternating 0 and 1 bits ending with two 1's in succession. Bi-phase-level data encoding must be selected in the Transmit and Receiver Mode Registers (TMR and RMR), as described in Chapter 3. External hardware must be provided to detect collisions and to signal the Transmitter when they occur. External hardware also must signal the Receiver when a frame ends based on loss of carrier. Upon collision detection, "back-off" timing must be determined by external hardware or host processor software.

Bit Oriented Synchronous Modes

As character-oriented synchronous protocols came into wider use in the 1960's and 70's, the number of characters having special significance for the hardware kept increasing. Hand in hand with this, the complexity of the required hardware processing and state machines rose drastically. Particularly troublesome was data "transparency", the ability to transmit any kind of "binary" data without conflict with the various control characters used in these protocols.

These problems might be less severe were they occurring today. But given the technology available in the 1960's, the proliferation of sync protocols was making it harder and harder to build general purpose datacom hardware. Instead, one had to build dedicated communications controllers for each protocol.

Bit oriented synchronous protocols were a response to these problems. IBM's SDLC was the first one widely used; subsequent standardization efforts added several refinements in defining HDLC. These protocols simultaneously minimized the amount of required hardware support, while lifting all restrictions on the content of the data transmitted. Figure 27 shows how in bit-oriented modes, frames are groups of sequential characters, each ending with a CRC code to verify its correctness as in character-oriented protocols. The difference lies in the Flag sequences used to begin, end, and separate frames.

When a bit-oriented synchronous Receiver starts to receive a frame, it looks for a Flag sequence (01111110) just as a character-oriented synchronous Receiver looks for its sync character. While sending a frame, a bit-oriented synchronous Transmitter continually checks whether any sequence of data bits could look like a Flag. It does this without regard for character boundaries. Whenever the data presented to a Transmitter includes a zero followed by five ones, the Transmitter adds an extra zero-bit after the fifth one-bit. Correspondingly, a bit-oriented synchronous Receiver monitors the serial data stream within a frame; any time it sees 0111110, regardless of character boundaries, it deletes the trailing zero.

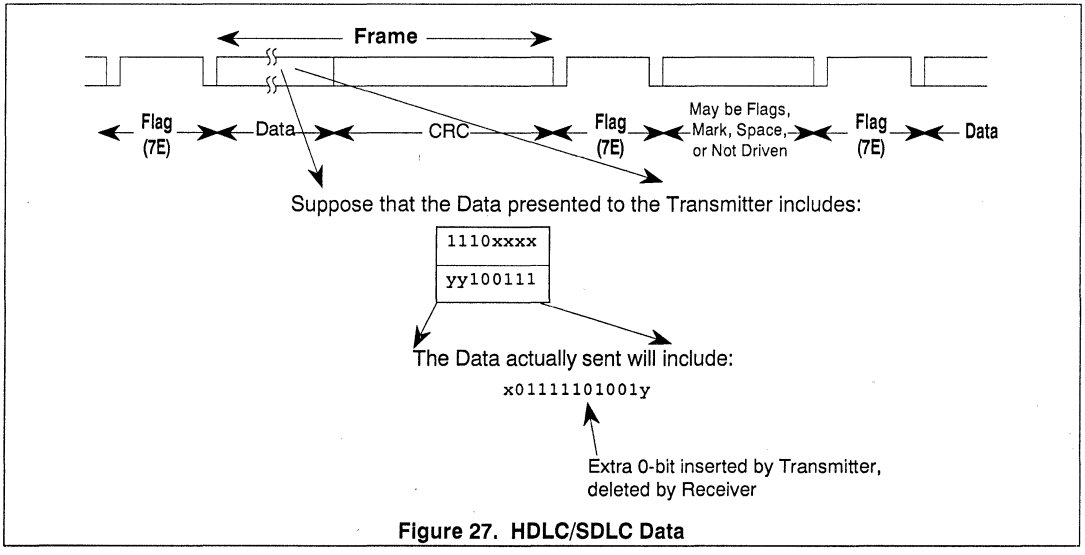


Figure 27. HDLC/SDLC Data

This relatively simple technique allows transmission of any kind of data and assures uniqueness of the Flag sequence within the data stream. (Uniqueness is assured as long as line errors don't occur.) This makes for simpler hardware than with some character-oriented synchronous protocols, in that the hardware only has to recognize a few bit sequences. They include 0111111 for zero-bit-stuffing by a Transmitter, 0111110 for bit removal by a Receiver, a Flag sequence, and finally an Abort sequence. An Abort is a zero followed by more consecutive ones than in a Flag (e.g., 7 or 15 ones).

As mentioned in the previous chapter, SDLC/HDLC protocols match up well with NRZI-Space encoding to ensure data transitions for clock resynchronization. This is because the Transmitter inverts NRZI-space data for every 0-bit and there are never more than five 1-bits in succession within a frame.

Finally, since the Flag-matching hardware operates without regard for character boundaries, bit-oriented synchronous protocols can handle frames that are any number of bits in length. (In character-oriented synchronous protocols, messages must be composed of an integer number of characters.)

The USC can handle most variations of *SDLC* and *HDLC* protocols, since it leaves the details of almost all such variations to the host software. One variation with hardware significance is *Loop mode.* In this mode, the Transmitter can forward received data from the "preceding" station in a loop of stations to the "next" one in the loop. When this station has a frame to send, host software can load the start of the frame into the TxFIFO and then enable the Transmitter. The Transmitter then waits until it detects the transmit-permission token called Go Ahead, which is the same as the short-Abort sequence 01111111 in HDLC/SDLC mode. The Transmitter then changes this character to a Flag and begins transmitting.

The Mode Registers (CMR, TMR and RMR)

Three Mode registers in each channel of the USC control the basic operation and serial protocol of the channel's Transmitter and Receiver.

The Channel Mode Register (CMR) selects among the various communication protocols mentioned in the preceding sections. Figure 28 shows that the MSbyte controls the mode of the Transmitter, while the LSbyte controls that of the Receiver. Software can select the modes of the two modules independently by writing bytes to the CMR, or it can set both modes simultaneously on a 16-bit bus using a 16-bit write.

Within each byte, the four LSbits select the major communications protocol. The coding for these fields is similar but not identical because some modes apply only to the Transmitter while others apply only to the Receiver:

Value	TxMode (CMR11-8)	RxMode (CMR3-0)
0000	Asynchronous	Asynchronous
0001	-	External Sync
0010	Isochronous	Isochronous
0011	Async w/Code V.	Async w/Code V.
0100	Monosync	Monosync
0101	Bisync	Bisync
0110	HDLC/SDLC	HDLC/SDLC
0111	Transp. Bisync	Transp. Bisync
1000	Nine-Bit	Nine-Bit
1001	802.3 (Ethernet)	802.3 (Ethernet)
1010	-	-
1011	-	-
1100	Slaved Monosync	-
1101	-	-
1110	HDLC/SDLC Loop	-
1111	-	-

Zilog reserves values shown above as "-" for future use; they should not be programmed in the indicated field.

Later sections describe each of these modes and protocols individually, including the significance of the Tx and RxSubMode bits (CMR15-12 and CMR7-4 respectively) in each case. The various major modes use the SubMode bits differently, to control protocol variations and options that are specific to each mode. (Sometimes the same SubMode option applies to two or more related major modes.)

Understanding the choices offered by the Channel Mode Register is perhaps the most important single factor in understanding the USC family.

The Transmit and Receive Mode Registers (TMR and RMR) contain basic control information for the Transmitter and Receiver, including the serial format and data-integrity checking. Figures 29 and 30 show the TMR and RMR respectively.

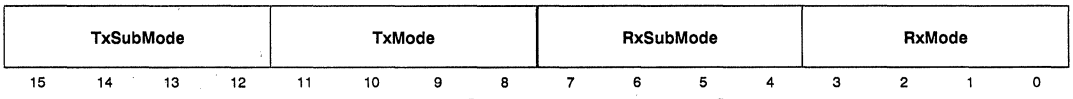


Figure 28. The Channel Mode Register (CMR)

TxEncode			TxCRCType		TxCRC Start	TxCRC Enab	TxCRC atEnd	TxParType		TxPar Enab	TxLength			TxEnable	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 29. The Transmit Mode Register (TMR)

RxDecode			RxCRCType		RxCRC Start	RxCRC Enab	Rsrvd	RxParType		RxPar Enab	RxLength			RxEnable	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 30. The Receive Mode Register (RMR)

Enabling and Disabling the Receiver and Transmitter

The **TxEnable** and **RxEnable** fields (TMR1-0 and RMR1-0) enable and disable the Transmitter and Receiver to send and receive serial data. 00 in TxEnable disables the Transmitter, so that it keeps its output inactive and doesn't transfer characters from the TxFIFO to its shift register. Assuming that the TxDMode field (IOCR7-6) is 00 to propagate the Transmitter's output onto TxD, the pin shows constant Mark/high if the MSBit of the TxIdle field (TCSR10) is 1 and/or the TxEncode field (TMR15-14) is 000 indicating NRZ data. If TxDMode is 00, TCSR10 is 0, and TxEncode is non-zero, the TxD pin shows encoded ones.

If software changes TxEnable to 00 while the Transmitter is sending a character, it discards the character and disables its output immediately. Similarly, 00 in RxEnable disables the Receiver: it ignores the RxD pin and doesn't assemble characters. If software changes this field to 00 while the Receiver is assembling a character, it discards the partial character.

01 in TxEnable or RxEnable disables the Transmitter or Receiver in a more "graceful" way than 00. If software changes TxEnable to 01 while the Transmitter is sending asynchronous data, it finishes sending the current character before going inactive. If software changes TxEnable to 01 while the Transmitter is sending synchronous data, it finishes sending the current frame or message before going inactive. If software changes RxEnable to 01 while the Receiver is receiving asynchronous data, it finishes assembling the current character before going inactive. If software changes RxEnable to 01 while the Receiver is receiving synchronous data, it finishes receiving the current frame or message before going inactive.

10 in TxEnable or RxEnable enables the Transmitter or Receiver unconditionally.

11 in TxEnable places the Transmitter under the control of the /CTS pin. /CTS should be programmed as an input in the CTSMode field of the Input/Output Control Register (IOCR15-14). In this case, the Transmitter only starts sending a character when /CTS is low. If /CTS goes high while the Transmitter is sending a character in an async mode, it finishes sending the character before going inactive. In any synchronous mode, /CTS high summarily disables the Transmitter. In either case, sooner or later, /CTS high forces TxD to Mark or ones as described above for TxEnable=00.

11 in RxEnable places the Receiver under the control of the /DCD pin. /DCD should be programmed as an input in the DCDMode field of the Input/Output Control Register (IOCR13-12). The Receiver ignores the RxD pin and does not assemble characters when /DCD is high. If /DCD goes high while the Receiver is assembling a character in External Sync mode or 802.3 (Ethernet) mode, it finishes assembling the character and places it in the RxFIFO before going inactive. In any other mode the Receiver discards any partial character when /DCD goes high.

Character Length

The **TxLength** and **RxLength** fields (TMR4-2 and RMR4-2) control how many bits the Transmitter sends and the Receiver assembles in each character. The channel interprets both fields as follows:

xMR4-2	Character Length
000	8 bits
001	1 bit
010	2 bits
011	3 bits
100	4 bits
101	5 bits
110	6 bits
111	7 bits

When TxLength specifies less than 8 bits, the Transmitter discards/ignores one or more of the more-significant bits of each byte that it takes from the TxFIFO.

When RxLength specifies less than 8 bits, the Receiver replicates the most significant received bit in the more significant bits of each byte it places in the RxFIFO. For Async mode, it includes a received Parity bit, if any, in each data byte. If RxLength, plus the Parity bit if any, is less than 8 bits, the Receiver fills out the more-significant bits of each byte with the Stop bit, which is 1 except when there's a Framing Error.

When RxLength is less than 8 in synchronous modes including HDLC/SDLC, the Receiver fills out the more

significant bits of each byte with the last received bit (the parity bit if one is used), except in three cases:

1. In Monosync and Bisync modes, when CMR4 is 1 so that sync characters are 8 or 16 bits long, but data characters contain less than 8 bits, each data character is left-justified in its byte.
2. In HDLC/SDLC mode, when CMR5-4 are non-zero so that address and control characters are 8 bits long but subsequent characters are less than 8 bits long, each subsequent character is left-justified in its byte.
3. In HDLC/SDLC mode, if the frame doesn't end on a character boundary, its final data bits are left-justified within the (right-justified) number of bits specified by RxLength, unless case 2 also applies, in which case they're left-justified in the last byte. (The number of bits in the last character of each HDLC/SDLC frame is always indicated in the RxResidue field of the RCSR.)

In any of these three cases of left-justified data, the less-significant bits are left over from the previous character.

If software enables parity checking in an asynchronous mode, the Transmitter and Receiver handle the parity bit as an additional bit after the number of bits defined by TxLength and RxLength. If software selects parity checking in a synchronous mode, the Transmitter and Receiver handle the parity bit as the last of the number of bits specified by TxLength and RxLength.

In Async with Code Violations (1553B) mode only, the Transmitter and Receiver can handle "words" that include up to 16 data bits, treating each word as two characters in the Transmit and Receive FIFOs. When software selects this option, the number of data bits per word is *eight more than* the number usually indicated by TxLength and RxLength.

Software should reprogram RxLength only while the Receiver is either disabled, in Hunt state in a synchronous mode, or between characters in an asynchronous mode. Software can reprogram TxLength at any time, but a new length takes effect only when the Transmitter loads the next character into its shift register.

Parity, CRC, Serial Encoding

A later section of this chapter, *Parity Checking,* discusses how bits 7-5 of both the TMR and RMR control parity checking.

Similarly, a later section of this chapter, *Cyclic Redundancy Checking,* describes how bits 12-8 of the TMR and RMR control CRC checking.

The **TxEncode** and **RxDecode** fields (TMR15-13 and RMR15-13) specify how the Transmitter encodes serial data on the TxD pin and how the Receiver decodes it on the RxD pin. See Chapter 3 for a full description of the following encodings:

xMR15-13	Data Format
000	NRZ
001	NRZB
010	NRZI-Mark
011	NRZI-Space
100	Biphase-Mark
101	Biphase-Space
110	Biphase-Level
111	Differential Biphase-Level

Asynchronous Mode

Software can select classic asynchronous operation for both the Transmitter and the Receiver, by programming the TxMode and RxMode fields (CMR11-8 and CMR3-0 respectively) to 0000. The earlier Figure 25 shows how a "0" Start bit precedes each character and a "Stop bit" follows each, the latter being a "1" condition that's more than 1/2 bit time long. The idle state of the line is 1, and the Transmitter and Receiver divide their input clocks by 16, 32, or 64 to arrive at the nominal bit time.

Software can make the Transmitter calculate and send a parity bit with each character and can make the Receiver check such parity bits, as described in the later section *Parity Checking.*

The two more significant TxSubMode bits (CMR15-14) control the minimum number of Stop bits that the Transmitter sends between consecutive characters. The Transmitter interprets them as follows:

CMR15-14	Minimum Length of Tx Stop
00	One bit time
01	Two bit times
10	One, "shaved" per CCR11-8
11	Two, "shaved" per CCR11-8

When CMR15 is 1 in this mode, the **TxShaveL** field of the Channel Control Register (CCR11-8) controls the exact length of the minimum Stop bit(s). If the 4-bit value in TxShaveL is "n", then the length of the shaved stop bit is (n+1)/16 bit times. The following table summarizes the stop bit possibilities afforded by CMR15-14 and CCR11-8:

CMR15-14	CCR11-8	Minimum Length of Tx Stop
00	xxxx	1 bit time
01	xxxx	2 bit times
10	0000-0111	1/2 or less: DO NOT USE
10	1000	9/16
10	1001	5/8
10	1010-1110	11/16 to 15/16
10	1111	1 (as with CMR15-14=00)
11	0000	17/16
11	0001	9/8
11	0010-1110	19/16 to 31/16
11	1111	2 (as with CMR15-14=01)

The two LSbits of the Tx and RxSubMode fields (CMR13-12 and 5-4) control the factors by which the Transmitter and Receiver divide their TxCLK and RxCLK inputs to arrive at the nominal bit length. The channel interprets both fields as follows:

CMR13-12 & CMR5-4	Nominal Bit Length
00	TxClock or RxClock / 16
01	TxClock or RxClock / 32
10	TxClock or RxClock / 64
11	Reserved, do not program

For the Receiver, choosing a larger divisor makes it sample the data on RxD more often. This may result in a slightly better error rate in marginal circumstances. For the Transmitter there is no significance to the divisor chosen, other than the convenience of choosing the same value as for the Receiver, so that the same source can be used for both RxCLK and TxCLK. (See Chapter 3 for more information about clock selection.)

Zilog reserves the two MSbits of the RxSubMode field (CMR7-6) in Asynchronous mode for use in future products. They should always be programmed as 00.

There is no such thing as a "received stop length" parameter: the Receiver does not expect or check for a particular stop bit length. It simply samples the received data at the nominal midpoint of a single Stop bit, and loads a corresponding Framing Error bit into the RxFIFO with each character. This bit migrates through the FIFO with its associated character and eventually appears as the CRCE/FE bit in the Receive Command/Status Register (RCSR3). Note that RCSR3 can represent the status at the time that a character marked with RxBound[1] status was read from the RxFIFO, or the status of the oldest 1 or 2 characters that are still in the RxFIFO, as described in the later section *Status Reporting.*

Break Conditions

A Break condition is a period of Space (zero) state on an Async line, that's longer than the length of a character. Such a sequence traditionally signals an exceptional condition or a desire to stop transmission in the opposite direction. Alternatively, a Break may mean that the switched or physical connection with the other station is broken. The Receiver detects a Break condition when it samples a supposed Stop bit as Space/zero (a Framing Error) and all the data bits were also Space/zero. In this case the Receiver doesn't place the all-zero character in the RxFIFO, but instead sets the Break/Abort bit in the Receive Command/Status Register (RCSR5). This bit can be enabled to cause an interrupt at the start of a Break, but there's no provision for an interrupt at the end of a Break. Software can tell when the Break ends by polling the Break/Abort bit. This is because the bit doesn't go back to 0 until software has written a 1 to the bit to "unlatch" it, and RxD has gone back to 1/High/Mark.

Software can send a Break by programming the TxDMode field of the Input/Output Control Register (IOCR7-6) to 10 to force TxD to low/space. Then it can use whatever kind of timing resources it has available to measure the desired duration of the Break. After this, it can program TxDMode back to 11 to force TxD to high/mark or to 00 to resume normal operation. Chapter 3 describes a channel's Counters and Baud Rate Generators that may be useful in timing the length of a transmitted Break. While most modern serial controllers will detect a Break that's only slightly longer than a character, older conventions required a Break to be much longer: 200 milliseconds or more.

Isochronous Mode

Software can select Isochronous operation for the Transmitter and the Receiver, by programming the TxMode and RxMode fields (CMR11-8 and CMR3-0 respectively) to 0010. This mode is similar to Asynchronous mode as described above, except that the Transmitter and Receiver use 1X instead of 16X, 32X, or 64X clocking. This typically means that an external bit clock must be provided. It's possible to use the DPLL to recover a 1X clock, but this is a lot like what the Receiver does in Async mode anyway.

Of the options available in the Channel Mode Register for Async mode, the only one that applies in Isochronous mode is CMR14. This controls whether the Transmitter sends one or two stop bits:

CMR14	Length of Tx Stop
0	1 bit time
1	2 bit times

The USC doesn't use the other 3 bits of the TxSubMode field in Isochronous mode, nor any of the RxSubMode bits, but Zilog reserves these bits for functional extensions in future products. Software should always program them with zeroes in Isochronous mode on a USC.

Nine-Bit Mode

This mode is compatible with various equipment including some Intel single-chip microcontrollers. Some contexts call it "address wakeup mode". Software can select it for the Transmitter and the Receiver by programming the TxMode and RxMode fields (CMR11-8 and CMR3-0 respectively) to 1000. Operation on the line is similar to Async mode, using a single stop bit and either eight data bits or seven data bits plus a parity bit. Following the eighth (MS) data

1 Previous USC documentation called RxBound "CV/EOF/EOT".

bit or the Parity bit, an additional bit differentiates normal data characters from "destination address" characters. Address characters identify which of several stations on the link should receive the following data characters. In effect, Nine Bit mode is like a Local Area Network using asynchronous hardware.

The Transmitter saves TxSubMode bit 3 (CMR15) with each character as it goes into the TxFIFO, and sends it as that character's address/data bit. By convention a 0 signifies "data" and a 1 signifies "address". As software or an external Transmit DMA controller writes each character into the TxFIFO, the channel saves the state of CMR15 with it. This bit accompanies the character through the FIFO and out onto the link.

TxSubMode bit 2 (CMR14) selects between eight data bits or seven data bits plus parity:

CMR14	Data bits
0	Eight
1	Seven plus parity. The TxParEnab bit in the Transmit Mode Register (TMR5) must be 1.

Typically, Nine Bit receivers check the parity of received address bytes. This means that when software selects eight data bits, it must calculate its own parity bit in the MSB of addresses.

As in Async mode, the two LSbits of the Tx and RxSubMode fields (CMR13-12 and 5-4) control whether the Transmitter and Receiver divide their TxCLK and RxCLK inputs by 16X, 32X, or 64X to arrive at the nominal bit length. See the preceding Async section for the field encodings and a discussion of the significance of this choice.

The Receiver sets the RxBound status bit for a received address character, that is, a character that has its ninth bit set to 1. This bit accompanies the character through the RxFIFO and ends up in the Receive Command / Status Register (RCSR4). Note that this mode uses the RxBound indicator quite a bit differently from other modes, in that it marks the start of each received block rather than the end. Because of this, some of the mechanisms associated with RxBound, that are described in later sections, aren't of much use in Nine-Bit mode. For example, you probably wouldn't want to store a Receive Status Block for an address character...

The USC doesn't use the two MSBits of the RxSubMode field (CMR7-6) in Nine Bit mode, but Zilog reserves these bits for future enhancements and software should program them as 00 in this mode.

Async with Code Violations (1553B) Mode

Software can select the Async with Code Violations (ACV) mode for the Transmitter and the Receiver by writing 0011 to the TxMode and RxMode fields, CMR11-8 and CMR3-0. The main use of this mode is to implement MIL-STD-1553B communications. However, there are at least two variations of this protocol in use, and the mode has some interesting properties for use in proprietary datacom schemes as well. Therefore this section will discuss the mode "at arm's length from" 1553B itself.

The mode resembles the Isochronous mode in that the Transmitter and Receiver use a 1X clock instead of 16, 32, or 64X oversampling.

1553B defines the smallest indivisible group of data on the line as either 16 or 14 consecutive bits, and calls such a group a "word". This section will use this term in this way, as well as to describe 16 bits transferred to or from a FIFO. We'll call 8 bits transferred to or from a FIFO a "byte".

Zilog recommends ACV mode only with Biphase-encoded data. Standard 1553B uses Biphase-Level encoding while an Army variant uses Biphase-Mark. (Before reading further you may want to review the *Data Formats and Encoding* section of Chapter 3.)

ACV mode replaces the Start bit of Async and Isochronous modes by a unique 3-bit sequence, of which the first and third do not include the usual clock transition. There are two different sequences: two ones followed by a zero begin a "command/status word" while two zeroes followed by a one signify a "data word".

Because of the missing clock transitions, these sequences can't occur in the subsequent data bits for each character. This allows a Receiver to recognize word boundaries even in a continuous data stream. (This can be difficult in normal async applications.)

The idle line state between characters is all ones as in other async modes. Because the 3-bit Start sequences are unique and recognizable, there's no need for a Stop bit to ensure a transition between characters. Therefore Stop bits are optional in ACV mode.

In standard 1553B, 16 data bits follow each Start sequence and are followed by an even parity bit. In an Army variant there are 14 data bits per word and no parity bit. Figure 31 shows these two standard data formats, and includes both kinds of Start sequences for all four Biphase modes.

If software selects ACV mode with any of the four NRZ encodings, the channel sends and scans for Start sequences that contain the same data bits, but such Start sequences aren't uniquely recognizable as in Biphase modes.

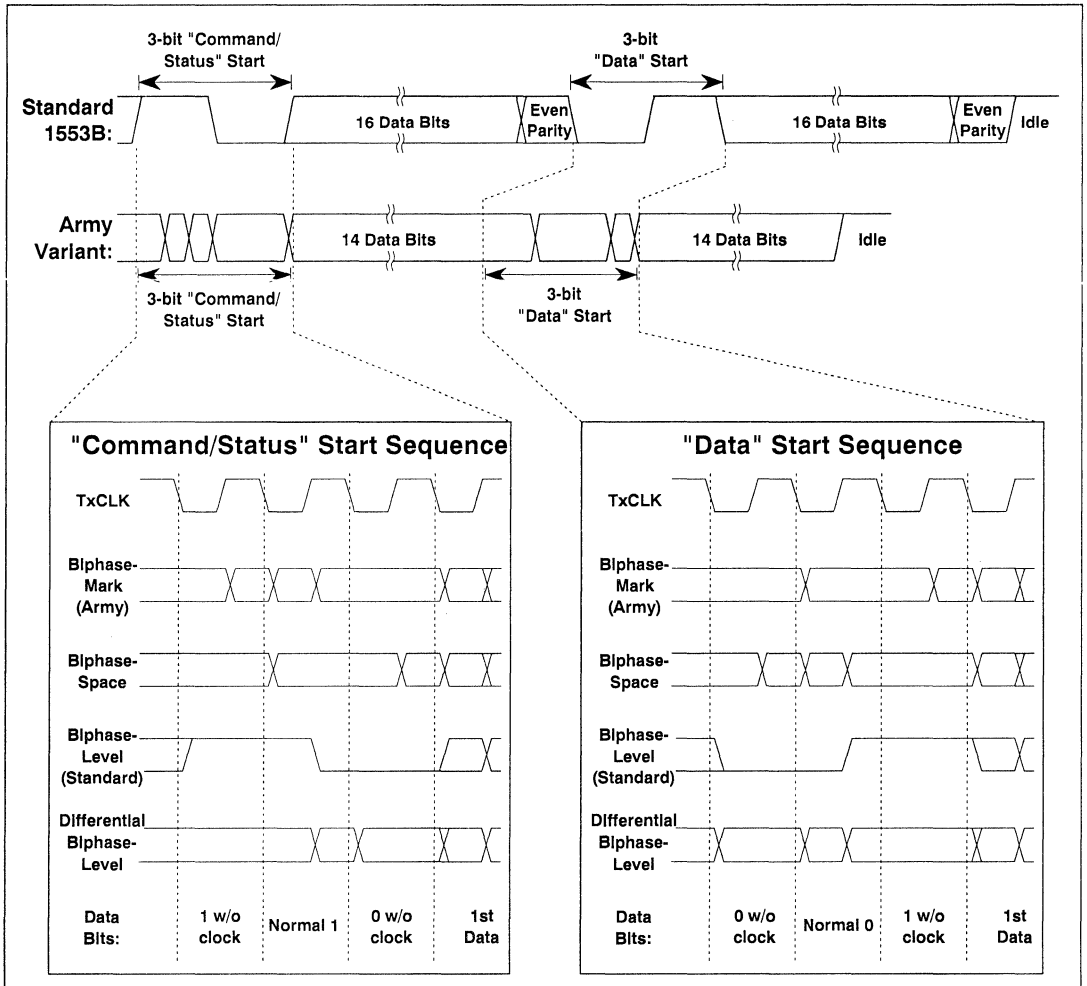


Figure 31. A 1553B Command/Status Word Followed by a Data Word

The two MSBits of the TxSubMode field (CMR15-14) select the minimum number of Stop bits that the Transmitter sends between words:

CMR15-14	# of Stop Bits Transmitted
00	One
01	Two
10	None
11	Reserved; do not program

If the CMR13 bit of the TxSubMode field is 1 in ACV mode, the Transmitter sends 8 more data bits per character/word than the number specified by the TxLength and TxParEnab fields of the Transmit Mode Register (TMR4-2 and TMR5). If CMR13 is 0, the Transmitter sends 8 or less bits in each character, as in other modes.

Similarly, if CMR4 in the RxSubMode field is 1 in ACV mode, the Receiver expects 8 more data bits per character/word than the number specified by the RxLength and RxParEnab fields of the Receive Mode Register (RMR4-2 and RMR5), and it marks the second byte of each received word with RxBound status in the RxFIFO. If CMR4 is 0, the Receiver expects 8 or less bits per character as in other modes, and it marks every received character with RxBound status.

Thus, for standard 1553B communications, program both CMR13 and CMR4 to 1, and program TMR7-2 and RMR7-2 to 001000 to specify 16 total data bits followed by an even parity bit. For the Army variant, again program CMR13 and CMR4 to 1, but program TMR7-2 and RMR7-2 to 000110 for 14 data bits without a parity bit.

When CMR13 and CMR4 are 1, each word on the line corresponds to either one 16-bit transfer, or two 8-bit transfers, to and from the FIFO's. Software can use the commands available in the Channel Command / Address Register (CCAR) to match the bit ordering used on the serial link and the byte ordering employed by the host processor.

The CMR12 bit of the TxSubMode field controls which of the two Start sequences the Transmitter sends in front of each word. When CMR12 is 1, it sends two ones followed by a zero, which signifies a "command/ status word". When CMR12 is 0, it sends two zeroes followed by a one, signifying a "data word". Software has to toggle CMR12 to send the two kinds of words. The channel captures the state of CMR12 in the TxFIFO with each data word, so that software can change it as needed.

The Receiver sends the identity of the Start sequence for each word through the RxFIFO with the data. At the "host end" of the FIFO, this information is available as the ShortF/CVType bit in the Receive Command/ Status Register (RCSR8). ShortF/CVType is 1 for a "command/status" word and 0 for a "data" word. Note that RCSR8 can represent the status at the time that an RxBound character was read from the RxFIFO, or the status of the oldest 1 or 2 characters that are still in the RxFIFO, as described later in *Status Reporting.*

Using "programmed I/O", software has to set CMR12 and sample RCSR8 for the two kinds of Start sequences. These matters can be handled on a DMA basis, without processor intervention for each word. Transmit software can use the Transmit Control Block[2] feature to change CMR12 for each block of words that use the same Start sequence. Receiving software can use the Receive Status Block feature to make the channel store the contents of the RCSR in memory after each word received. This status includes the ShortF/CVType bit. See *Using TCB's and RSB's in ACV (1553B) Mode* later in this chapter for more details on how to use these features.

The USC doesn't use the three MSBits of the RxSubMode field (CMR7-5) in ACV mode, but Zilog reserves these bits for future enhancements and software should always program them as 000 in this mode.

External Sync Mode

Software can select this mode only for the Receiver, by programming the RxMode field of the Channel Mode Register (CMR3-0) as 0001. This value is not defined for the TxMode field (CMR11-8).

This is the most primitive synchronous mode. To use it, software must program the DCDMode field of the Input/Output Control Register (IOCR13-12) to 01, to specify that the /DCD pin carries a Sync input. External hardware must provide a low-active signal on this pin, that controls when the Receiver should capture data. When the external hardware establishes synchronization and/or data validity, it should drive /DCD low. The timing should be such that the channel first samples /DCD low at the same rising edge of RxCLK at which the first data bit that it should capture is available on RxD. (Typically, RxCLK comes directly from the /RxC pin in this mode.)

While /DCD stays low the Receiver samples RxD on each rising edge of RxCLK. Ideally, the external hardware should negate /DCD such that the channel samples it high on the rising RxCLK edge after the one on which it samples the last bit of the last character. But if /DCD goes high while the Receiver is in the midst of assembling a received character, it continues on to sample the remaining bits of the character and place the character in the RxFIFO. Because of this, it's OK for /DCD to go high during the last character, at any time after a hold time after the RxCLK edge at which the Receiver samples the first bit of the character.

Software can make the Receiver check a parity bit in each character as described in the following section *Parity Checking.* Besides or instead of character parity, software can make the Receiver check a CRC code as described in the *Cyclic Redundancy Checking* section.

The USC doesn't use the RxSubMode field (CMR7-4) in External Sync mode, but Zilog reserves this field for future enhancements and software should program it as 0000 in this mode.

Monosync and Bisync Modes

The Binary Synchronous Communications protocol put forth by the IBM Corporation in the 1960's is often abbreviated as "Bisync". But we will use the latter term more generally, to mean a mode of a USC channel, in which the Transmitter sends, and the Receiver searches or "hunts" for, a Sync pattern composed of two characters totalling 16 bits or less. By contrast, we'll use the term "Monosync" to mean a mode in which the Transmitter sends, and the Receiver matches, a sync pattern of eight bits or less. Use of Bisync mode with the two sync characters equal represents a middle ground, having the advantage that the two-character pattern match by the Receiver is more reliable and secure than the sync match in Monosync mode.

Software can select these modes for the Transmitter and/or the Receiver, by programming the value 0100 (for Monosync) or 0101 (for Bisync) into the TxMode

[2] Previous USC documentation called this the Transmit Status Block feature.

and/or RxMode fields of the Channel Mode Register (CMR11-8 and CMR3-0).

Software can make the Transmitter calculate and send a parity bit with each character and can make the Receiver check such parity bits, as described in the *Parity Checking* section.

In such character-oriented synchronous modes, blocks of consecutive characters are called *messages.* Besides or instead of character parity, software can make the Transmitter calculate and send a Cyclic Redundancy Check (CRC) code for each message and can make the Receiver check a CRC in each message, as described later in *Cyclic Redundancy Checking.*

On the transmit side, the Transmitter "concludes a message" in either of two situations: when it underruns or after it sends a character marked with "EOF/EOM" status. The Transmitter underruns when the TxFIFO is empty and the transmit shift register needs a new character. Software can mark a character as End Of Message directly, using a command in the Transmit Command/Status Register (TCSR), or more automatically by using the Transmit Character Counter as described in a later section.

The MSBit of the TxSubMode field (CMR15) determines whether the Transmitter sends a CRC when it concludes a message because of an Underrun condition. The TxCRCatEnd bit in the Transmit Mode Register (TMR8) determines whether it does so when it concludes a message because of a character marked as End Of Message. If CMR15 or TMR8 (as applicable) is 1, the Transmitter sends the CRC code that it has accumulated while sending the message. If CMR15 or TMR8 is 0, it doesn't send a CRC code; if there's any message-level checking, it must be sent like normal data.

After the CRC, or immediately if CMR15 or TMR8 is 0, in Monosync mode the Transmitter sends the Sync character in the LSByte of the Transmit Sync Register (TSR7-0). In Bisync mode it sends the "SYN1" character in TSR15-8 if CMR14 is 0, while if CMR14 is 1 it sends one or more character pairs. The Transmitter takes the first character of each such pair from TSR7-0; by convention it's called "SYN0". The second character of each pair comes from TSR15-8 and is called "SYN1".

After sending this closing Sync character or pair, if/while software doesn't present another message, the Transmitter maintains the TxD signal in the "idle line state" defined by the TxIdle field of the Transmit Command / Status Register (TCSR10-8). If this field is 000, it continues to send more of the same Sync character or pair that it sent to terminate the message. Other TxIdle values select constant or alternating-bit patterns, as described later in *Between Frames, Messages, or Characters.*

If the CMR13 bit in the TxSubMode field is 1, the Transmitter sends a "Preamble" before the "opening" sync character that precedes each message. Software can select the length and content of the Preamble in the Channel Control Register (CCR11-8). A typical use of the Preamble is to send a square-wave pattern for bit rate determination by a phase locked loop.

The Transmitter always sends at least one "opening" Sync pattern before the first data character of a message (after the Preamble if any). In Monosync mode it sends one character from TSR15-8, while in Bisync mode it sends the "SYN0" character from TSR7-0 followed by "SYN1" from TSR15-8. (In Bisync mode an opening Sync sequence is always a character pair, regardless of CMR14.)

The LSBits of the TxSubMode and RxSubMode fields (CMR12 and CMR4 respectively) specify the length of the Sync characters that the Transmitter sends before and after each message and between messages, and for which the Receiver hunts. If CMR12 or CMR4 is 1, sync characters have the same length as data characters, namely the length specified by the TxLength field in the Transmit Mode Register (TMR4-2) or the RxLength field of the Receive Mode Register (RMR4-2). If sync characters are less than 8 bits long, they must be programmed in the least significant bits of TSR15-8, RSR7-0 and, for Bisync, TSR7-0 and RSR15-8. Furthermore, to guarantee that the Receiver matches such Sync characters, the "unused" MSBits among RSR7-0 (and for Bisync RSR15-8) must be programmed equal to the MS active bit.

If CMR12 or CMR4 is 0, Sync characters are 8 bits long regardless of the length of data characters.

On the receive side, the CMR5 bit of the RxSubMode field determines what the Receiver does with Sync characters. In CMR5 is 1, the Receiver strips characters that match the character in RSR15-8, and neither places them in the RxFIFO nor includes them in its CRC calculation. (In Bisync mode, aside from the initial sync match the Receiver treats characters that match "SYN0" in RSR7-0, but don't match "SYN1" in RSR15-8, as normal data.) If CMR5 is 0, the Receiver places all Sync characters inside a message in the RxFIFO and includes them in the CRC calculation.

The USC doesn't use the two MSBits of the RxSubMode field (CMR7-5) in Monosync and Bisync modes, nor CMR14 in the TxSubMode field in Monosync mode. Zilog reserves these bits for future enhancements, and software should always program these bits with zeroes in these modes.

Transparent Bisync Mode

This mode is more specific to the Transparent Mode option of IBM Corp.'s Binary Synchronous Communications protocol than is the Bisync mode described above. Software can select this mode for the Transmitter and the Receiver, by programming the TxMode and RxMode fields of the Channel Mode Register (CMR11-8 and CMR3-0) to 0111.

In Monosync and Bisync modes the Sync characters are programmable, but in this mode a channel uses the fixed characters "DLE" for the first of a sync pair, and "SYN" for the second of a pair. (Software can make the Transmitter send only SYNs for closing and idle Syncs.) The LSBits of the TxSubMode and RxSubMode fields (CMR12 and CMR4) control whether the Transmitter and Receiver use the ASCII or EBCDIC codes for control characters, with a 1 specifying EBCDIC.

Besides using DLE before an opening and possibly a closing SYN, the Transmitter can check whether each data character coming out of the TxFIFO is a DLE and insert another DLE if so. This feature allows any kind of data to be sent "transparently". The Transmitter doesn't include such an inserted DLE in its CRC calculation. Software can selectively enable and disable this function using the **Enable DLE Insertion** and **Disable DLE Insertion** commands, as described later in the *Commands* section. In general software should enable DLE insertion for sending data and disable it for sending a control sequence that starts with DLE. The channel routes the state controlled by these commands through the TxFIFO with each character, so that software can change the state as needed.

Similarly, in Transparent Bisync mode the Receiver checks whether each character coming out of its shift register is a DLE. If so, it sets a state bit. If the next character is also a DLE, the Receiver doesn't include it in the RxFIFO nor in the CRC calculation.

If the character after a DLE is any of the terminating codes "ITB", "ETX", "ETB", "EOT", or "ENQ", the Receiver places the terminating character in the RxFIFO marked with RxBound status. As described in later sections, this marking may set the channel's Received Data Interrupt Pending bit and thus force an interrupt request on its /INT pin, and/or it may force a DMA request on the /RxREQ pin.

The first "DLE-SOH" or "DLE-STX" in a message makes the Receiver enable its CRC generator for subsequent data. Therefore, the CRC in Transparent Bisync mode covers all the data after the first DLE-SOH or DLE-STX.

The Receiver doesn't take any other special action based on received DLE's.

A Transmitter in Transparent Bisync mode sends a DLE-SYN pair at the start of a message, but a Receiver in this mode syncs up to SYN-SYN. This is so that software can determine "transparency" separately for each message, by testing whether the first character of the message in the RxFIFO is a DLE.

The following table shows the ASCII and EBCDIC codes that a channel recognizes in this mode:

Character	ASCII $Code_{16}$	EBCDIC $Code_{16}$
DLE	10	10
ENQ	05	2D
EOT	04	37
ETB	17	26
ETX	03	03
ITB	1F	1F
SOH	01	01
STX	02	02
SYN	16	32

Given the dedicated nature of the Sync characters, the Transmitter interprets the three MSBits of the TxSubMode field similarly to the way it does so in Bisync mode. If CMR15 is 1, it sends a CRC when a Tx Underrun condition occurs. If CMR14 is 1, the Transmitter sends one or more DLE-SYN pairs after a message, else it just sends SYNs. If CMR13 is 1, it sends a Preamble sequence before the opening Sync at the start of each message.

The same data checking options apply to this mode as in Monosync and Bisync, but since we're quite protocol-specific here, we can say that character parity is typically not used while CRC-16 checking is. While the Receiver can detect the end of the frame in Transparent Bisync mode, the Receive Status Block feature can't be used to capture the CRC Error status of the frame, for reasons discussed later in the *Cyclic Redundancy Checking* section. But the selective inclusion/exclusion of received data in the CRC calculation, that's typical of this mode, precludes the kind of automatic reception that the RSB feature allows in modes like HDLC/SDLC anyway.

The USC doesn't use the three MSBits of the RxSubMode field (CMR7-5) in Transparent Bisync mode, but Zilog reserves these bits for future enhancements and software should always program them as 000 in this mode.

Slaved Monosync Mode

This mode applies only to the Transmitter. Software selects it by programming 1100 in the TxMode field of the Channel Mode Register (CMR11-8), while programming 0100 in the RxMode field (CMR3-0) to select Monosync mode for the Receiver.

The mode is intended to implement the X.21 standard and similar schemes in which character boundaries on TxD must align with those on RxD. For this to be meaningful, RxCLK and TxCLK typically come from the same source, as described in Chapter 3.

Most of the setup and operation in this mode is the same as in Monosync mode, which was described in an earlier section. CMR15 determines whether the Transmitter sends a CRC in an Underrun condition. CMR12 selects whether sync characters are the same length as data characters, or are 8 bits long.

CMR13 controls the major operating option in Slaved Monosync mode. (In regular Monosync mode this bit controls whether the Transmitter sends a Preamble before each message; in this mode it can't send one.)

The Transmitter will not go from an inactive to an active state while CMR13 is 0. If CMR13 is 1 when the Receiver signals that it has matched a Sync character, the Transmitter sets the OnLoop bit in the Channel Command / Status Register (CCSR7) and becomes active. That is to say, the Transmitter can go active at any received Sync character, not just one that makes the Receiver exit from "Hunt mode".

Once the Transmitter starts, operation is identical with Monosync mode. The Transmitter sends the Sync character from TSR7-0. Then it sends data from the TxFIFO, until the TxFIFO underruns or until it sends a character marked as End of Message. Then the Transmitter sends the CRC if software has programmed that it should do so for this kind of termination. Finally it sends a Sync character and checks the CMR13 bit again.

If CMR13 is still 1, the Transmitter waits, sending the programmed Idle line condition, until the software triggers it to send another message. If, however, software cleared CMR13 to 0 during the message just concluded, or if it does so while the channel is sending the Idle condition, the Transmitter goes inactive but it leaves OnLoop (CCSR7) set. In the inactive state it sends continuous ones until software programs CMR13 back to 1 again, and the Receiver signals Sync detection.

If all the transmitted and received sync and data characters are the same length, and the same clock is used for both the Transmitter and Receiver, this method of starting transmission assures that transmitted characters start and end simultaneously with received characters, as required by X.21.

The USC doesn't use CMR14 in the TxSubMode field in Slaved Monosync mode, but Zilog reserves this bit for future enhancements and software should always program it as zero in this mode.

IEEE 802.3 (Ethernet) Mode

Software can select this mode for the Transmitter and the Receiver, by programming 1001 into the TxMode and RxMode fields of the Channel Mode Register (CMR11-8 and CMR3-0).

The USC's capabilities for handling Ethernet communications are less comprehensive than those offered by various dedicated Ethernet controllers. In particular, external hardware must detect collisions and generate the pseudo-random "backoff" timing when a collision occurs.

In Ethernet parlance, blocks of consecutive characters are called *frames* rather than messages.

Since Ethernet is a relatively specific, well-defined protocol we can define the proper settings for many of the channel's register fields and options. We can specify the exact values that software should program into the Transmit Mode Register ($D703_{16}$) and Receive Mode Register ($D603_{16}$). These values specify Biphase-Level encoding, a 32-bit CRC sent at End of Frame, no parity, and 8 bit characters, all according to Ethernet practise and IEEE 802.3. In addition the 2 LSBits specify auto-enabling based on signals from external hardware on /CTS and /DCD.

On the transmit side, software should program the TxPreL and TxPrePat fields of the Channel Control Register (CCR11-8) to 1110. This value makes the Transmitter send the 64-bit Preamble pattern 1010... before each frame. In 802.3 mode the Transmitter automatically changes the 64th bit from 0 to 1 to act as the "start bit".

Furthermore, software should program the TxIdle field of the Transmit Command / Status Register (TCSR10-8) to 110 or 111. These values select an Idle line condition of constant Space or Mark. This condition, in turn, allows external logic to detect the missing clock transition in the first bit after the end of the CRC, and turn off its transmit line driver. (In a low-cost variant, such an Idle state can simply disable an open-collector or similar unipolar driver.) Another alternative is to use the Tx Complete output on /TxC to control the driver.

External logic must detect collisions that may occur while the channel is sending, and signal the Transmitter by driving the /CTS pin high when this occurs. Besides the auto-enable already noted for TMR1-0, software should write the CTSMode field of the Input / Output Control Register (IOCR15-14) as 0x to support this use of /CTS.

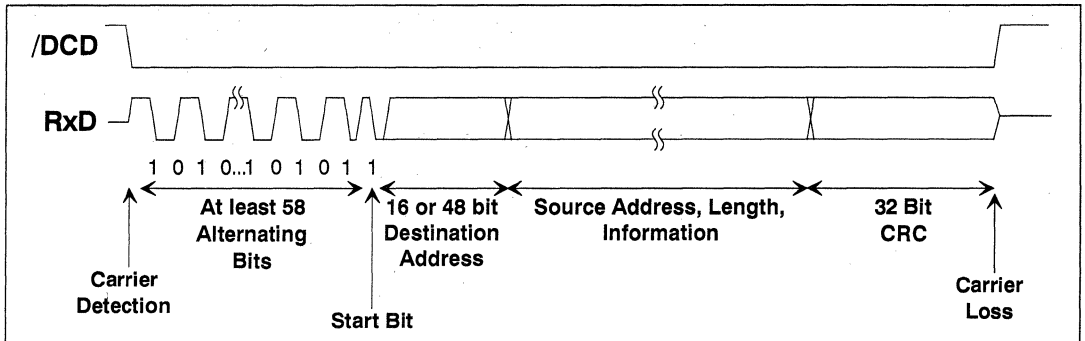


Figure 32. Carrier Detection for a Received Ethernet Frame

As in other synchronous modes, the MSBit of the TxSubMode field (CMR15) controls whether the Transmitter sends its accumulated CRC code if a Transmit Underrun condition occurs.

On the receive side, external logic should monitor the link and drive the /DCD pin low when it detects carrier. Figure 32 shows the relationship between an Ethernet frame on RxD and the signal on /DCD. Besides the auto-enable already noted for RMR1-0, software should program the DCDMode field of the Input / Output Control Register (IOCR13-12) as 01 to control the /DCD pin.

After /DCD goes low, the Receiver hardware hunts for 58 alternating bits of preamble, with the final 0 changed to a 1 as a "start bit". When it finds this sequence it starts assembling data and may check the Destination Address in the frame as described below.

After a frame, the external hardware should drive /DCD high so that it sets up to the rising RxCLK edge after the one at which it samples the last bit of the CRC. In this mode and External Sync mode only among synchronous modes, if /DCD goes high while the Receiver is in the midst of assembling a character, it continues on to sample the remaining bits of the character and place the character in the RxFIFO.

The receiver marks the character that was partially or completely assembled when /DCD went high with RxBound status in the RxFIFO. As described in later sections, this marking may set the channel's Received Data Interrupt Pending bit and thus force an interrupt request on its /INT pin, and/or it may force a DMA request on the /RxREQ pin.

The LSBit of the RxSubMode field (CMR4) controls whether the Receiver checks an Address field at the start of each frame. If CMR4 is 0, the Receiver places all received frames in the RxFIFO and leaves address-checking to the software. (Some contexts call this "promiscuous mode".) If CMR4 is 1, the Receiver compares the first two characters (16 bits) of each frame to the contents of the Receive Sync Register (RSR). It compares RSR0 to the first bit received, and RSR15 to the last bit, regardless of any "Select Serial Data MSB First" commands that the software may have written to the RTCmd field (CCAR15-11). The Receiver ignores the frame unless the address matches, or unless the first 16 bits are all ones, which indicates a frame that should be received by all stations. The Receiver places the address in the RxFIFO so that the software can differentiate "locally addressed" frames from "global" ones.

Except in the CRC, characters ("octets") are sent LSBit first. The Length field that follows the Destination and Source Address fields is sent MSByte-first. IEEE 802.3 doesn't include any other byte ordering information.

The USC doesn't use the three LSBits of the TxSubMode field (CMR14-12) in 802.3 mode, nor the three MSBits of RxSubMode (CMR7-5), but Zilog reserves these bits for future enhancements. Software should always program them with zeroes in this mode.

HDLC / SDLC Mode

Software can select this mode for both the Transmitter and the Receiver, by writing 0110 to the TxMode and RxMode fields of the Channel Mode Register (CMR11-8 and CMR3-0).

In some sense this is the most important mode of the USC, at least for new designs. It is similar to character-oriented synchronous modes in that data characters follow one another on the serial medium without any extra/overhead bits, and are organized into blocks of data with CRC checking applied to the block as a whole.

For HDLC and SDLC, the blocks of data are called *frames.* Uniquely recognizable 8-bit sequences called *Flags,* consisting of 01111110, precede and follow each frame. HDLC/SDLC protocols ensure the uniqueness of Flags, without imposing any restrictions on the data that can be transmitted, by having the Transmitter insert an extra 0 bit whenever the last six

bits it has sent are 011111. A Receiver, in turn, removes such an inserted zero bit whenever it has sampled 0111110 in the last seven bit times.

Besides Flags, HDLC and SDLC define another uniquely recognizable bit sequence called an Abort, consisting of a zero followed by more consecutive ones than the six in a Flag. Depending on the exact dialect of HDLC or SDLC, and the security desired in communicating an abort, software can program the Transmitter to send Aborts consisting of a zero followed by either 7 or 15 consecutive ones.

On the Transmit side, the two MSBits of the TxSubMode field (CMR15-14) control what the Transmitter does if a Transmit Underrun condition occurs, that is, if it needs another character to send but the TxFIFO is empty:

CMR15-14	Underrun Response
00	Send an Abort consisting of 01111111
01	Send an Abort consisting of a zero followed by 15 consecutive ones
10	Send a Flag
11	Send the accumulated CRC followed by a Flag, that is, make the data transmitted so far into a proper frame.

After sending the sequence specified by this field, the Transmitter sends the next frame if software or the external Transmit DMA controller has placed new data in the TxFIFO. Otherwise it sends the Idle line condition specified by the TxIdle field of the Transmit Command / Status Register (TCSR10-8), as described later in *Between Messages, Frames, or Characters.* This section also describes the conditions under which the Transmitter will combine the closing Flag of one frame, and the opening Flag of the next, into a single 8-bit instance.

Software can make the Transmitter send an Abort sequence at any time, by writing the "Send Abort" command to the TCmd field of the Transmit Command/Status Register (TCSR15-12). If the field described above is 01, the Transmitter sends an extended Abort when software issues this command; otherwise it sends the shorter Abort sequence.

If CMR13 is 1, the Transmitter sends the Preamble sequence defined by the TxPreL and TxPrePat fields of the Channel Control Register (CCR11-8), before it sends the opening Flag of each frame.

If the TxIdle field (TCSR10-8) is 000 to select Flags as the idle line condition, CMR12 selects whether consecutive idle Flags share a single intervening 0. If CMR12 is 1, the idle pattern is 011111101111110..., while if CMR12 is 0 it is 01111110 01111110... A Flag that opens or closes a frame never shares a zero with an idle-line Flag, even if CMR12 is 1.

On the Receive side, when the receiver detects the closing Flag of a frame, it marks the preceding (partial or complete) character with RxBound status in the RxFIFO. As described in later sections, this marking may set the channel's Received Data Interrupt Pending bit and thus force an interrupt request on its /INT pin, and/or it may force a DMA request on the /RxREQ pin.

The receiver automatically copes with single Flags between frames, and with shared zeroes between Flags, as described above for the transmit side.

Received Address and Control Field Handling

The RxSubMode field in the Channel Mode Register (CMR7-4) determines how the Receiver processes the start of each frame, i.e., whether it handles Address and/or Control fields. To the extent that the Receiver handles Address or Control field(s), it does so in multiples of 8 bits. Thereafter it divides data into characters of the length specified by the RxLength field of the Receive Mode Register (RMR4-2). The Receiver interprets this field as described below. (An "x" in a bit position means the bit doesn't matter.)

CMR7-4	Address/Control Processing
xx00	The Receiver doesn't handle the Address or Control field. It simply divides all the data in all received frames into characters per RxLength and places it in the RxFIFO.
xx01	The Receiver checks the first 8 bits of each frame as an address. If they are all ones or if they match the contents of the LSByte of the Receive Sync Register (RSR7-0), the Receiver receives the frame into the RxFIFO, otherwise it ignores the frame through the next Flag. After placing the first 16 bits of the frame in the FIFO as two 8-bit bytes, it divides the rest of the frame into characters per RxLength.
x010	The Receiver checks an 8-bit address as described above. If these bits are all ones or if they match the RSR, the Receiver places the first 24 bits of the frame in the RxFIFO as 3 8-bit bytes before shifting to dividing characters according to RxLength.
x110	The Receiver checks an 8-bit address as described above. If these bits are all ones or if they match the RSR, the Receiver places the first 32 bits of the frame in the RxFIFO as 4 8-bit

bytes before shifting to dividing characters according to RxLength.

0011 The Receiver processes an Extended Address at the start of each frame. First it checks the first 8 bits of the frame as described above. If these bits are all ones or if they match the RSR, as the Receiver places each 8 bits of the address into the RxFIFO, it checks the LSBit of the 8. If the LSBit is 0, it goes on to put the next 8 bits into the RxFIFO as part of the address as well, through an address byte that has its LSBit 1. Then, the Receiver places the next 16 bits of the frame into the RxFIFO as two 8-bit bytes, before shifting to dividing characters according to RxLength.

0111 The Receiver processes an Extended Address as described for 0011. If the first 8 bits of the address are all ones or if they match the RSR, the Receiver places the 24 bits after the extended address into the RxFIFO as 3 8-bit bytes, before shifting to dividing characters per RxLength.

1011 The Receiver processes an Extended Address as described for 0011, and then an "Extended Control field". If the first 8 bits of the address are all ones or if they match the RSR, the Receiver places the next 8 bits after the extended address in the RxFIFO without examination. Then, as it stores each subsequent 8 bits in the RxFIFO, the Receiver checks the MSBit of the 8. If the MSBit is 1, it continues to receive more 8-bit bytes, through one that has its MSBit 0. Thereafter the Receiver places one more 8-bit byte into the RxFIFO, before shifting to dividing characters per RxLength.

1111 This mode differs from that described above for 1011 only in that the Receiver places the 16 bits after the extended address in the RxFIFO without examination, before starting to check MSBits for the end of the "extended Control field".

Note that even though the Receiver can scan through an Extended Address, it will still only match its first byte. Note also that it matches RSR0 against the first bit received, and RSR7 against the last bit, regardless of whether software has written a "Select Serial Data MSB First" command to RTCmd (CCAR15-11).

If the RxSubMode field specifies some degree of Address and Control checking, that is, if it's not xx00, and a frame ends before the end of the Address and possibly the Control field specified by the RxSubMode value, the Receiver sets a Short Frame bit in the status for the last character of the frame. This bit migrates through the RxFIFO with the last character, eventually appearing as the ShortF/CVType bit in the Receive Command / Status register (RCSR8). Note that this bit can represent the status at the time that an RxBound character was read from the RxFIFO, or the status of the oldest 1 or 2 characters that are still in the RxFIFO, as described in a later section, *Status Reporting*. Note, however, that this length checking doesn't report a problem if a frame ends within a CRC that follows an address and control field.

If RxLength (RMR4-2) is 000, specifying 8 bits per character, all RxSubMode (CMR7-4) values except xx00 are equivalent aside from short-frame checking.

Frame Length Residuals

The Receiver detects and strips inserted zeroes, Flags, and Aborts before any other processing, and doesn't include these bits/sequences in the RxFIFO nor in CRC calculations. If the Receiver has assembled a partial character when it detects a Flag or Abort, it stores the partial character left-justified in an RxFIFO entry. (That is, in the MSBits of the byte regardless of RxLength.) The Receiver saves the number of bits received in this last byte in the **RxResidue** field of the Receive Command/Status Register (RCSR11-9). RxResidue remains available until the end of the next received frame. Software can use the Receive Status Block feature as described in a later section, to store the RCSR in memory after the frame. This reduces processing requirements still further.

Conversely, to send a frame that doesn't contain an integral number of characters, software must ensure that the number of bits in the last character of the frame is written into the **TxResidue** field of the Channel Command/Status Register (CCSR4-2). This must happen before the Transmitter takes the last character out of the TxFIFO.

Figure 33 shows the CCSR. The Transmit Control Block feature can be used to set the TxResidue value for each block under DMA control, without the intervention of processor software. The active bits of a partial character must be right-justified, that is, they must be the LSBits of the last character. If the TxParEnab bit in the Transmit Command / Status Register (TCSR5) is 1 specifying parity generation, for a partial character the Transmitter sends the parity bit after the number of bits specified by TxResidue, while in other characters the parity bit is the last one of the character length specified by TxLength (TMR4-2).

The encoding of RxResidue and TxResidue is as for RxLength and TxLength: 000 specifies that the last character contains eight bits, while 001-111 specify 1 to 7 bits respectively.

Handling a Received Abort

The later section *Status Handling* will note that a channel sets the Break/Abort bit in the Receive Command/Status Register (RCSR5) immediately when it recognizes an Abort sequence -- this notification is not tied to a specific point in the received data stream. The simplest way for software to handle this condition is to ignore any data that has already been read out of the RxFIFO for the current frame, and issue a Purge Rx FIFO command followed by an Enter Hunt Mode command. (These operations are covered in the later section *Commands*.)

Software can try examining received data, in memory for a DMA application or in the RxFIFO for "programmed I/O", to see if it can locate any valid frames that may have been received before the Abort. But this procedure should take into account that the Abort sequence itself sets the RxBound bit for the character preceding it. It should accept a frame only if 1) the CRCE/FE bit is 0 for its RxBound character, indicating CRC correctness, 2) the RxResidue field is 0 if the protocol restricts frames to multiples of 8 bits in length, and 3) the frame passes any other link-level frame-verification tests for the protocol being used.

HDLC / SDLC Loop Mode

This mode applies only to the Transmitter. Software can select it by programming the TxMode field of the Channel Mode Register (CMR11-8) as 1110 while programming the RxMode field (CMR3-0) as 0110 to select HDLC / SDLC mode.

Loop mode is useful in networks in which the nodes or stations form a physical loop. Except for one station that acts in a "Primary" or Supervisory role, each must pass the data it receives from the "preceding" station to the "following" one. The only time that a secondary station can break out of this echoing mode is when it receives a special sequence called a "Go Ahead" and it has something to send.

Again, this is a specific protocol and we can define how certain other register fields should be programmed for its intended application. For IBM SDLC Loop compatibility, software should program the Transmit Mode Register (TMR) as 6702_{16}. This enables the Transmitter with NRZI-Space encoding, 16-bit CCITT CRC, no parity, and 8 bit characters. Software also should program the TxIdle field in the Transmit Command/Status Register (TCSR10-8) as 000 to select Flags as the idle line state.

The two MSBits of the TxSubMode field (CMR15-14) control what the Transmitter does if an Underrun condition occurs, that is, if it needs a character to send but the TxFIFO is empty. The available choices are similar to those in normal HDLC/SDLC mode but the Transmitter has a wider range of subsequent actions:

CMR15-14	Response to Underrun
00	The Transmitter sends an Abort ("Go Ahead") sequence consisting of a zero followed by seven consecutive ones, and then stops sending and reverts to echoing the data it receives. Zilog doesn't recommend this option in IBM SDLC Loop applications because only the Primary station should issue a "Go Ahead" sequence (and it should be in regular HDLC/ SDLC mode).
01	Like 00 except that the Abort includes 15 one-bits.
10	The Transmitter sends Flags on an Underrun, until another frame is ready or until software clears CMR13 to 0.
11	The Transmitter sends its accumulated CRC followed by Flags on an Underrun, until another frame is ready to transmit or until software clears CMR13 to 0. Zilog doesn't recommend this option either, because the frame format probably hasn't been met when there's an underrun.

The CMR13 bit plays a different role when the Transmitter is first being enabled to "insert this station into the loop", as compared to normal operation after that. Before software programs the Channel Mode Register for SDLC Loop mode and enables the Transmitter, the TxD pin carries continuous Ones. If software initially enables the Transmitter with CMR13 0, it continues to output Ones on TxD. When CMR13 is 1 after software first enables the Transmitter, the channel sends Zeroes on TxD until the Receiver detects a "Go Ahead" sequence (01111111). At this point the channel starts passing data from RxD to TxD with a 4-bit delay, and sets the OnLoop bit in the Channel Command/Status Register (CCSR7; see Figure 33).

RCCF Ovflo	RCCF Avail	Clear RCCF	DPLL Sync	DPLL 2Miss	DPLL 1Miss	DPLLEdge		On Loop	Loop Send	Resrvd	TxResidue			/TxACK	/RxACK
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 33. The Channel Command/Status Register (CCSR)

On Loop stays 1 unless the part is reset or software programs the TxMode field to a different value. Once OnLoop is 1 and the channel is repeating data from RxD to TxD, CMR13 controls what the Transmitter does when it receives a(nother) Go Ahead sequence. If CMR13 is 0, the channel just keeps repeating data, including the "GA". If CMR13 is 1 when the Receiver detects another "Go Ahead", the Transmitter changes the last bit of the GA from 1 to 0 (making it a Flag), sets the **LoopSend** bit (CCSR6) and proceeds to start sending data. (If there's no data available in the TxFIFO it keeps sending Flags, otherwise it sends the data in the TxFIFO.)

When the Transmitter has been sending data and encounters either a character marked as "EOF/EOM", or an underrun condition when CMR15=1, CMR13 determines how it proceeds. If CMR13 is 1 in either of these situations, the Transmitter stays active and sends Flags or additional frames as they become available in the TxFIFO. If CMR13 is 0 after the channel has sent a closing Flag or an idle Flag, it clears the LoopSend (CCSR6) bit and returns to repeating data from RxD onto TxD.

CMR12 controls whether the Transmitter sends idle Flags with shared zero bits, as described for normal HDLC / SDLC mode.

Cyclic Redundancy Checking

A USC channel will send and check CRC codes only in synchronous modes, namely External Sync, Monosync, Slaved Monosync, Bisync, Transparent Bisync, HDLC/SDLC, HDLC/SDLC Loop, and 802.3 modes.

The **TxCRCType** and **RxCRCType** fields in the Transmit and Receive Mode Registers (TMR12-11 and RMR12-11) control how the Transmitter and Receiver accumulate CRC codes.

00 in either field selects the 16-bit CRC-CCITT polynomial $X^{15}+X^{12}+x^5+1$. In HDLC, HDLC Loop, and 802.3 modes, the Transmitter inverts each CRC before sending it, the Receiver checks for remainders of $F0B8_{16}$, and the TxCRCStart and RxCRCStart bit(s) should be programmed as 1 to start the CRC generators with all ones. In other synchronous modes the Transmitter sends accumulated CRCs normally and the Receiver checks for all-zero remainders.

01 in either field selects the CRC-16 polynomial $x^{16}+x^{15}+x^2+1$. The Transmitter sends accumulated CRCs normally and the Receiver checks for all-zero remainders. This choice is not compatible with HDLC, HDLC Loop, and 802.3 protocols, and in these modes CRC-16 will not operate correctly even between USC family Transmitters and Receivers.

10 in TxCRCType or RxCRCType selects the 32-bit Ethernet polynomial $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$. In HDLC, HDLC Loop, and 802.3 modes, the Transmitter inverts each CRC before transmitting it, the Receiver checks for remainders equal to $C704DD7B_{16}$, and the TxCRCStart and/or RxCRCStart bit(s) should be programmed as 1 to start the CRC generator(s) with all ones. In other synchronous modes the Transmitter sends CRCs normally and the Receiver checks for all-zero remainders.

Zilog reserves the value 11 in TxCRCType or RxCRCType for future product enhancements; it should not be programmed.

The **TxCRCStart** and **RxCRCStart** bits (TMR12 and RMR12) control the starting value of the Transmit and Receive CRC generators for each frame or message. A 0 in this bit selects an all-zero starting value and a 1 selects a value of all ones. In HDLC, HDLC Loop, and 802.3 modes these bits should be 1.

The Transmitter and Receiver automatically clear their CRC generators to the state selected by these CRCStart bits at the start of each frame. The Transmitter does this after it sends an opening Sync or Flag sequence. The Receiver does so each time it recognizes a Sync or Flag sequence (it may be the last one before the first character of the frame or message). For special CRC requirements, the **Clear Rx and Tx CRC** commands give software the ability to clear the CRC generators at any time. See the later section *Commands* for a full description of these operations.

The **TxCRCEnab** and **RxCRCEnab** bits (TMR9 and RMR9) control whether the channel processes transmitted and received characters through the respective CRC generators. A 0 excludes characters from the CRC while a 1 includes them. The Transmitter captures the state of TxCRCEnab with each character as it's written into the TxFIFO, so that software can change the bit dynamically for different characters.

If the **TxCRCatEnd** bit (TMR8) is 1 and the TxMode field (CMR11-8) specifies a synchronous mode, the Transmitter sends the contents of its CRC generator after sending a character marked as EOF/EOM. If TxCRCatEnd is 0 the Transmitter doesn't send a CRC after such a character. (A character can be marked as EOF/EOM if software writes a command to the Transmit Command/Status Register (TCSR), or when software or an external Transmit DMA controller writes one or two characters to the TxFIFO so that the Transmit Character Counter decrements to zero.) Whether or not it sends a CRC, the Transmitter then sends a Sync or Flag sequence, depending on the protocol.

In synchronous modes, the MS 1 or 2 bits of the TxSubMode field (CMR15 and in some modes also CMR14) control whether the Transmitter sends the contents of its CRC generator if it encounters a

Transmit Underrun condition, namely if it needs a character to send but the TxFIFO is empty. Whether or not it sends a CRC, the Transmitter then sends a Sync or Flag sequence, depending on the protocol.

On the receive side, in synchronous modes other than HDLC/SDLC, HDLC/SDLC Loop, and 802.3, there's a two character delay between the time a Receiver places each received character in the RxFIFO and when it processes (or doesn't process) the character through the CRC generator. Therefore, software can use examine each received character and set RxCRCEnab appropriately to exclude certain characters from CRC checking, if it can do so before the next one arrives. A Receiver doesn't introduce this delay in HDLC/SDLC, HDLC/SDLC Loop, or 802.3 mode, because in these modes all characters in each frame should be included in the CRC calculation.

Figure 34 shows how a Receiver routes data to the Receive CRC generator differently in HDLC/SDLC, HDLC/SDLC Loop, and 802.3 modes than in other synchronous modes. In the former three modes, the Receiver shifts each bit from RxD into the CRC generator when it shifts the bit into its main shift register. In other sync modes, the Receiver passes the data through a second shift register located between the main shift register and the CRC generator. This second shift register is effectively (RxLength) bits long, and gives the software time to decide whether to include each received character in the CRC calculation.

The Receive CRC generator constantly checks whether its contents are "correct" according to the kind of CRC specified by the RxCRCType field (RMR12-11). In some modes this simply means whether it contains an all-zero value. The CRC generator provides a corresponding Error output that the Receiver captures in the RxFIFO with each received character. This bit migrates through the RxFIFO with each character and eventually appears as the CRCE/FE bit in the Receive Command/Status Register (RCSR3). Software should ignore this bit for all characters except the one associated with the end of each message or frame (it's almost always 1).

The CRCE bit that's important is the one that reflects the output of the CRC generator after the Receiver has shifted the last bit of the CRC into it. But the operating difference described above affects which character this bit is associated with. The Receiver always places the CRC code itself in the RxFIFO; if RxLength calls for 8-bit characters the CRC represents either 2 or 4 characters. In HDLC/SDLC or 802.3 mode, the CRCE bit associated with the last character of the CRC is the one that shows the CRC-correctness of the frame. But in the other synchronous modes, the CRCE bit of interest is the one with the second character after the last character of the CRC. This means that the Receive Status Block feature can't be used to capture the CRC correctness of received messages in Transparent Bisync mode.

Note that the CRCE/FE bit can represent the status at the time that an RxBound character was read from the RxFIFO, or the status of the oldest 1 or 2 characters that are still in the RxFIFO, as described later in *Status Reporting.*

Because the Receiver places all the bits of each received CRC in the RxFIFO, a USC channel can be used for CRC-pass-through applications like bridges and routers. This is not true of all serial controllers.

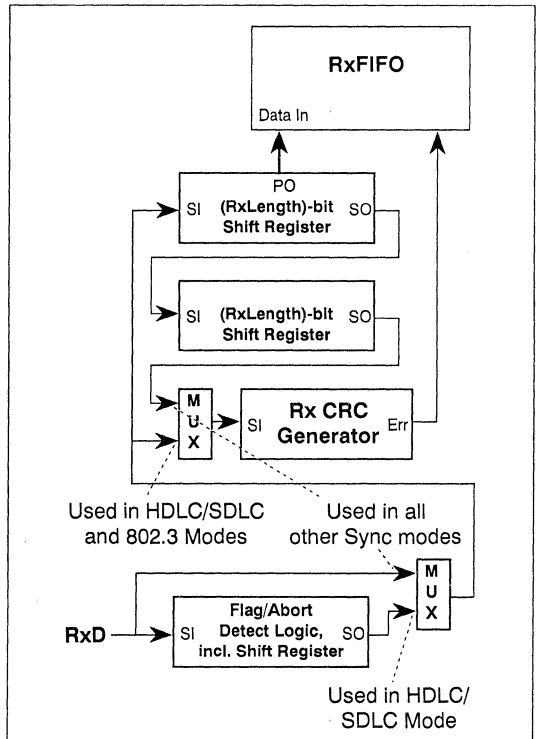


Figure 34. A Model of the Receive Datapath

Parity Checking

A USC channel can handle a Parity bit in each character in either asynchronous or synchronous modes, although some synchronous protocols use CRC checking only.

If the **TxParEnab** bit in the Transmit Mode Register (TMR5) is 1, the Transmitter creates a parity bit as specified by the **TxParType** field (TMR7-6) and sends it with each character. Similarly, if the **RxParEnab** bit (RMR5) is 1, the Receiver checks a parity bit in each received character, according to the **RxParType** field (RMR7-6). A channel interprets TxParType and RxParType as follows:

xMR7-6	Type of Parity
00	Even
01	Odd
10	Zero
11	One

For unencoded data, 10/Zero is the same as "Space parity" and 11/One is the same as "Mark parity".

TxParEnab and TxParType are "global states" in that a channel doesn't carry these bits thru the TxFIFO with each character.

In asynchronous modes, the Transmitter and Receiver handle the parity bit as an additional bit after the number of bits specified by the TxLength and RxLength fields (TMR4-2 and RMR4-2). In synchronous modes they handle the parity bit as the last (most significant) bit of that number. The Receiver includes a parity bit in the data characters in the RxFIFO and Receive Data Register (RDR), except in asynchronous modes with 8 bit data.

If RxParEnab is 1 and the Receiver finds that the parity bit of a received character is not as specified by RxParType, it sets a Parity Error bit. This bit accompanies the character through the RxFIFO, eventually appearing as the PE bit in the Receive Command / Status Register (RCSR2). The PE bit can represent a latched interrupt bit, or the status at the time that an RxBound character was read from the RxFIFO, or the status of the oldest 1 or 2 characters that are still in the RxFIFO, as described in the next section.

Status Reporting

The most important status reported by the Transmitter and Receiver is available in the LSBytes of the Transmit and Receive Command / Status Registers (TCSR and RCSR). Figures 36 and 37 show the format of these registers. It will be helpful to describe some common characteristics of these status bits before discussing each individually.

When software writes and reads transmit and received data directly to and from a serial controller, it can read and write status and control registers as needed to handle the overall communications process. But with the USC, external DMA controllers often handle the data without software/processor intervention. Because of this, software needs other means of controlling the transmit and receive processes and tracking their status. These means include the Transmit and Receive Character Counters and the Transmit Control Block and Receive Status Block features. Later sections describe these features in considerable detail. For now we just note that Receive Status Blocks allow the Receive DMA controller to store a version of the RCSR in memory with the received data. Such stored status differs slightly from that which software can read from the RCSR.

Software can program a channel to assert its Interrupt Request output (/INTA or /INTB) based on certain bits in the TCSR and RCSR. Chapter 6 covers interrupts in detail; for now we'll just note that a channel typically sets one of these bits when a specified event occurs or a specified condition starts. Such a bit typically remains 1 until host software clears or "unlatches" it by writing a 1 to it. This means that a channel won't request another interrupt for the same condition until software has written a 1 to the bit. For the two interrupts that reflect the start of an ongoing condition, IdleRcved and the "break" sense of Break/Abort, the Receiver doesn't clear the RCSR bit until the software has written a 1 to unlatch the bit, and the condition has ended.

Five of the bits in the RCSR (ShortF/CVType, RxBound[3], CRCE/FE, PE, and RxOver) are associated with particular received characters. The Receiver queues these bits through the RxFIFO with the characters. The corresponding bits in the RCSR may reflect the status of the oldest character(s) in the FIFO, or that of the character last read out of the FIFO, as described in the next few paragraphs.

In order for these queued interrupt features to operate properly, software should set the **WordStatus** bit in the Receive Interrupt Control Register (RICR3) to 1 before it reads data from the RxFIFO/RDR 16 bits at a time, and to 0 before it reads data 8 bits at a time.

The RxBound, PE, and RxOver bits actually operate differently in the RCSR depending on whether software has enabled each to act as a source of interrupts. If the Interrupt Arm (IA) bit[4] in the Receive Interrupt Control Register (RICR) for one of these bits is 1, the channel sets the RCSR bit to 1 when a character having the subject status becomes the oldest one in the RxFIFO, or the second-oldest with

---

[3] Previous USC documentation called RxBound "CV/EOF/EOT".

[4] Previous USC documentation called the bits that control individual interrupt sources Interrupt Enable (IE) bits, the same as those that enable entire interrupt types.

WordStatus=1, and once one of these bits is 1, it stays that way until software writes a 1 to it. (The channel doesn't actually set the Receive Status IP bit to request an interrupt for one of these bits, until software or thel Receive DMA controller reads the associated character from RDR.)

For ShortF/CVType and CRCE/FE, and for RxBound, PE, and RxOver when the associated IA bit is 0, if the last time that software or an external Receive DMA controller read this channel's RxFIFO via the RDR, the channel provided a character marked with RxBound status, then these RCSR bits reflect the status of that character. This is true only until software reads the (MSByte of) RCSR, or a Receive DMA controller stores it in the Receive Status Block, or until software or the Receive DMA controller reads RDR again.

For ShortF/CVType and CRCE/FE, and for RxBound, PE, and RxOver when the associated IA bit is 0, if the last time that software or the Receive DMA controller read the RxFIFO via the RDR, the character returned (both of the characters returned) had RxBound=0, or if software has read the (MSByte of the) RCSR or the Receive DMA controller has stored it in a Receive Status Block since the last time either one read the RDR, then the RCSR bit reflects the status of the oldest character(s) in the RxFIFO (if any). In this latter case, if the RxFIFO is empty the status bit is not defined. If the WordStatus bit is 1 in the Receive Interrupt Control Register (RICR3) and there are two or more characters in the FIFO, the status bit is the inclusive OR of the status of the oldest two characters in the FIFO. Otherwise it reflects the status of the oldest character in the FIFO.

Just in case that wasn't perfectly clear, the flowchart of Figure 35 presents the same information.

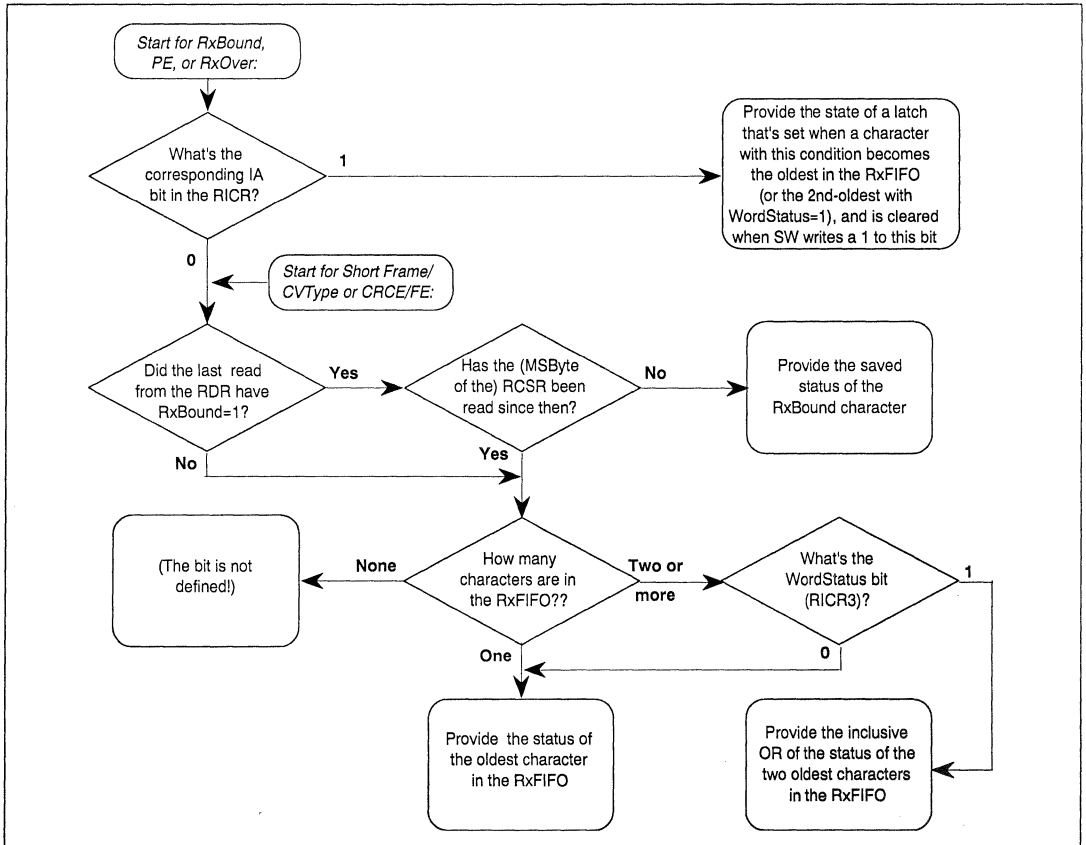


Figure 35. How a USC Channel Provides the "Queued" Status Bits in the RCSR

TCmd				Rsrvd	TxIdle			Pre Sent	Idle Sent	Abort Sent	EOF/ EOM Sent	CRC Sent	All Sent	Tx Under	Tx Empty
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 36. The Transmit Command/Status Register (TCSR)

Detailed Status in the TCSR

The Transmitter sets the **PreSent** bit (TCSR7) in a synchronous mode, when it has finished sending the Preamble specified in the TxPreL and TxPrePat fields of the Channel Control Register (CCR). A channel can request an interrupt when this bit goes from 0 to 1 if the PreSent IA bit in the Transmit Interrupt Control Register (TICR7) is 1. Software must write a 1 to PreSent to unlatch and clear it, and to allow further interrupts if TICR7 is 1; writing a 0 to PreSent has no effect. See the later section *Between Frames, Messages, or Characters* for more information on Preambles.

The Transmitter sets the **IdleSent** bit (TCSR6) in any mode, when it has finished sending "one unit" of the Idle line condition specified in the TxIdle field in the MSByte of this TCSR. If the Idle condition is Syncs or Flags as described later in *Between Frames, Messages, or Characters,* the unit is one character or sequence and the flag and interrupt can recur for each one sent. For any other Idle condition, the Transmitter sets the flag and interrupt only once, when it has sent the first bit of the condition. The channel can request an interrupt when this bit goes from 0 to 1 if the IdleSent IA bit in the Transmit Interrupt Control Register (TICR6) is 1. Software must write a 1 to IdelSent to unlatch and clear it, and to allow further interrupts if TICR6 is 1; writing a 0 to IdleSent has no effect.

The Transmitter sets the **AbortSent** bit (TCSR5) in HDLC/SDLC or HDLC/SDLC Loop mode, when it has finished sending an Abort sequence. A channel can request an interrupt when this bit goes from 0 to 1 if the AbortSent IA bit in the Transmit Interrupt Control Register (TICR5) is 1. Software must write a 1 to AbortSent to unlatch and clear it, and to allow further interrupts if TICR5 is 1; writing a 0 to AbortSent has no effect. See the earlier sections *HDLC/SDLC Mode* and *HDLC/SDLC Loop Mode* for more information on Abort sequences.

The Transmitter sets the **EOF/EOM Sent** bit (TCSR4) in a synchronous mode, when it has finished sending a closing Flag or Sync sequence. A channel can request an interrupt when this bit goes from 0 to 1 if the EOF/EOM Sent IA bit in the Transmit Interrupt Control Register (TICR4) is 1. Software must write a 1 to EOF/EOM Sent to unlatch and clear it, and to allow further interrupts if TICR4 is 1; writing a 0 has no effect. See the later section *Between Frames, Messages, or Characters* for more information on closing Flags and Syncs.

The Transmitter sets the **CRCSent** bit (TCSR3) in a synchronous mode, when it has finished sending a Cyclic Redundancy Check sequence. A channel can request an interrupt when this bit goes from 0 to 1 if the CRC Sent IA bit in the Transmit Interrupt Control Register (TICR3) is 1. Software must write a 1 to CRCSent to unlatch and clear it, and to allow further interrupts if TICR3 is 1; writing a 0 to CRCSent has no effect. See the section *Cyclic Redundancy Checking* for more information on CRC's.

The read-only bit **AllSent** (TCSR2) is 0 in asynchronous modes, while the Transmitter is sending a character. Software can use this bit to figure out when the last character of an async transmission has made it out onto TxD, before changing the mode of the Transmitter.

The Transmitter sets the **TxUnder** bit (TCSR1) in any mode, when it needs another character to send but the TxFIFO is empty. It does this even in asynchronous modes. A channel can request an interrupt when this bit goes from 0 to 1 if the TxUnder IA bit in the Transmit Interrupt Control Register (TICR1) is 1. Software must write a 1 to TxUnder to unlatch and clear it, and to allow further interrupts if TICR1 is 1; writing a 0 has no effect. The Transmitter sets TxUnder one or two clocks before the current character is completely sent on TxD.

The read-only bit **TxEmpty** (TCSR0) is 1 when the TxFIFO is empty, and 0 when it contains 1 or more characters.

RCmd (WO)				RxResidue			ShortF/	Exited	Idle	Break	Rx	CRCE	PE	Rx	Rx
2ndBE	1stBE						CVType	Hunt	Rcved	/Abort	Bound	/FE		Over	Avail
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 37. The Receive Command/Status Register (RCSR)

Detailed Status in the RCSR

A channel sets the read-only **2ndBE** bit (RCSR15) to 1 when software or an external Receive DMA controller reads data from the RDR, there are two or more characters in the RxFIFO, and the Receiver marked the second-oldest one with one or more of RxBound, Parity Error, or Rx Overrun status. (The bit's name stands for Second Byte Exception.) A channel clears this bit to 0 when software or the Receive DMA controller reads data from the RxFIFO/RDR, there are two or more characters in the RxFIFO, and the Receiver didn't mark the second-oldest one with any of these three conditions. If software or the Receive DMA controller reads data from the RDR when there's only one character in it, this bit is undefined until the next time one of them reads RDR.

A channel sets the read-only **1stBE** bit (RCSR14) to 1 when software or an external Receive DMA controller reads data from the RDR, and the Receiver marked the oldest character read with one or more of RxBound, Parity Error, or Rx Overrun status. (The bit's name stands for First Byte Exception.) A channel clears this bit to 0 when software or the Receive DMA controller reads data from the RDR, and the Receiver didn't mark the oldest character with any of these three conditions.

The Receiver queues a **ShortF/CVType** bit through the RxFIFO with each character. RCSR8 may reflect the status at the time that an RxBound character was read from the RxFIFO, or the status associated with the oldest 1 or 2 character(s) still in the RxFIFO, as described earlier in this *Status Reporting* section. In a stored Receive Status Block it always represents the status of the preceding RxBound character.

This bit will be 1 only in HDLC/SDLC or Async with Code Violation (1553B) mode, and only for characters that the Receiver also marks with RxBound=1. When the RxSubMode field (CMR7-4) specifies Address and possibly Control field processing in HDLC/SDLC mode, the Receiver sets this bit for the last character of a frame if it hasn't come to the end of the specified field(s) by the end of the frame.

In Async with Code Violations (1553B) mode, this bit identifies which of the two types of Code Violation introduced each received word. A 0 indicates a Data word and a 1 indicates a Command/Status word. When the RxSubMode bit CMR4 is 1, signifying that each word includes more than 8 data bits, this bit is valid with the second byte of each received word (the one marked with RxBound status).

The Receiver sets the **ExitedHunt** bit (RCSR7) in any mode, when it leaves its Hunt state. In Async modes this happens right after software enables the Receiver. In External Sync mode, the Receiver leaves Hunt state when the Enable/Sync signal on /DCD goes from high to low. In Monosync, Bisync, or Transparent Bisync mode the Receiver leaves Hunt state when it recognizes a Sync sequence. In HDLC/SDLC mode the Receiver leaves Hunt state when it recognizes an opening Flag. In 802.3 (Ethernet) mode, if software has enabled address checking the Receiver leaves Hunt state when it matches the Address at the start of a frame, otherwise it does so after detecting the start bit at the end of the Preamble.

A channel can request an interrupt when this bit goes from 0 to 1 if the ExitedHunt IA bit in the Receive Interrupt Control Register (RICR7) is 1. Software must write a 1 to ExitedHunt to unlatch and clear it, and allow further interrupts if RICR7 is 1; writing a 0 to ExitedHunt has no effect.

The Receiver sets the **IdleRcved** bit (RCSR6) when it samples RxD as one for 15 consecutive RxCLKs in HDLC/SDLC mode, or for 16 consecutive RxCLKs in any other mode. A channel can request an interrupt when this bit goes from 0 to 1 if the IdleRcved IA bit in the Receive Interrupt Control Register (RICR6) is 1. Software must write a 1 to IdleRcved to unlatch it, and to allow further interrupts if RICR6 is 1; writing a 0 has no effect. A channel doesn't actually clear RCSR6 until software has written a 1 to unlatch it, and RxD has gone to 0 to end the idle condition. (IdleRcved isn't useful in Async modes that use a 16X, 32X, or 64X clock. In these cases keep RICR6=0 to avoid interrupts, and ignore RCSR6.)

The Receiver sets the **Break/Abort** bit (RCSR5) in an asynchronous mode when it detects a Break condition, that is, when it samples the Stop bit of a character as 0, and all the preceding data bits (and the parity bit if any) have also been 0. It sets the bit in HDLC/SDLC mode when it detects seven consecutive 1s, i.e., an Abort or Go Ahead sequence.

This bit is not associated with a particular point in the received data stream, for either the Break or Abort condition.

A channel can request an interrupt when this bit goes from 0 to 1 if the Break/Abort IA bit in the Receive Interrupt Control Register (RICR5) is 1. Software

must write a 1 to Break/Abort to unlatch it, and to allow further interrupts if RICR5 is 1; writing a 0 has no effect. In async modes, a channel doesn't actually clear RCSR5 until software has written a 1 to unlatch it, and RxD has gone to 1 to end the break condition.

The Receiver queues a **RxBound**[5] bit through the RxFIFO with each received character. It sets the bit with a character that represents the boundary of a logical grouping of data on the line, but this indication isn't visible to software until the character is the oldest one in the RxFIFO.

As described earlier in this *Status Reporting* section, RCSR4 may represent an interrupt bit, or the status associated with the oldest 1 or 2 character(s) still in the RxFIFO; or may be 1 if a RxBound character was just read from the RxFIFO. Since the Receive Status Block feature stores the RCSR in memory after each character that the Receiver marks with this bit set, a Receive Status Block always shows RxBound as 1.

In HDLC/SDLC mode the Receiver sets RxBound for the last complete or partial character before an ending Flag or Abort. In Transparent Bisync mode it sets this bit for an ENQ, EOT, ETB, ETX, or ITB character that follows a DLE. In External Sync or 802.3 (Ethernet) mode the Receiver sets this bit for the character just completed or partially assembled when the /DCD pin went High. In Nine-Bit mode it sets this bit for an address character. In the Async with Code Violations (1553B) mode, it sets this bit for the second character of each received word if the CMR13 bit is 1 to enable word lengths greater than 8 bits, or for every character if not. Note that the Receiver never sets this bit in other modes, including Monosync and Bisync modes.

A channel can request an interrupt when software or a DMA channel reads a character from the RDR that has this bit set, if the RxBound IA bit in the Receive Interrupt Control Register (RICR4) is 1. In this case software must write a 1 to RxBound to unlatch it and allow further interrupts; writing a 0 has no effect.

The Receiver queues a **CRCE/FE** bit through the RxFIFO with each received character. RCSR3 may represent the status at the time that a RxBound character was read from the RxFIFO, or the status associated with the oldest 1 or 2 character(s) still in the RxFIFO, as described earlier in this *Status Reporting* section. In a stored Receive Status Block it represents the status of the previous character, which in turn represents the CRC-correctness of the frame in 802.3 and HDLC/SDLC modes.

In synchronous modes the Receiver makes CRCE/FE 0 if its CRC generator showed "correct" status when it stored the character in the RxFIFO, or 1 if the CRC generator wasn't correct. See the earlier section *Cyclic Redundancy Checking* for more information. In asynchronous, isochronous, or Nine-Bit mode the Receiver makes this bit 1 to show a Framing Error if it samples the associated character's Stop bit as 0.

The Receiver queues a **PE** bit through the RxFIFO with each received character. RCSR2 may represent an interrupt bit, or the status at the time that a RxBound character was read from the RxFIFO, or the status associated with the oldest 1 or 2 character(s) still in the RxFIFO, as described earlier in this *Status Reporting* section. In a stored Receive Status Block it may represent an interrupt bit or the status of the previous 1 or 2 character(s).

The Receiver sets this bit to show a Parity Error for a character if the RxParEnab bit (RMR5) is 1 and the character's parity bit doesn't match the condition specified by the RxParType field. See the earlier section *Parity Checking* for more information.

A channel can request an interrupt when software or a DMA channel reads a character from the RDR that has this bit set, if the PE IA bit in the Receive Interrupt Control Register (RICR2) is 1. In this case software must write a 1 to Abort/PE to unlatch it and allow further interrupts; writing a 0 to PE has no effect.

The Receiver queues a **RxOver** bit through the RxFIFO with each received character. It sets the bit to indicate a Receive FIFO overrun, but the overrun isn't visible to software until the character that caused it is the oldest one in the RxFIFO.

As described earlier in this *Status Reporting* section, RCSR1 may represent an interrupt bit, or the status at the time a RxBound character was read from the RxFIFO, or the status associated with the oldest 1 or 2 character(s) still in the RxFIFO. In a stored Receive Status Block this bit may represent an interrupt bit or the status of the previous character.

The Receiver sets this bit to 1 for the first character for which there was no room, which overwrites its predecessor in the RxFIFO. Once this happens, the Receiver doesn't store any more received characters in the RxFIFO, until software writes a command that purges the RxFIFO to the RTCmd field in the Channel Command / Address Register (CCAR15-11).

A channel can request an interrupt when software or a DMA channel reads a character from the RDR that has this bit set, if the RxOver IA bit in the Receive Interrupt Control Register (RICR1) is 1. In this case, software must write a 1 to RxOver to unlatch it and allow further interrupts; writing a 0 has no effect.

The read-only bit **RxAvail** (RCSR0) is 1 if the RxFIFO contains 1 or more characters, or 0 if it's empty.

---

[5] Previous USC documentation called RxBound "CV/EOF/EOT".

DMA Support Features

When software writes and reads all the data to and from a serial controller, it can maintain its own counters and length-tracking mechanisms, and can use them to tell when to read status and issue commands. But in DMA applications we would like to "decouple" the processor and its software from such intimate and real-time involvement with the transmit and receive processes. This is only possible if we include features in the serial and/or DMA controllers, by which software can figure out the length and correctness of frames or messages long after they're received, and by which the hardware can change parameters and save status information at appropriate points with as little processor software involvement as possible.

The USC features that support such operation include the Receive and Transmit Character Counters, the RCC FIFO that stores the length of received frames, the Transmit Control Block feature that allows software to include control information with transmit data in Transmit DMA buffers, and the Receive Status Block feature that stores status with received data in Receive DMA buffers. The following subsections describe these features.

The Character Counters

The Transmitter includes a 16-bit Transmit Character Counter (TCC) that software can use to control the length of transmitted frames and messages in DMA applications. The Receiver includes a similar Receive Character Counter (RCC) that software can use to record and save the length of frames and messages in DMA applications. Software can also use the RCC to specify the maximum frame/message length allowed in such applications.

While most of this section describes these features in terms of the length of frames and messages in synchronous protocols, they may be useful in asynchronous work as well. In particular, for Async with Code Violations (1553B) transmitting, software can use the TCC and Transmit Control Block features to control which type of Code Violation (Command/Status or Data) to send, for each series of words of the same type. Similarly, 1553B receiving software can use the RCC and Receive Status Block feature to make an external Receive DMA controller store the type of Code Violation after each received word. A later subsection describes these features more fully.

Figures 38 and 39 show the structure of the TCC and RCC features, respectively. Software can write the 16-bit Transmit Count Limit Register (TCLR) at any time, to define the length of the next transmitted message(s) or frame(s). Similarly, it can write the 16-bit Receive Count Limit Register (RCLR) at any time, to define the maximum length of future received messages and frames. Software can also use the Transmit Control Block feature to make a channel automatically fetch a new value for the TCLR and TCC from memory before each block of characters. The TCLR and RCLR can be read back at any time. A channel never changes their values except to clear them to zero at reset time, and when it loads TCLR from a 32-bit Transmit Control Block.

Writing the TCLR or RCLR doesn't have any immediate effect on the TCC or RCC feature. Only when one of several events occurs does a channel load the value from TCLR or RCLR into the actual 16-bit character counter. If the value in TCLR or RCLR is zero at that time, the channel disables the TCC or RCC feature, while if the value is nonzero it enables the feature.

A channel loads the value from the TCLR into the Transmit Character Counter, and enables or disables the TCC accordingly, when one of the following occurs:

1. software writes the Trigger Tx DMA (or Trigger Tx and Rx DMA) command to the RTCmd field of the Channel Command/Address Register (CCAR15-11),
2. software writes the Load TCC (or Load RCC and TCC) command to RTCmd in the CCAR,
3. software writes the Purge Tx FIFO (or Purge Tx and Rx FIFO) command to RTCmd in CCAR, or
4. the TxCtrlBlk field in the Channel Control Register (CCR15-14) is 10, specifying a two-word Transmit Control Block, and an external Transmit DMA controller fetches (the second byte of) the second word containing the new character count. Which is to say, the channel fetches the count "through" the TCLR.

A channel loads the value from the RCLR into the Receive Character Counter, and enables or disables the RCC feature, when any of the following occur:

1. software writes the Trigger Rx DMA (or Trigger Tx and Rx DMA) command to the RTCmd field of the Channel Command / Address Register (CCAR15-11),
2. software writes the Load RCC (or Load RCC and TCC) command to RTCmd in the CCAR,
3. software writes the Purge Rx FIFO (or Purge Tx and Rx FIFO) command to RTCmd in CCAR, or
4. the Receiver detects an opening Flag or Sync character.

Once a channel has loaded the TCC or RCC with a non-zero value (which enables the feature) it decrements the counter for each character/byte written into the associated FIFO. That is, the Transmitter decrements the TCC by 1 or 2 when software or an external Transmit DMA controller loads transmit data into the TxFIFO. The Receiver decrements the

RC Counter by 1 for each character/byte that it transfers from its shift register into the RxFIFO.

A non-zero TCLR value should represent the number of characters to send, not including any Transmit Control Block information. A non-zero RCLR value can be either all ones, or the maximum number of characters/bytes allowed in a message or frame, including any CRC but not including any Receive Status Block information. For applications like 1553B, the RCLR value should simply be the number of characters/bytes between successive Receive Status Blocks. For frame or message-oriented applications in which there's no particular maximum received frame or message length, the all-ones value simplifies computing the length of each frame or message slightly. This value allows software to obtain the frame length by simply ones-complementing the value read from RCCR or from a Received Status Block in memory, rather than by subtracting it from the starting value.

On the Transmit side, software can read the value in the TCC at any time from the Transmit Character Count Register (TCCR), but writing the TCCR address has no effect. Figure 38 shows a decoder that detects when the counter contains 0001. When software or an external Transmit DMA controller writes enough data into the TxFIFO so that the TCC counts down to 0, the channel marks the character that corresponds to decrementing from 1 to 0 as End of Frame / End of Message. When this character gets to the other end of the FIFO, the marking makes the Transmitter conclude the frame appropriately. (Typically, it sends a CRC and a closing Flag or Sync character after the marked character.)

If software or an external Transmit DMA controller writes 16 bits to the TDR while the counter contains 0001, the channel only puts the character on the D7-0 lines into the TxFIFO -- it ignores the data on D15-8. In a system in which even-addressed bytes fall on D7-0 (e.g., a system based on an Intel processor) this isn't a problem. On the other hand, in systems in which even-addressed bytes reside on D15-8 (e.g., a system based on the Zilog Z8000 or 16C0x or a Motorola 680x0) it can cause problems. In such systems, if the last character of a frame falls at an even address, software must copy the last character into the subsequent odd address as well, before presenting the buffer to the Transmit DMA controller. (Typically software needs to do this for an odd-length frame.)

The Transmitter suppresses its DMA request from the time an external Transmit DMA controller places the EOF/EOM character in the TxFIFO until the Transmitter sends it. When software uses the Transmit Control Block feature, this procedure ensures that the Transmit DMA controller doesn't load the control information for the next frame or message, while the Transmitter still needs the values for the current one.

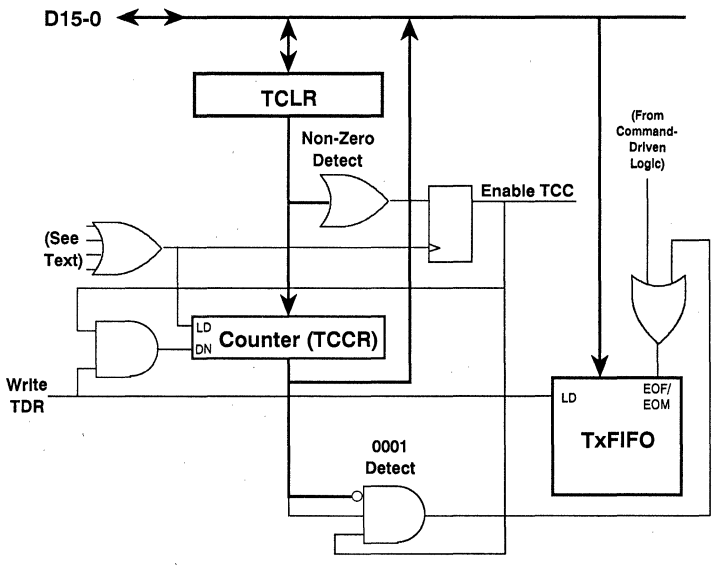


Figure 38. A Model of the Transmit Character Counter Feature

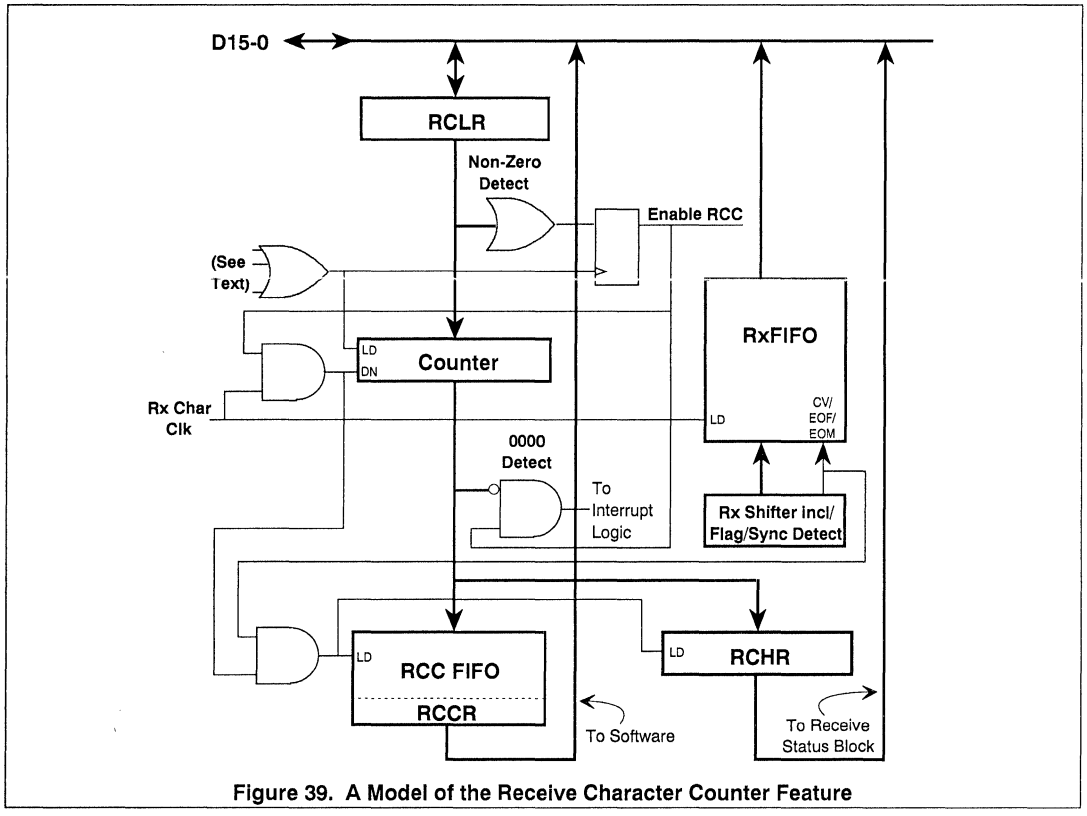


Figure 39. A Model of the Receive Character Counter Feature

On the Receive side, software can't directly read the RCC (except perhaps by using test modes that are beyond the scope of this section). Instead, when the Receiver detects an end-of-frame situation, it captures the decremented value in the counter into a four-entry RCC FIFO and in a register called RCHR. (It may do this when it receives a Flag or Sync character, or, in External Sync and 802.3 modes only, when the /DCD pin goes false.) It then reloads the RCC from RCLR in preparation for the next frame. If software enables two-word Receive Status Blocks, the channel stores the value from RCHR as the second word of the RSB.

Besides recording the length of received frames/messages, the RCC feature can help detect frames or messages that are longer than a maximum length defined by the serial protocol. This typically happens because the Flag, terminating character or Sync character(s) separating two frames or messages gets corrupted on the serial link. This makes the two frames or messages look like a single continuous one to the Receiver. The usual strategy in such a case is to ignore (or possibly "NAK") the whole mess.

If the channel decrements the RCC to zero and then receives another character as part of the same frame/message, it sets the **RCCUnder L/U** bit in the Miscellaneous Interrupt Status Register (MISR3). To use this feature to check for overly long frames or messages, program the RCLR with the maximum number of characters that a frame or message can validly have. This value should include any terminating and CRC characters but exclude any Receive Status Block information. Also, arm the RCC Underflow interrupt by setting the RCCUnder IA bit in the Status Interrupt Control Register (SICR3), as described in Chapter 6.

If the channel ever sets RCCUnder L/U and interrupts, clear the condition by writing a 1 to the L/U bit, discard the data received for the frame(s) by purging the RxFIFO, reprogram the external Receive DMA controller if one is being used, and do whatever else is necessary to clean up the situation. Then write the "Enter Hunt Mode" command to the RCmd field of the Receive Command/Status Register (RCSR15-12).

The RCC FIFO

Figure 39 shows the RCC FIFO. When software has enabled the Receive Character Counter, the FIFO captures the contents of the RCC at the end of each frame or message in External Sync, Transparent Bi-sync, 802.3, and HDLC/SDLC modes. (The previous section described how the Receiver decrements the RCC by one for each character it receives.)

The RCC FIFO can hold up to four 16-bit entries. Figure 40 shows the Channel Command/Status Register (CCSR), the 3 MSBits of which allow software to monitor and control the RCC FIFO. The **RCCFAvail** bit (CCSR14) is 1 if the RCC FIFO contains at least one entry, or is 0 if the RCC FIFO is empty.

When RCCFAvail is 1, software can read the oldest entry in the RCC FIFO from the Receive Character Count Register (RCCR). It can then compute the length of the frame or message by subtracting this ending value from the starting value that came from the Receive Count Limit Register (RCLR). (Or, if the starting value was all ones, software can simply one's complement the value from RCCR.) Reading the RCCR removes the oldest entry from the RCC FIFO.

For internal synchronization reasons, a channel does not set RCCFAvail for one bit time after it sets other status related to an End of Frame condition. That is, it sets these bits on the rising edge of RxCLK after the rising edge of RxCLK on which it places the RxBound character in the RxFIFO, and from which it may force a Receive Data interrupt or a Receive DMA request.

If software has enabled the RCC, and a frame or message ends when the RCC FIFO is already full, the new value overwrites its predecessor, and the three oldest entries are not affected. The channel remembers this event in a status bit that it routes through the RCC FIFO, much like it routes other status bits through the RxFIFO. When software reads the preceding entries so that an overwriting/overwritten entry becomes the oldest one left in the RCC FIFO, the channel sets the **RCCFOvflo** bit in the Channel Command/Status Register (CCSR15). Once RCCFOvflo is set, the only way to clear it (other than to Reset the whole channel) is to write a 1 to the **ClearRCCF** bit (CCSR13). This also empties the RCC FIFO and clears RCCFAvail.

Writing to the RCCFOvflo and RCCFAvail bits has no effect, nor does writing a 0 to the ClearRCCF bit. ClearRCCF always reads as 0.

Transmit Control Blocks[6]

Figure 41 shows the Channel Control Register. Its **TxCtrlBlk** field (CCR15-14) controls what the Transmitter does with the first 16 or 32 bits of data that an external Transmit DMA controller fetches from memory at the start of a frame or message. (While software can use Transmit Control Blocks when it fills the TxFIFO, there's no obvious reason to do so, compared to just writing the control registers directly.) The Transmitter interprets TxCtrlBlk as follows:

TxCtrlBlk	Kind of TCB's used
00	No Transmit Control Block
01	16-bit Transmit Control Block
10	32-bit Transmit Control Block
11	Reserved; do not program

When TxCtrlBlk is 01 or 10, a channel treats the next 16 or 32 bits, that software or an external Transmit DMA controller writes to the TDR, as a Transmit Control Block after any of following happen:

1. after software writes the Trigger Tx DMA (or Trigger Tx and Rx DMA) command to the RTCmd field of the Channel Command / Address Register,
2. after software writes the Load TCC (or Load RCC and TCC) command to RTCmd,
3. after software writes the Purge Tx FIFO (or Purge Tx and Rx FIFO) command to RTCmd, or
4. after the Transmit DMA controller (or software) writes data into the TxFIFO that decrements the TCC to zero. As noted earlier, the Transmitter drops its DMA request from the time the DMA controller fetches the last character of a frame, until after it moves the character to its shift register. It does this so that the DMA controller doesn't fetch the Transmit Control Block for the next frame or message, while the Transmitter still needs the control information for the current frame.

Figure 42 shows a 32-bit Transmit Control Block as part of a sequence of 16-bit words in memory. The MS 4 bits of the first word (or the only word in a 16-bit TCB) define a new TxSubMode value for the following transmit data. A channel writes these bits into the TxSubMode field of its Channel Mode Register (CMR15-12) without changing the rest of the CMR. Bits 4-2, of the first or only word, define the TxResidue value for the following frame in HDLC/SDLC or HDLC/SDLC Loop mode. The channel writes these bits into the TxResidue field of the Channel Command/Status Register (CCSR4-2) without affecting the rest of the CCSR. The channel ignores bits 11-5 and 1-0 of the first or only word, but Zilog reserves these bits for future enhancements and software should ensure that they're all zero.

---

[6] Previous USC documentation called these Transmit Status Blocks.

RCCF Ovflo	RCCF Avail	Clear RCCF	DPLL Sync	DPLL 2Miss	DPLL 1Miss	DPLLEDGE	On Loop	Loop Send	Resrvd	TxResidue	/TxACK	/RxACK
15	14	13	12	11	10	9 8	7	6	5	4 3 2	1	0

Figure 40. The Channel Command/Status Register (CCSR)

TxCtrlBlk	Wait4 Tx Trig	Rsrvd	Async:TxShaveL		RxStatBlk	Wait4 Rx Trig	Reserved (0)
			Sync:TxPreL	Sync:TxPrePat			
15 14	13	12	11 10	9 8	7 6	5	4 3 2 1 0

Figure 41. The Channel Control Register (CCR)

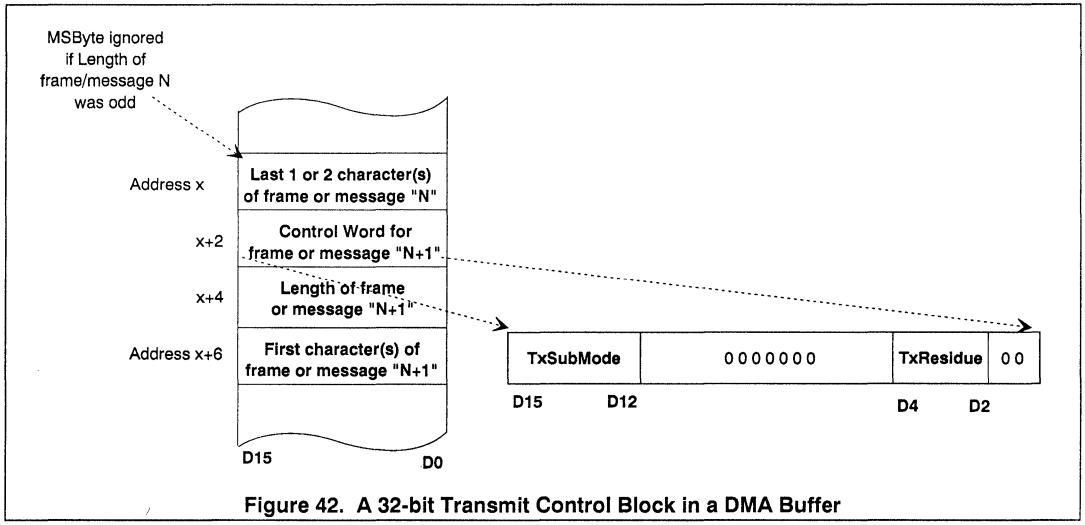


Figure 42. A 32-bit Transmit Control Block in a DMA Buffer

A channel transfers the second word of a 32-bit TCB through the Transmit Count Limit Register (TCLR) and into the TC Counter (TCCR). Therefore this word should contain the number of characters/bytes that follow this TCB, until the end of the frame or message. In a non-block-structured mode like 1553B, the value simply reflects the number of bytes until the next TCB. Note that with a 16-bit TCB, the channel still reloads the TC Counter, but it uses the old value in TCLR to do so. Thus, 16-bit TCBs are useful in protocols that use fixed-length frames or messages, but 32-bit TCBs should be used when successive transmitted frames or messages can vary in length.

Figure 42 shows a TCB in the middle of a memory buffer, that is, directly following the last characters of the previous frame. Perhaps more typically, the TCB would be the first two or four bytes of a memory buffer dedicated to this frame or message.

Receive Status Blocks

The Receiver sets the RxBound bit in the RxFIFO to indicate the end of a frame, message, or word, in External Sync, Transparent Bisync, 802.3, HDLC/SDLC, and ACV/1553B modes. In these modes the Receiver can store summary/status information in memory after the last character of the frame, message, or word. The **RxStatBlk** field of the Channel Control Register (CCR7-6) controls whether it does this. A channel interprets it like TxCtrlBlk:

```
RxStatBlk  Kind of RSB's used
      00   No Receive Status Block
      01   16-bit Receive Status Block
      10   32-bit Receive Status Block
      11   Reserved; do not program
```

If this field is either 01 or 10, the Receiver stores the status from the Receive Command / Status Register (RCSR) after the frame. The 10 value makes the channel also store the ending value of the Receive Character Counter in a second 16-bit word after the RCSR status word. (For what it's worth, the channel temporarily stores the RCC value in a holding register called RCHR.) Figure 43 shows a 32-bit RSB.

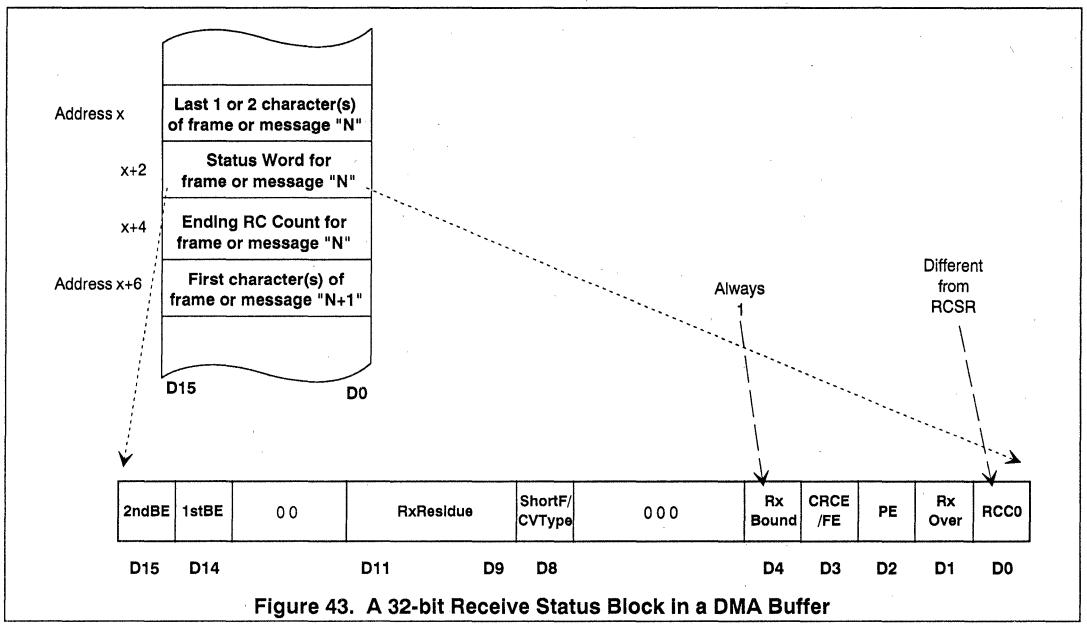


Figure 43. A 32-bit Receive Status Block in a DMA Buffer

The only trouble with the 32-bit RSB option is that software has to know how long each received frame is, in order to find the RSB that indicates the length. (This is somewhat reminiscent of trying to follow a forward-linked-list backward.) Typically software will need to use the RCC FIFO to keep track of frame lengths instead of, or in addition to, Receive Status Blocks.

Figure 43 shows the contents of the first word of a Receive Status Block; they are identical with the contents of the RCSR with the following exceptions:

1. The channel forces the bits that correspond to ExitedHunt, IdleRcved, and Break/Abort in the RCCR to 0. These are "global" rather than "queued" status bits and must be handled by software on a more or less real-time basis.
2. The LSBit of the first word of an RSB is a copy of the LSBit of the RCC at the end of the frame, rather than the RxAvail bit that's in the RCCR. This bit is also available in the RCC FIFO and in the second word of a 32-bit RSB, but for 16-bit DMA operation it may be handy to have it here, especially in a 16-bit RSB.

The CRCE/FE bit in an RSB reflects the CRC-correctness of the frame in 802.3 and HDLC/SDLC modes, but not in Transparent Bisync mode.

When software or an external Receive DMA controller reads 16 bits from the RDR, and the Receiver has marked the oldest character in the RxFIFO as End of Frame or End of Message, the channel only takes that one character out of the RxFIFO. When the Receive DMA controller is doing 16-bit transfers, software has several ways to figure out whether the 16-bit "word" preceding a RSB contains one or two characters/bytes.

The most straightforward way is to compute the length of the frame or message, by subtracting the ending RCC value in the RCC FIFO or the second word of the RSB, from the starting RCC value that the hardware took from RCLR. (If the starting value was all ones, software can just ones-complement the ending value.) If the result is odd there's one character in the 16-bit word that precedes the RSB, while if it's even there are two characters in the word.

A "narrower" version of the same computation is that if bit 0 of the first or second word of the RSB is the same as the units bit of the starting RCC value that came from RCLR, then the preceding word contains two characters. If the two bits are different the word contains only one character.

Still another method applies only when bits 2-1 of the first word of the RSB, namely Parity Error and Rx Overrun, are both 0. (Most "modern" protocols don't use parity checking anyway.) The usual handling for an Rx Overrun condition in synchronous modes involves forcing the receiver into Hunt mode for the start of the next frame or message, which means that an RSB would never be stored for a frame that encountered an overrun. When PE and RxOver are both zero, if bit 14 of the first word of the RSB (1stBE) is 1, there is one character in the preceding word, while if bit 14 is 0 there are two characters.

Figure 43 shows the first characters of the next frame stored right after the RSB. This indicates that the DMA controller didn't switch memory buffers between the frames.

Using TCB's and RSB's in ACV (1553B) Mode

In Async with Code Violations (1553B) mode, the Receiver sets the RxBound bit for the second (or only) byte of each word received. It does this so that software can use the Receive Status Block mechanism to record the type of Code Violation (Command/Status or Data) that introduced each word. To use this facility, software should program the RxStatBlk field (CCR7-6) to 01 to select 16-bit RSB's. The Receiver then stores a 16-bit status word after each word (or byte) of received data. The ShortF/CVType bit (bit 8) of the status word is 1 after a "command/status" word and 0 after a "data" word.

On the Transmit side, software can use Transmit Control Blocks to send any sequence of mixed Command/Status and Data words under DMA control. To do this, it should program TxCtrlBlk (CCR15-14) to 10 to select 32-bit TCB's, and should structure the data in memory so that a TCB precedes each block of words of the same kind. Bit 12 of the first word of each TCB (the LSBit of the TxSubMode value) should be 1 for a block of Command/Status words and 0 for a block of Data words. The second word of each TCB should specify the number of bytes in the block (typically, this is twice the number of words).

Commands

Commands are encoded values that software writes to a register field to change the state of a channel or make it perform some action. Typically commands don't take any software-perceptible time to perform. USC command fields are write-only; reading them back may yield zeroes or some unrelated status item.

Often commands represent a more compact and efficient way to provide control features than dedicated register bits. In fact, commands are so popular that each USC channel includes three separate encoded command fields! Figure 44 shows the Channel Command/Address Register. Software can write any of 18 different commands that affect the Transmitter and/or the Receiver to its **RTCmd** field (CCAR15-11). In addition, software can write any of ten commands that affect the Transmitter to the **TCmd** field in the Transmit Command/Status Register (TCSR15-12).

Finally, software can write any of five commands that affect the Receiver to the **RCmd** field in the Receive Command/Status Register (RCSR15-12).

Writing all zeroes to any of the command fields does nothing, which can be useful when the intent is to write to other fields of the register. Zilog reserves other values not listed below for future extensions to the USC family; such values should not be written to the subject field.

RTCmd Value	Function
00010	Reset Highest Serial IUS
00100	Trigger Channel Load DMA
00101	Trigger Rx DMA
00110	Trigger Tx DMA
00111	Trigger Rx and Tx DMA
01001	Purge Rx FIFO
01010	Purge Tx FIFO
01011	Purge Rx and Tx FIFO
01101	Load RCC
01110	Load TCC
01111	Load RCC and TCC
10001	Load TC0
10010	Load TC1
10011	Load TC0 and TC1
10100	Select Serial LSBit First
10101	Select Serial MSBit First
10110	Select D15-8 First
10111	Select D7-0 First

TCmd Value	Function
0010	Clear Tx CRC Generator
0101	Select TICRHi=FIFO Status
0110	Select TICRHi=/INT Level
0111	Select TICRHi=/TxREQ Level
1000	Send Frame/Message
1001	Send Abort
1100	Enable DLE Insertion
1101	Disable DLE Insertion
1110	Clear EOF/EOM
1111	Set EOF/EOM

RCmd Value	Function
0010	Clear Rx CRC Generator
0011	Enter Hunt Mode
0101	Select RICRHi=FIFO Status
0110	Select RICRHi=/INT Level
0111	Select RICRHi=/RxREQ Level

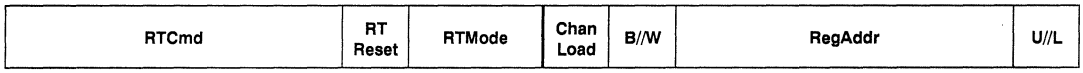


Figure 44. The Channel Command/Address Register (CCAR)

A description of each command follows, in alphabetical order. Some of them include references to other chapters or sections, which provide more information that's important to fully understanding the command.

Clear EOF/EOM (TCmd:=1110): this command conditions a channel so that it doesn't mark the next character that software or an external Transmit DMA controller writes to the Transmit Data Register as End of Frame/End of Message. Since a channel assumes this state after each write to the TDR, and after a hardware or programmed Reset, software will need this command only if it "changes its mind" about where the frame ends, between issuing a Set EOF/EOM command and writing the TDR.

Clear Rx or Tx CRC Generator[7] (RCmd or TCmd:= 0010): these commands force the Receive or Transmit CRC Generator to all zeroes or all ones, depending on the RxCRCStart bit in the Receive Mode Register (RMR10) or the TxCRCStart bit in the Transmit Mode Register (TMR10). Software will seldom need these commands because the Receiver and Transmitter automatically clear their associated CRC generators at the start of each frame.

Disable DLE Insertion (TCmd:=1101): this command applies only to Transparent Bisync mode. It conditions a channel so that it doesn't check subsequent characters written to the Transmit Data Register (TDR) for DLE characters, and so that it doesn't add any DLE characters to the transmitted data stream. Software should use this command before writing a two-character control sequence that starts with DLE to the TDR. DLE insertion remains disabled until software issues the Enable DLE Insertion command or until a hardware or software Reset. Each channel queues the state that's affected by this and the following command through its TxFIFO with each character, so that software can change the state as needed.

Enable DLE Insertion (TCmd:=1100): this command applies only to Transparent Bisync mode. It conditions a channel so that it checks subsequent characters written to the Transmit Data Register (TDR) for DLE characters, and adds another DLE for each DLE written to the TDR. Software should use this command before writing normal data to the TDR. DLE insertion remains enabled until software issues the Disable DLE Insertion command. Each channel queues the state that's affected by this and the preceding command through its TxFIFO with each character, so that software can change the state as needed.

Enter Hunt Mode (RCmd:=0011): this command forces the Receiver into "Hunt Mode" immediately, regardless of its previous state. In synchronous modes, this means that the Receiver starts searching for a Sync or Flag sequence. In asynchronous modes it starts searching for a start bit or (in 1553B mode) for a code violation. In any mode, the Receiver discards any partial character that was in progress when software issued the command.

Load RCC and/or TCC[8] (RTCmd:=01101-01111): these commands load the Receive and/or Transmit Character Counter from the Receive and/or Transmit Count Limit Register (RCC from RCLR and/or TCC from TCLR). This may enable or disable character counting. If software has enabled the Transmit Control Block feature in the TxCtrlBlk field of the Channel Control Register (CCR15-14=01 or 10), a Load TCC or Load RCC and TCC command also conditions the Transmitter to treat the next data written to the Transmit Data Register as a TCB.

Load TC0 and/or TC1 (RTCmd:=10001-10011): these commands load the counter in Baud Rate Generator 0 and/or 1 from the Time Constant 0 and/or 1 Register (BRG0 from TC0R and/or BRG1 from TC1R). Loading a BRG via one of these commands also enables it to count. This is particularly important when software has programmed a BRG for single cycle mode (HCR1=1 for BRG0 or HCR5=1 for BRG1) and it has stopped after counting down to zero. See Chapter 3 for more information about the BRG's.

Purge Rx and/or Tx FIFO (RTCmd:=01001-01011): these commands remove all entries from the RxFIFO and/or TxFIFO. They also reload the Receive and/or Transmit Character Counter from the Receive and/or Transmit Count Limit Register (RCC from RCLR and/or TCC from TCLR). This may enable or disable character counting. If software has enabled the Transmit Control Block feature in the TxCtrlBlk field of the Channel Control Register (CCR15-14=01 or 10), a Purge Tx FIFO command also conditions the Transmitter to treat the next data written to the Transmit Data Register as a TCB. If software is using an external Transmit DMA channel, a Purge Tx FIFO command may cause the /TxREQ pin to be asserted immediately, while if it's using Transmit Data interrupts, the command may cause the /INTA or /INTB pin to be asserted immediately. (The previous two sentences also apply to a Purge Rx and Tx FIFO command.)

---

[7] Previous USC documentation called these commands Preset CRC

[8] Previous USC documentation called these commands Reload TCC and/or RCC

Reset Highest IUS (RTCmd:=00010): Chapter 6 describes how this command clears the highest-priority Interrupt Under Service latch in the channel that's currently set (if any).

Select D15-8 or D7-0 First[9] (RTCmd:=10110-10111): these commands control which of the two characters in a 16-bit write to the TDR/TxFIFO the Transmitter sends first. They also control how the channel arranges the oldest and second-oldest characters in the RxFIFO when software or an external Receive DMA controller reads 16 bits from it via the Receive Data Register. "D15-8 First" is the default value after either a hardware or programmed reset, and is compatible with the Zilog Z8000, Zilog 16C0x and Motorola 680x0 processors. "D7-0 First" should be programmed for the Zilog Z380 and most Intel processors. A channel applies this option only during a 16-bit transfer, between the TxFIFO or RxFIFO and the AD15-0 pins. However, if the Transmit Character Counter contains 0001 and the Transmit DMA controller writes 16 bits to the TxFIFO, the channel only puts the character from AD7-0 in the TxFIFO, regardless of these commands. In a "D7-0 First" system this isn't a problem. But if the last character of a frame or message falls at an even address when using the Transmit DMA controller in a "D15-8 First" system, software must copy the last character into the subsequent odd address as well. (Usually this applies to a frame with an odd length.)

Select RICRHi=/INT Level (RCmd:=0110): this command conditions a channel so that subsequent accesses to the MSByte of its Receive Interrupt Control Register (RICR15-8) read or write the number of received characters at which the channel starts requesting a Receive Data interrupt, as described in Chapter 6. If software uses a Receive DMA controller to store data in memory, it should disable Receive Data interrupts.

Select RICRHi=/RxREQ Level (RCmd:=0111): this command conditions a channel so that subsequent accesses to the MSByte of its Receive Interrupt Control Register (RICR15-8) read or write the number of received characters at which the Receiver asserts /RxREQ to a Receive DMA controller, as described in Chapter 5.

Select RICRHi=FIFO Status (RCmd:=0101): this command conditions a channel so that reading the MSByte of its Receive Interrupt Control Register (RICR15-8 yields the number of characters in its RxFIFO. This is described more fully in *The Data Registers and the FIFOs* later in this chapter.

Select Serial Data LSB or MSB First (RTCmd:= 10100-10101): these commands control whether a channel transmits and assembles serial data with the Least Significant or Most Significant bit going first on the line. "LSB first" is the default after either a hardware or programmed reset, and is the method used in most traditional data communications schemes. A channel applies this option as it transfers data between the AD pins and the FIFOs. Because of this, these commands don't affect functions like matching addresses and sync characters and sending syncs. This, in turn, means that software must program such values "backward" in the TSR and RSR for "MSB first" applications.

Select TICRHi=/INT Level (TCmd:=0110): this command conditions a channel so that subsequent accesses to the MSByte of its Transmit Interrupt Control Register (TICR15-8) read or write the number of empty TxFIFO entries at which the Transmitter starts requesting a Transmit Data interrupt, as described in Chapter 6. If software uses a Transmit DMA controller to fetch data from memory, it should disable Transmit Data interrupts.

Select TICRHi=/TxREQ Level (TCmd:=0111): this command conditions a channel so that subsequent accesses to the MSByte of its Transmit Interrupt Control Register (RICR15-8) read or write the number of empty TxFIFO entries at which the Transmitter asserts /TxREQ to a Transmit DMA controller, as described in Chapter 5.

Select TICRHi=FIFO Status (TCmd:=0101): this command conditions a channel so that reading the MSByte of its Transmit Interrupt Control Register (TICR15-8) yields the number of empty entries in its TxFIFO. This is described more fully in *The Data Registers and the FIFOs* later in this chapter.

Send Abort (TCmd:=1001): this command is valid only in HDLC/SDLC mode and makes the Transmitter send an Abort (Go Ahead) sequence. If the 2 MSBits of the TxSubMode field of the Channel Mode Register (CMR15-14) are 01, the Abort consists of a zero followed by 15 consecutive ones. Otherwise it consists of a zero followed by seven ones. After sending the Abort, the Transmitter operates as it would have after sending a closing Flag. That is, if Wait2Send (TICR2) is 0 and there's data in the TxFIFO, it starts a new frame, otherwise it sends the Idle condition defined by the TxIdle field (TCSR10-8).

Send Frame/Message (TCmd:=1000): if the Wait2Send bit in the Transmit Interrupt Control Register (TICR2) is 1, the Transmitter waits between frames, sending the Idle pattern defined by the TxIdle field of the Transmit Command/Status Register (TCSR10-8), until software issues this command. The later section *Synchronizing Frames/Messages with*

---

9 Previous USC documentation called these commands Select Straight Memory Data and Select Swapped Memory Data.

Software Response describes how this feature differs from the one controlled by the Wait4TxTrig bit in the Channel Control Register and the Trigger Tx DMA command in RTCmd.

Set EOF/EOM (TCmd:=1111): this command conditions a channel so that it marks the next character, that software or an external Transmit DMA controller writes to the Transmit Data Register (TDR), as End of Frame/End of Message. This marking makes the Transmitter perform the appropriate closing actions after sending the character. (For example, in HDLC/SDLC mode it sends a CRC and then a closing Flag.) Typically, after issuing this command, software should write the last character of the frame or message to the LSByte of the Transmit Data Register (TDR7-0). The channel automatically clears the state set by this command when software (or a Transmit DMA controller) writes to the TDR. Therefore this command applies to at most one character.

Trigger Channel Load DMA (RTCmd:=00100): Chapter 7 will describe how this command puts a channel in a special mode in which an external Transmit DMA controller can initialize all the registers in the channel. Software must program and set up an external Transmit DMA controller as for transmitting data, before it issues this command.

Trigger Rx and/or Tx DMA (RTCmd:=00101-00111): if one of the Wait4xxTrig bits in a channel's Channel Control Register (CCR13 for Tx, CCR5 for Rx) is 1, the channel stops requesting that kind of DMA transfer after the end of each frame. When this happens, software should use one of these commands to re-enable requests to the external DMA controller(s), for the next frame. These commands also load the Receive and/or Transmit Character Counter from the Receive and/or Transmit Count Limit Register (RCC from RCLR and/or TCC from TCLR). This may enable or disable character counting. If software has enabled the Transmit Control Block feature in the TxCtrlBlk field of the Channel Control Register (CCR15-14=01 or 10), a Trigger Tx DMA or Trigger Tx and Rx DMA command also conditions the Transmitter to treat the next 16 or 32 bits written to the Transmit Data Register as a TCB. The later section *Synchronizing Frames/Messages with Software Response* describes how this feature differs from the one controlled by the Wait2Send bit in the Transmit Interrupt Control Register and the "Send Frame/Message" command in TCmd.

Resetting a USC Channel

Figure 44 shows the **RTReset** bit in the Channel Command/Address Register (CCAR10). Software can use this bit to reset a channel to a known and inactive state like that produced by driving the /RESET pin low. (The most significant difference is that the USC requires software to write the Bus Configuration Register (BCR) after a hardware reset, but not after this kind of "software Reset".)

To software-reset a channel when using a 16-bit data bus:

1. Write CCAR (or its MSByte) with RTReset=1.
2. Write a 16-bit zero to CCAR.

To software-reset a channel when using an 8-bit bus:

1. Write the MSByte of CCAR with RTReset=1.
2. Write the LSByte of CCAR with an 8-bit zero.
3. Write the MSByte of CCAR with an 8-bit zero.

The way this "software reset" works is that the 1 state of RTReset conditions the channel's register address decoding logic so that the subsequent write operation actually writes data into all the registers in the channel. Between the time that software writes RTReset as 1, and when it writes it back to 0, the channel doesn't drive I/O pins, it either 3-states output pins or holds them in their inactive state, but register bits that don't directly affect these pins are unchanged/undefined.

Leaving the RTReset bit set is a common mistake made by first-time users of a USC family member.

The Data Registers and the FIFOs

When the RxFIFO contains received characters, software can read the "oldest" 1 or 2 characters in it from the Received Data Register (RDR). When software uses an external Receive DMA controller, it takes care of taking data out of the RxFIFO. *The Mode Registers: Character Length,* earlier in this Chapter, describes how the Receiver aligns characters and fills out bytes in the RDR/RxFIFO when characters are less than 8 bits long.

Similarly, when the TxFIFO isn't full software can write 1 or 2 characters to it via the Transmit Data Register (TDR), or an external DMA controller can do so.

Chapter 2 describes how software can access the TDR and RDR using a register address that may be 1) multiplexed on the AD5-1 pins, 2) full-time on AD13-8 if only AD7-0 carry data, or 3) written into the Channel Command / Address Register (CCAR5-1).

Two other features of the USC make it easier for software to access these registers when the AD lines don't carry multiplexed addresses and the data bus is 16 bits wide. Host processor write cycles to the USC, with the D//C pin high, always write the TDR. Similarly, host processor read cycles from the USC, with D//C high, always read the RDR. A system designer may connect D//C to a processor address line, such as A1 for a non-multiplexed 16-bit bus or A7 for a multiplexed bus.

Chapter 2 also describes how to write the Bus Configuration Register to configure the USC for a 16-bit data

bus. With a 16-bit data bus, software can write two characters at once to the TDR, or an external Transmit DMA controller can read two characters from memory at once. Similarly, software can read two characters at a time from the RDR, or an external Receive DMA controller can write two characters into memory in each bus cycle. The earlier section *Commands* describes how the "Select D15-8 First" and "Select D7-0 First" commands allow the two characters in each 16-bit transfer, to the TDR or from the RDR, to be arranged in either order. This is important because available microprocessors differ about the order.

With a 16-bit data bus, software can read or write most USC registers as a 16-bit word, or can read or write either their "more significant" byte (bits 15-8) or "less significant" byte (bits 7-0). The TDR and RDR are different in this regard: software should never read or write their more significant bytes alone, only as part of a 16-bit transfer. On a Zilog Z8000 or 16C0x or Motorola 680x0 based system this typically means that software should write bytes to the TDR and read bytes from the RDR at an odd address. On an Intel 80x86 processor software should typically write bytes to the TDR and read bytes from the RDR at an even address.

On a 16-bit bus there's no way for software to read single characters from RDR, or write single characters to TDR, using an address that makes D//C high. To do this, software must either address the LSByte of TDR/RDR directly, or it must write the address of the LSByte to the CCAR.

The TxFIFO and RxFIFO have a maximum capacity of 32 characters (bytes) each. A USC channel empties them of all data when external hardware drives the /RESET pin low, when software resets the channel via the RTReset bit (CCAR10), and when software writes a "Purge Rx and/or Tx FIFO" command to the RTCmd field (CCAR15-11).

The RxFIFO becomes one byte more full for each character received on the serial link, and one or two bytes less full each time software or an external Receive DMA controller reads data from it via the RDR. The TxFIFO becomes one or two bytes more full each time software or an external Transmit DMA controller writes data to it via the TDR, and one byte less full each time the Transmitter moves a character into its output shift register.

The exceptions to the above statements are that in Async with Code Violations (1553B) mode with the Extended Word option selected, the RxFIFO becomes two bytes more full for each received word, and the TxFIFO becomes two bytes emptier each time the Transmitter transfers a word to its shift register.

Each channel maintains a counter for each FIFO that reflects its current contents. Software can read the number of received characters/bytes that are currently in the RxFIFO. To do this, it may first have to write the "Select RICRHi=FIFO Status" command to the RCmd field of the Receive Command/Status Register (RCSR15-12). Then software can read the MSByte of the Receive Interrupt Status Register (RICR15-8). The resulting 8-bit value represents the number of received characters in the RxFIFO. It ranges from 0 for an empty RxFIFO to 32 for a full one. Software can skip the step of writing the Select command if it hasn't written any of the other "Select RICRHi=..." commands to the RCSR since the last time it issued this command.

Similarly, software can read the number of entries that are currently empty in the TxFIFO. It may first have to write the "Select TICRHi=FIFO Status" command to the TCmd field of the Transmit Command/Status Register (TCSR15-12). Then software should read the MSByte of the Transmit Interrupt Status Register (TICR15-8). The resulting 8-bit value represents the number of empty positions in the TxFIFO. It ranges from 0 for a full TxFIFO to 32 for an empty one. As for the RxFIFO, software can skip the step of writing the Select command if it hasn't written any of the other "Select TICRHi" commands to the TCSR since the last time it issued this command.

The USC channels continually compare the contents of these counters against two "threshold" levels for each. Chapter 5 describes how the "Tx DMA Request Level" determines how empty the TxFIFO must get before the Transmitter starts requesting that an external Transmit DMA controller should read more data from memory. Once the Transmitter has started to request DMA transfer, it typically keeps doing so until the DMA controller has filled the TxFIFO or until the Transmit Character Counter has counted down to zero.

Chapter 5 also describes how the "Receive DMA Request Level" controls how full the RxFIFO should get before the Receiver starts requesting that an external Receive DMA controller should move data to memory. Once the Receiver has started to request DMA transfer, it typically keeps doing so until the DMA controller has emptied the RxFIFO, or until it has stored the last character of a frame or message.

Chapter 6 describes how, if software enables "Transmit Data" interrupts, the "Transmit /INT Level" controls how empty the TxFIFO should get before the Transmitter starts requesting such an interrupt. It also describes how, if software enables "Receive Data" interrupts, the "Receive /INT Level" controls how full the RxFIFO should get before the Receiver starts requesting such an interrupt. Software doesn't use these kinds of interrupts in USC applications in which external Transmit and Receive DMA controllers handle the data. But if software does use data interrupts, the

interrupt service routine should fill the TxFIFO or empty the RxFIFO completely each time it executes. (As a minimum the ISR should transfer enough data to bring the FIFO status below the threshold level, or should raise the threshold level to accomplish the same thing.)

Between Frames, Messages, or Characters

Synchronous Transmission

When software issues a "Set EOF/EOM" command and then writes data to a channel's TDR, or when software or an external Transmit DMA controller fetches enough data so that the TCC counts down to zero, the channel flags the last character of the message or frame in the TxFIFO. After this last character gets to the other end of the TxFIFO and out onto the serial link, the Transmitter terminates the frame or message. The Transmitter also terminates a frame or message if it needs a character from the TxFIFO but it's empty (an "underrun" condition). The Transmitter's exact actions at these points depend on the serial mode/protocol and perhaps on certain programmed options.

If the TxCRCatEnd bit in the Transmit Mode Register (TMR8) is 1, the Transmitter sends the CRC code it has accumulated during the frame, after a character marked as the end of a frame or message. If the TxSubMode field says to do so, the Transmitter sends its accumulated CRC in an underrun situation. The CRC can be 16 or 32 bits long.

Then, or right after the last character from the TxFIFO if it doesn't send the CRC, except in 802.3 (Ethernet) mode the Transmitter sends a closing Sync or Flag sequence as determined by the TxMode and sometimes the TxSubMode, as follows:

TxMode	Closing sequence:
Monosync	(TSR15-8)
Slaved Monosync	(TSR15-8)
Bisync	(TSR15-8) if CMR14=0 (TSR7-0)(TSR15-8) if CMR14=1
Transparent Bisync	SYN if CMR14=0 DLE-SYN if CMR14=1 (ASCII or EBCDIC per CMR12)
802.3 (Ethernet)	None
HDLC/SDLC	Flag (01111110)
HDLC/SDLC Loop	Flag (01111110)

Then, or immediately after sending the CRC in 802.3 (Ethernet) mode, the Transmitter decides whether to send another frame or message immediately or not. In HDLC/SDLC Loop mode only, when it sends a closing or idle Flag the Transmitter checks whether software has cleared the CMR13 bit to signal the end of sending activity. If so, it returns to repeating data from RxD onto TxD. In any other mode, and in Loop mode if CMR13 is 1, the Transmitter commits to sending a new message or frame when:

1. there is at least one character in the TxFIFO, and

2a. either the Wait2Send bit in the Transmit Interrupt Control Register (TICR2) is 0, or

2b. software has written the "Send Frame/Message" command to the TCmd field of the Transmit Command/Status Register (TCSR15-12) since the end of the last frame.

If these conditions aren't met, the Transmitter sends the "Idle line condition" specified by the **TxIdle** field of the Transmit Command/Status Register (TCSR10-8). This field also determines what the Transmitter sends between characters in async modes. The Transmitter interprets TxIdle as follows:

```
TxIdle   Idle Line Condition
   000   The idle line condition is the
         default for the mode/protocol
         defined by TxMode:
         * All ones in 802.3 and all
         async modes.
         * Flags in HDLC/SDLC and
         HDLC/SDLC Loop.
         * Sync sequences in Monosync,
         Slaved Monosync, Bisync, and
         Transparent Bisync.  (In the
         Bisync modes these are like
         closing Syncs: they may be
         single characters or pairs
         based on CMR14.)
   001   Alternating zeroes and ones
   010   Continuous zeroes
   011   Continuous ones
   100   Reserved; do not program
   101   Alternating Mark and Space
   110   Continuous Space (TxD low)
   111   Continuous Mark (TxD high)
```

With choices 000-011, the Transmitter encodes the idle condition as specified by the TxEncode field of the Transmit Mode Register (TMR15-13), while for choices 101-111 it doesn't encode the condition. Software can use these idle-condition options to keep Phase Locked Loop and decoding circuits at the remote receiver "in sync" between messages, frames, or async characters. Consider the sections of Chapter 3 that deal with data encoding and the DPLL, and whatever standards or specifications apply to your application, in selecting how to program TxIdle.

In sync modes, once the conditions to start sending a message or frame (described above) are met, the Transmitter may send a bit sequence called a Preamble. A Preamble can be used to synchronize Phase Locked Loop and decoding circuits at the remote receiver. Whether the Transmitter sends a

Preamble is a function of the TxMode and sometimes the TxSubMode, as follows:

TxMode	Preamble sent?
Monosync	If CMR13=1
Slaved Monosync	Never
Bisync	If CMR13=1
Transparent Bisync	If CMR13=1
802.3 (Ethernet)	Always
HDLC/SDLC	If CMR13=1
HDLC/SDLC Loop	Never

If the Transmitter sends a Preamble, the **TxPreL** and **TxPrePat** fields of the Channel Control Register (CCR11-10 and CCR9-8) control its length and content:

TxPreL	Length of Preamble Sent
00	8 bits
01	16 bits
10	32 bits
11	64 bits

TxPrePat	Preamble Pattern Sent
00	All zeroes
01	All ones
10	101010...
11	010101...

For 802.3 (Ethernet) mode, the proper values are TxPreL=11 and TxPrePat=10; the Transmitter automatically modifies the last (64th) bit from a 0 to a 1 to act as the "start bit". For other modes, consider the sections of Chapter 3 that deal with data encoding and the DPLL, and whatever standards or specifications apply to your application, in deciding whether to use a preamble and if so what kind.

After sending the Preamble, or when the conditions for starting a frame have been met if there is no Preamble, except in 802.3 (Ethernet) mode the Transmitter sends an opening Flag or Sync sequence. In the two Bisync modes this may differ from the closing sequence:

TxMode	Opening sequence:
Monosync	(TSR15-8)
Slaved Monosync	(TSR15-8)
Bisync	(TSR7-0)(TSR15-8)
Transparent Bisync	DLE-SYN (ASCII or EBCDIC per CMR12)
802.3 (Ethernet)	None
HDLC/SDLC	Flag (01111110)
HDLC/SDLC Loop	Flag (01111110)

In the HDLC/SDLC and HDLC/SDLC Loop modes only, the Transmitter will combine the closing and opening Flags into a single instance if all of the following are true:

1. software has not selected sending a Preamble (CMR13=0; this doesn't apply in Loop mode),
2. the Wait2Send bit (TICR2) is 0, and
3. at least one character is available in the TxFIFO as the Flag is going out.

As described in the earlier section *Status Reporting,* software can use four of the bits in the Transmit Command/Status Register (TCSR) to track the progress of the Transmitter through these inter-frame activities. They occur in the time order CRCSent, then EOF/EOM Sent, IdleSent, and finally PreSent. Chapter 6 describes how software can enable any or all of these conditions to cause an interrupt.

Async Transmission

As described in the previous section, the TxIdle field of the Transmit Command/Status Register (TCSR10-8) controls what kind of idle line condition the Transmitter sends between characters (or words) in asynchronous modes. The bits in the Channel Command Register that define the Preamble in sync modes (CCR11-8) can be used in Async mode to "shave" the length of transmitted Stop bits.

Synchronous Reception

Between the end of one message or frame and the start of the next, the Receiver goes through states that are similar to the inter-message or inter-frame activities that are described above for the Transmitter. As described in the earlier section *Status Reporting,* software can use some or all of the following status bits to track these state changes: RxBound (RCSR4), CRCE/FE (RCSR3), IdleRcved (RCSR6), and ExitedHunt (RCSR7). If the DPLL is used, chapter 3 describes the DPLLSync bit in the Channel Command/Status Register (CCSR12) which bears a certain symmetry with the PreSent bit on the Transmit side. Chapter 6 describes how software can enable the RxBound, IdleRcved, and/or Exited Hunt conditions to cause an interrupt.

The IdleRcved logic isn't as flexible as the corresponding TxIdle logic in the Transmitter, in that it only detects an Idle condition consisting of (15 or 16) consecutive ones.

In HDLC/SDLC mode the Receiver automatically copes with single Flags between frames and with shared zeroes between Flags (011111101111110).

Synchronizing Frames/Messages with Software Response

In some applications, software can simply set up DMA buffers for multiple frames or messages, and set the USC's Transmitter and/or Receiver and external DMA controller(s) into operation to send and/or receive all of them. In other applications, software has to interact with and supervise the communications process more closely. (The extreme case is when software has to check status register bits for each character that it transfers to the TxFIFO or from the RxFIFO.)

The USC provides two alternatives for interlocking the start of transmission of a frame or message with software response, and one similar interlock on the receive side.

If the **Wait2Send** bit in the Transmit Interrupt Control Register (TICR2) is 1, then each time the Transmitter finishes sending a frame and before it sends the next, it waits for software to write the Send Frame/Message command to the TCmd field of the Transmit Command / Status Register (TCSR15-12). Depending on the programmed mode the Transmitter may then go on to send the Preamble or the opening Sync or Flag. This kind of interlock allows the software to reprogram global Transmitter parameters that may need to change between frames or messages. It allows an external Transmit DMA controller (or software) to fill the TxFIFO in preparation for the next frame or message, before software issues the Send Frame/ Message command. One use for this interlock would be to change the TxCRCatEnd bit in the Transmit Mode Register (TMR8) between frames, in an application in which the Transmitter should calculate a CRC in some messages or frames but not in others.

If the **Wait4TxTrig** bit in the Channel Control Register (CCR13) is 1, then each time the Transmitter finishes sending a frame and before it sends the next, it waits for software to issue the Trigger Tx DMA (or Trigger Rx and Tx DMA) command before it requests DMA operation. This is a "more stringent" interlock than the preceding one, in that the external Transmit DMA controller won't fill the TxFIFO in preparation for the next frame, until software issues the command. This kind of interlock is useful if DMA-related parameters, or parameters that go through the TxFIFO with the data, need to be changed between frames. The most obvious example is reprogramming the buffer location and length in the Transmit DMA controller.

On the Receive side, if the **Wait4RxTrig** bit in the Channel Control Register (CCR5) is 1, then after an external Receive DMA controller has written a character marked as RxBound to memory (and after it has written the Receive Status Block if software has enabled this feature) the Receiver doesn't assert /RxREQ to the Receive DMA controller again until software writes the Trigger Rx DMA (or Trigger Rx and Tx DMA) command to the RTCmd field of the Channel Command/Status Register (CCAR15-11). Software can use this interlock to reprogram the Receive DMA controller between frames.

5. Direct Memory Access (DMA) Interfacing

Chapter 4 described many of the features of the USC that support handling serial traffic on a DMA basis, that is, without processor intervention on a byte-by-byte basis. This chapter describes how to interface external DMA controllers and how to program the USC to work with them.

DMA and processor data transfers can be mixed in several ways. The USC's two Receivers and two Transmitters can be handled via any mixture of DMA and programmed transfers. Furthermore, software can even mix DMA and programmed transfers for a particular Receiver or Transmitter.

For example, software could use the Wait4RxTrig bit (CCR13) to inhibit DMA transfers at the start of each received frame, so that it can read the first few characters of the frame from the RxFIFO itself. The software can then determine the kind of frame from examining those first characters, optionally program the receive DMA controller accordingly, and then write the "Trigger Rx DMA" command to the RTCmd field of the Channel Command/Address Register (CCAR15-11). The DMA controller can then transfer the rest of the frame into memory without further software intervention.

Flyby vs. Flowthrough DMA Operation

DMA controllers can operate in one of two ways that are called "flyby" or single-cycle mode and "flow-through" or two-cycle mode. Figures 45 and 46 illustrate flowthrough mode, in which the DMA controller performs two bus cycles for each piece of data transferred between the peripheral device and memory. The first cycle reads data from the source, be it the peripheral or the memory. The DMA controller captures this read data and then presents it on the data bus again in the second cycle, which is a write to memory if the data came from the device, or a write to the device if the data came from memory.

The main advantage of flowthrough transfers is that they involve minimal hardware design considerations, because both cycles of each pair are similar to bus cycles performed by the host processor. In the case of the USC there's a secondary advantage in that the /TxACK and/or /RxACK pin(s) can be used for general-purpose input or output.

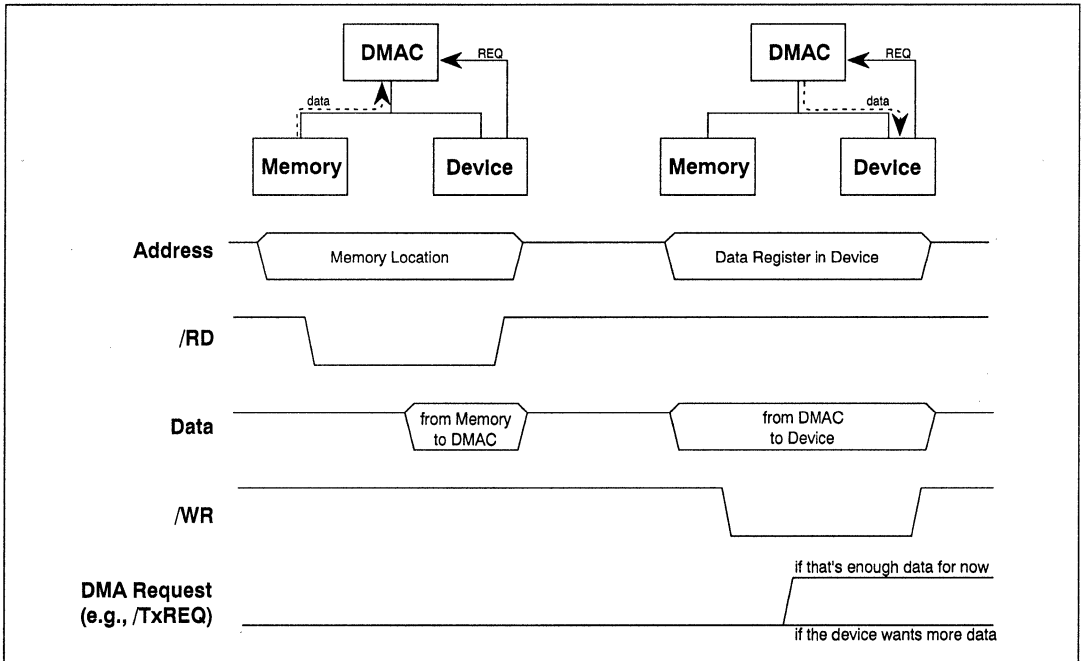


Figure 45. Flowthrough DMA Transfer, Memory to Peripheral Device

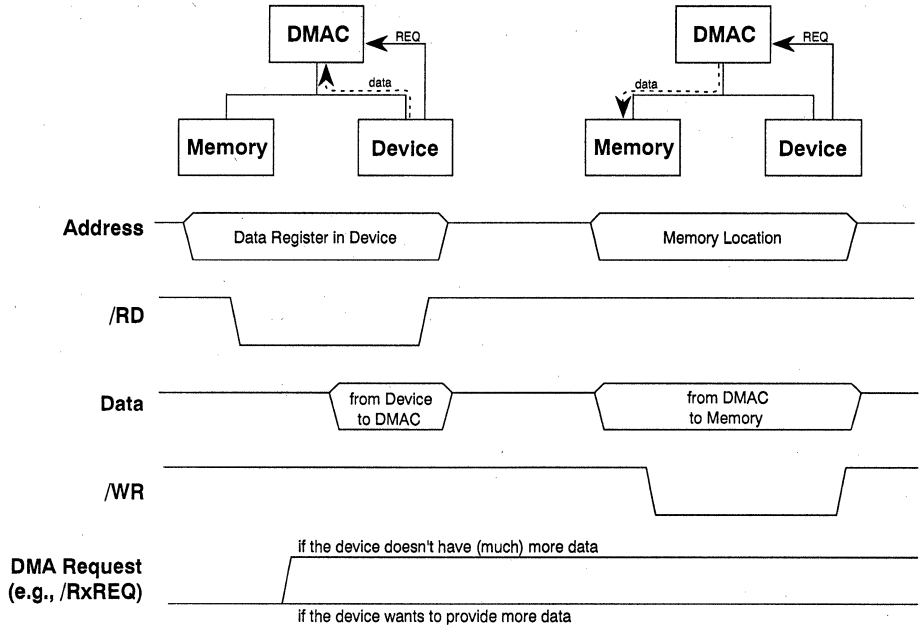


Figure 46. Flowthrough DMA Transfer, Peripheral Device to Memory

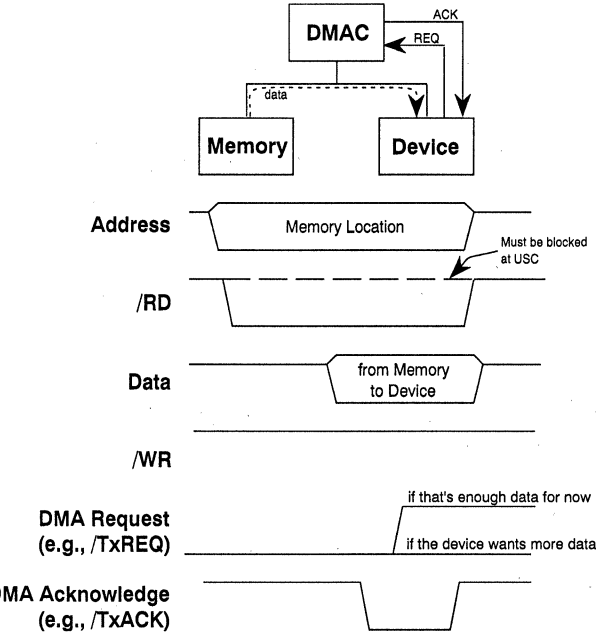


Figure 47. Flyby DMA Transfer, Memory to Peripheral Device

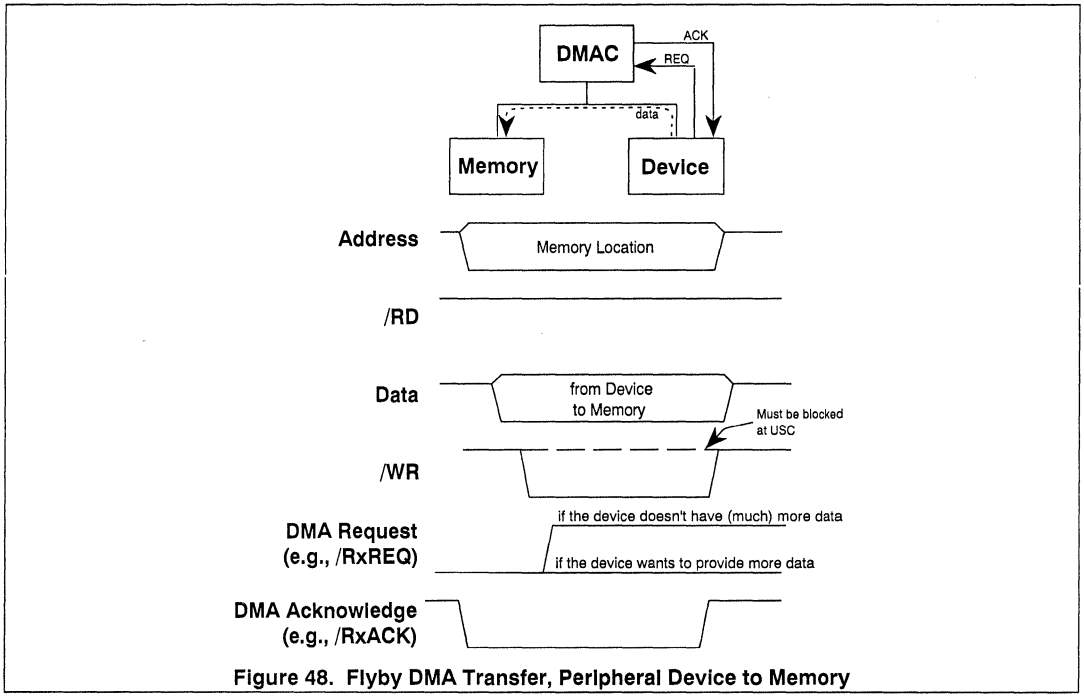


Figure 48. Flyby DMA Transfer, Peripheral Device to Memory

Figures 47 and 48 illustrate flyby (single-cycle) operation. In addition to the Request signal from the device to the DMA controller, there's an Acknowledge signal from the DMAC back to the device. The DMA controller performs just one bus cycle for each piece of data transferred, in which the address lines and standard bus control signals tell the memory what to do to fulfill its part in the transaction. But in addition to this signalling, the DMA controller asserts the Acknowledge line to the device to tell it to perform its part, i.e. to place data on the data lines for a write to memory, or to capture data that's being read from memory.

The main advantage of flyby mode is faster operation, but there's a price to be paid in greater design complexity. Most DMA controllers place this burden mostly on the device side, and try to make DMA cycles appear to the memory as much like processor cycles as possible.

The USC's Transmitters and Receivers can operate in either mode, with one important covenant for flyby operation. Chapter 2 noted that only one among /DS, /RD, /WR, /PITACK, and those /TxACK and /RxACK pins that are used as DMA Acknowledge lines, may be asserted at the same time. While system designers usually think of signals like /DS, /RD, and /WR as being important only when they're qualified by assertion of /CS, the above restriction is true regardless of the state of /CS.

Since the DMA controller typically asserts /DS or /RD or /WR to the memory during a flyby DMA cycle, in order to use flyby transfers **the system designer must provide external logic that blocks /DS, or /RD and /WR, from being asserted at the USC simultaneously with /TxACK or /RxACK.** The simplest way to do this is with a logic gate or two to keep the pin(s) high whenever the DMA controller is in control of the system bus.

DMA Requests by the Receiver and Transmitter

In general, a DMA controller only transfers data when the associated device requests that it do so. To use either flowthrough or flyby DMA operation with a USC Receiver or Transmitter, connect the /RxREQ or /TxREQ pin to the Request input of the DMA controller, and program the RxRMode or TxRMode field (IOCR9-8 or IOCR11-10 respectively) to 01. The 01 value makes the channel output the Receiver's or Transmitter's DMA request on /RxREQ or /TxREQ.

The Transmitter asserts /TxREQ to the transmit DMA controller as follows:

1. when TxRMode (IOCR11-10) is 01 and the Transmitter isn't "holding between frames", from the time that the number of empty character positions in the TxFIFO exceeds the Transmit

DMA Request Level value (TICR15-8 after a "Select TICRHi=/TxREQ Level" command), until

a. the TxFIFO is filled, or

b. the Transmit Character Counter counts down to zero, indicating that the DMA controller is fetching the last character of a message or frame, and either

i. the Transmit Control Block feature is enabled, and/or

ii. the Wait4TxTrig bit (CCR13) is 1.

2. from the time software writes a Trigger Channel Load DMA command to the Channel Command / Address register (CCAR), until a DMA transfer into CCAR clears the ChanLoad bit (CCAR7).

Each of 1.b.i and 1.b.ii establishes a separate "holding between frames" state for the Transmitter. The Transmitter clears the former one automatically, when it finishes sending the frame. Software must clear the latter one, by issuing a "Trigger Tx DMA" command to the RTCmd field of the Channel Command / Address Register (CCAR15-11).

Point 1.b.i reflects the fact that, when Transmit Control Blocks are enabled, the Transmitter stops requesting further DMA transfers after the DMA controller fetches the last character of one frame, until it has sent that character and terminated the frame or message. The Transmitter does this so that the loading of the TCB information for a new frame doesn't affect sending the end of the preceding frame.

When RxRMode (IOCR9-8) is 01 and the Receiver isn't "holding between frames", it asserts /RxREQ to the receive DMA controller in two situations:

A. from the time that the number of received characters in the RxFIFO exceeds the Receive DMA Request Level value (RICR15-8 after a "Select RICRHi=/RxREQ Level" command), until the DMA controller empties the RxFIFO, or

B. in HDLC/SDLC, Ethernet/802.3, Transparent Bisync, or 1553B mode, from the time that the Receiver places a byte marked with RxBound status into the RxFIFO, until the DMA controller has read out the RxBound character. (Such RxBound status signifies the last character of each frame or message in HDLC, Ethernet, and Transparent Bisync mode, and the second or only character of each word in ACV/1553B mode.)

If the software has enabled Receive Status Blocks, the channel keeps /RxREQ asserted while the DMA controller stores the status block in memory. Also, if the number of characters left in the RxFIFO, after the DMAC has read out the RxBound character, still exceeds the Receive DMA Request Level, the channel keeps asserting /RxREQ per condition A.

Note that, if the Wait4RxTrig bit in the Channel Control Register (CCR4) is 1, then after the receive DMA controller writes a character marked with RxBound status into memory (plus the Receive Status Block if this feature is enabled), the Receiver enters a "holding between frames" state. In this state, it doesn't request any more DMA transfers until after software writes a "Trigger Rx DMA" command to the RTCmd field of the Channel Command/Address Register (CCAR15-11). This interlock overrides points A and B above.

The Receive Character Counter feature cannot force the Receiver to assert /RxREQ.

A channel negates /TxREQ within a specified time of the start of the bus cycle that fills the TxFIFO or fetches the last character of the frame or message. A channel negates /RxREQ within a specified time after the start of a bus cycle that empties the RxFIFO or completes the storing of the Receive Status Block.

Programming the DMA Request Levels

As noted in other chapters, the MSByte of the Transmit and Receive Interrupt Control Registers (TICR and RICR) may each represent any of several registers. The content of each register's MSByte depends on which of several selection commands was most recently written to the Transmit or Receive Command Status Register (TCSR or RCSR), respectively. The selections for the Transmitter and Receiver are independent.

To program or read back a DMA Request Level, first write the "Select RICRHi=/RxREQ Level" or "Select TICRHi=/TxREQ Level" command (both being the value 0111) to the TCmd or RCmd field of the Transmit or Receive Command/Status Register (TCSR15-12 or RCSR15-12). This step can be omitted if it's known that no 0101 or 0110 commands have been written to TCSR or RCSR since the last time 0111 was written there. The DMA Request Level value can then be read or written as the MSByte of the TICR or RICR.

The Transmit DMA Request Level should be programmed with 1 less than the number of empty TxFIFO positions, at which the Transmitter should start asserting /TxREQ. The Receive DMA Request Level should be programmed with 1 less than the number of received characters in the RxFIFO, at which the Receiver should start asserting /RxREQ. For example, if the Receiver should request DMA operation when its 32-byte RxFIFO is 3/4 full, software should write hex 70 to RCSR15-8 to select the DMA Request Level as RICR15-8, and then write decimal 23 (hex 17) to RICR15-8.

Note that a Purge Tx FIFO (or Purge Rx and Tx FIFO) command can make a channel immediately assert /TxREQ.

DMA Acknowledge Signals

Each channel of the USC has a /TxACK and an /RxACK pin. In modes other than flyby DMA operation, these pins can be used as outputs or as polled inputs, as described in Chapter 3. For flyby DMA applications, connect these pins to the acknowledge outputs of the DMA controller, and program the RxAMode and TxAMode fields of the Hardware Configuration Register (HCR3-2 and HCR7-6) with 01. The 01 value makes the USC route the signals from these pins to the DMA Acknowledge inputs of the Receiver and Transmitter.

The USC channel provides data on the AD lines within a specified time after /RxACK goes low. For Transmit DMA cycles, data must be valid on the AD lines for specified setup and hold times around each trailing/rising edge on /TxACK. If the /WAIT//RDY pin is configured for the Ready (Data Transfer Acknowledge) function as described in Chapter 2, the USC channel drives /WAIT//RDY low after either /TxACK or /RxACK goes low. Note that, in a system in which the DMA controller requires a Ready or Data Transfer Acknowledge signal, external logic will probably want to condition and combine /WAIT//RDY from the USC and the corresponding signal from the memory, to produce the signal for the DMA controller.

Separating Received Frames in Memory

In some block-oriented communications protocols, software needs to separate received frames or messages so that there is one and only one in each buffer area in memory. Since the only signals between the USC and an external DMA controller are REQ and ACK, there's no way for the USC to tell the DMAC about frame/message boundaries. Therefore there's no way for the DMA transfer process to automatically separate frames/messages in memory by hardware means, and if such separation is to be done it must be by means of software intervention between frames.

As described in an earlier section of this chapter, a USC Receiver asserts the /RxREQ pin when it places a character with end of frame/message (RxBound) status in the RxFIFO, regardless of the FIFO fill level. This promotes separation of received frames/ messages in memory,

The channel then keeps /RxREQ asserted at least until the DMA channel moves the RxBound character from the RxFIFO to memory, at which time the channel sets the RxBound status bit (RCSR4). If the Receive Status interrupt on RxBound is armed and enabled, software can respond to the resultant interrupt by reprogramming the DMA channel for the next memory buffer and restarting it to store the next frame there.

However, this feature is not in itself a complete solution because the USC will keep /RxREQ asserted until the RxFIFO is empty. Unless the DMA response, DMA transfer rate, and interrupt response are all fast relative to the data rate, the first few characters of the next frame may arrive and be stored at the end of the preceding frame's memory buffer, before software can reprogram the DMA controller for the next buffer.

The answer to this problem is to use the Wait4RxTrig bit (CCR5) that's described near the end of Chapter 4. When this bit is set to 1, the USC channel negates /RxREQ as it moves the RxBound character to memory or completes storing the Receive Status Block. Software can then respond to the Receive Status interrupt, reprogram the DMA channel for the next frame, and finally write the "Trigger Rx DMA" command to the RTCmd field (CCAR15-11) to allow the channel to assert /RxREQ again.

6. Interrupts

The interrupt subsystem of the USC derives from Zilog's long experience in providing the most advanced interrupt capabilities in the microprocessor field. These capabilities can be used to their best advantage in a system including a Zilog processor and other Zilog peripherals, but it's easy to interface the USC to interrupt other processors as well. This chapter describes the USC's interrupt capabilities and how to use them in various system applications.

The USC dedicates eight pins to interrupts. Each channel has its own interrupt request output (/INTA and /INTB). The /SITACK and /PITACK inputs signal that the processor is acknowledging an interrupt, in different ways for use with different kinds of host microprocessors.

For applications in which interrupt acknowledge cycles cannot easily be detected at the USC, software can simulate such a cycle.

Each channel has its own Interrupt Enable In (IEIA, IEIB) and Out (IEOA, IEOB) pins. These signals allow systems including several Zilog-compatible peripherals to use an *interrupt acknowledge daisy chain* to select how multiple interrupting devices should be serviced. This can eliminate the need for a separate interrupt controller as in other approaches. On the other hand, because the USC provides separate Interrupt Request outputs and Interrupt Enable inputs for each channel, external interrupt control logic can process interrupt requests in a round-robin or dynamic-priority fashion among the channels in one or more USCs.

Interrupt Acknowledge Daisy Chains

Figure 49 shows an interrupt acknowledge daisy chain. The highest-priority daisy-chainable device that can request an interrupt has its IEI pin tied High. Because of this, it can always request an interrupt, and it "has first claim at" providing an interrupt vector in answer to an interrupt acknowledge cycle. The IEO pin of the highest-priority device is connected to the IEI pin of the next-higher-priority device. This "daisy chaining" of IEO outputs to IEI inputs continues until the lowest-priority daisy-chainable device that can request an interrupt, which has its IEO pin left unconnected.

With the USC as with all Zilog-compatible devices except Z80 family members, the IACK daisy chain serves two separate functions. **During** an interrupt acknowledge cycle, the daisy chain acts to select the highest-priority requesting device as the one to return an interrupt vector. **After that**, until the resulting interrupt service routine is over, the daisy chain serves to block interrupt requests from devices having a lower priority than that of the one currently being serviced, while allowing them from higher-priority devices.

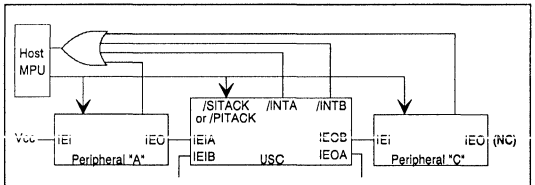


Figure 49. An Interrupt Daisy Chain

This daisy-chain structure allows *nesting* of interrupt service routines. Nesting can greatly improve worst-case interrupt response times for critical real-time applications as well as I/O-intensive computing systems. Whether or not host software uses nested interrupts, the USC's interrupt subsystem provides the most efficient interrupt handling possible.

External Interrupt Control Logic

There are two valid reasons why a system designer might choose not to use an interrupt acknowledge daisy chain (plus the less valid one of not being familiar with them). First, in a system that includes many USC channels all having similar baud rates and serial traffic, the strict priority among channels, that's inherent in a daisy chain, might endanger proper interrupt servicing for the channel(s) at the low-priority end of the chain. In such cases, interrupt service requirements may be more easily guaranteed by using a central interrupt controller that distributes interrupt acknowledgements among the channels on a round-robin (rotating-priority) basis. Such schemes target "fairness" rather than priority in interrupt servicing among the channels.

A second reason not to use a simple/wired interrupt daisy chain would be in a system in which data rates vary over a considerable range among several USC channels, and are determined dynamically rather than being known as the system is being designed. (A channel's interrupt servicing requirements typically vary directly with its serial data rate.) In such a system, external interrupt logic can distribute interrupt acknowledge cycles using a dynamic priority determined by each channel's data rate.

Both rotating-priority and dynamic-priority systems can be arranged as shown in Figure 50. The interrupt control logic maintains the IEI inputs of the channels high most or all of the time, so that the channels can assert their /INT outputs. The logic may simply OR the /INT outputs of the various channels to make the interrupt request to the processor. Alternatively, in a

dynamic-priority system with a processor that supports multiple levels of interrupts, the control logic may assign different channels to different processor levels.

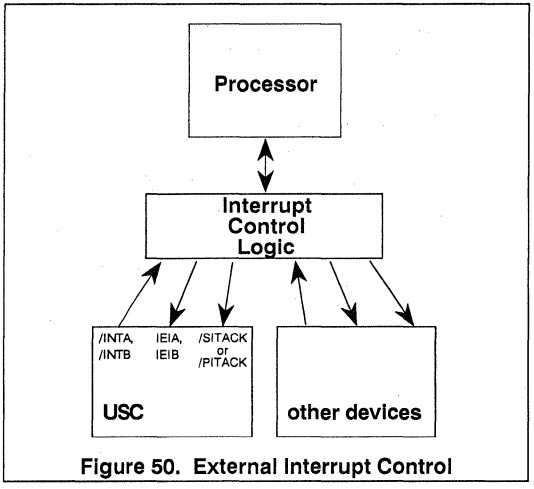


Figure 50. External Interrupt Control

Regardless of how the interrupt control logic derives the processor request, when the processor does an interrupt acknowledge cycle, the logic must select a particular device from among those requesting an interrupt, to "receive" the cycle. The control logic can implement this choice in one of two ways. First, it can negate the IEI inputs of all but one device, and then wait for the specified setup time before presenting the cycle to all of them using the /PITACK or /SITACK signal and possibly other bus control signals. (/PITACK's probably easier to use in this kind of application.) Or, it can simply present the cycle only to the selected channel, typically using a single pulse on /PITACK.

Using /RxREQ and /TxREQ as Interrupt Requests

When an external DMA controller isn't used to handle the Receive or Transmit data for a channel, the corresponding REQ pin isn't used to output a DMA transfer request. In this case software can still program the pin as a DMA request output, and the system designer can use the output signal as another interrupt request instead.

As we will see, software can program "FIFO request levels" for a FIFO's contribution to the /INT pin, that are similar to those discussed in Chapter 5 for the way the FIFO controls its REQ pin. Using an REQ pin for another interrupt request line is advantageous only in a system in which the host processor has multiple interrupt request levels, and the software allows/uses nested interrupts. In such a system, the REQ pin(s) can be connected to a different request level than is the /INT pin, so that data interrupts have a different priority than other kinds of interrupts.

The *DMA Requests by the Receiver and Transmitter* section of Chapter 5 describes how the channel asserts the REQ pin until the software has completely filled the TxFIFO or emptied the RxFIFO, or until the end of the message/frame, whichever comes first. This differs from how the channel asserts its /INT output, and means that an interrupt service routine must take or provide data until the FIFO is full or empty or until the end of the frame or message, in order to avoid immediate re-interruption.

Internal Interrupt Operation

Internally, the USC uses a daisy-chaining scheme much like that described earlier. Each channel includes six interrupt "types", that are arranged in a fixed priority order. Four of the six types include several independent interrupt stimuli or "sources".

Figure 51 presents a model of the typical internal structure of the interrupt subsystem, for a source "s" that is of type "t". Note that the Figure represents a model of the USC's interrupt logic rather than the exact logic; it's included only as an aid to understanding the interrupt subsystem.

Each individual source has an associated register bit that we'll call its Interrupt Arm or IA bit. (Previous Zilog documents called this bit an Interrupt Enable or IE bit, but also used the same term for another bit that applies to the entire type. To distinguish between these two kinds of register bits, this description will call the one that applies to the individual sources "IA".)

IA bits are fully under software control. When an IA bit is 1, the associated source can cause an interrupt.

The sources are typically readable as register bits themselves, and may be derived from various kinds of logic, such as logic that compares the fullness of a FIFO with a threshold level at which to interrupt, or logic that detects transitions of another register bit. Whenever one of the sources for a type and its IA bit are both true, an "Interrupt Pending" register bit (IP) for the type is set to 1. For USC family members, IP bits are set independently of the state of the associated IUS bits, and are cleared to 0 only by software (or by Reset).

A close examination of Figure 51 will show that setting of IP is delayed if an "armed" source comes true during an interrupt acknowledge cycle, but that's not particularly important for understanding the USC's interrupt subsystem...

A second register bit associated with each type is the Interrupt Enable or IE bit. This bit is also under full software control. When an IE bit is 1, an interrupt can be requested when the type's IP bit is 1. Note that an

IP bit can be set while its associated IE bit is 0; if software sets IE before it clears the associated IP bit, an immediate interrupt can result.

There is one more register bit for each type, called the Interrupt Under Service or IUS bit. The interrupt logic sets the IUS bit for a type to 1 during an interrupt acknowledge cycle, if the daisy chain shows that it is the highest-priority type that's currently requesting an interrupt. (This includes types in higher-priority devices and higher-priority types within the channel.) Aside from a hardware or software Reset, an IUS bit can only be reset to 0 by software. This is typically done near the end of an interrupt service routine for that type. During the execution of the interrupt service routine for a given type, the type's IUS bit blocks interrupt requests from lower-priority types.

The And gate near the top of Figure 51 shows the actual conditions for a type to request an interrupt. A type's IP and IE bits must both be 1, its IUS bit must be 0, and its incoming "IEI" signal must be true. IEI true indicates that no higher-priority type (on-chip or external) has its IUS bit set. Finally, a Master Interrupt Enable (MIE) register bit for the channel must be set to 1.

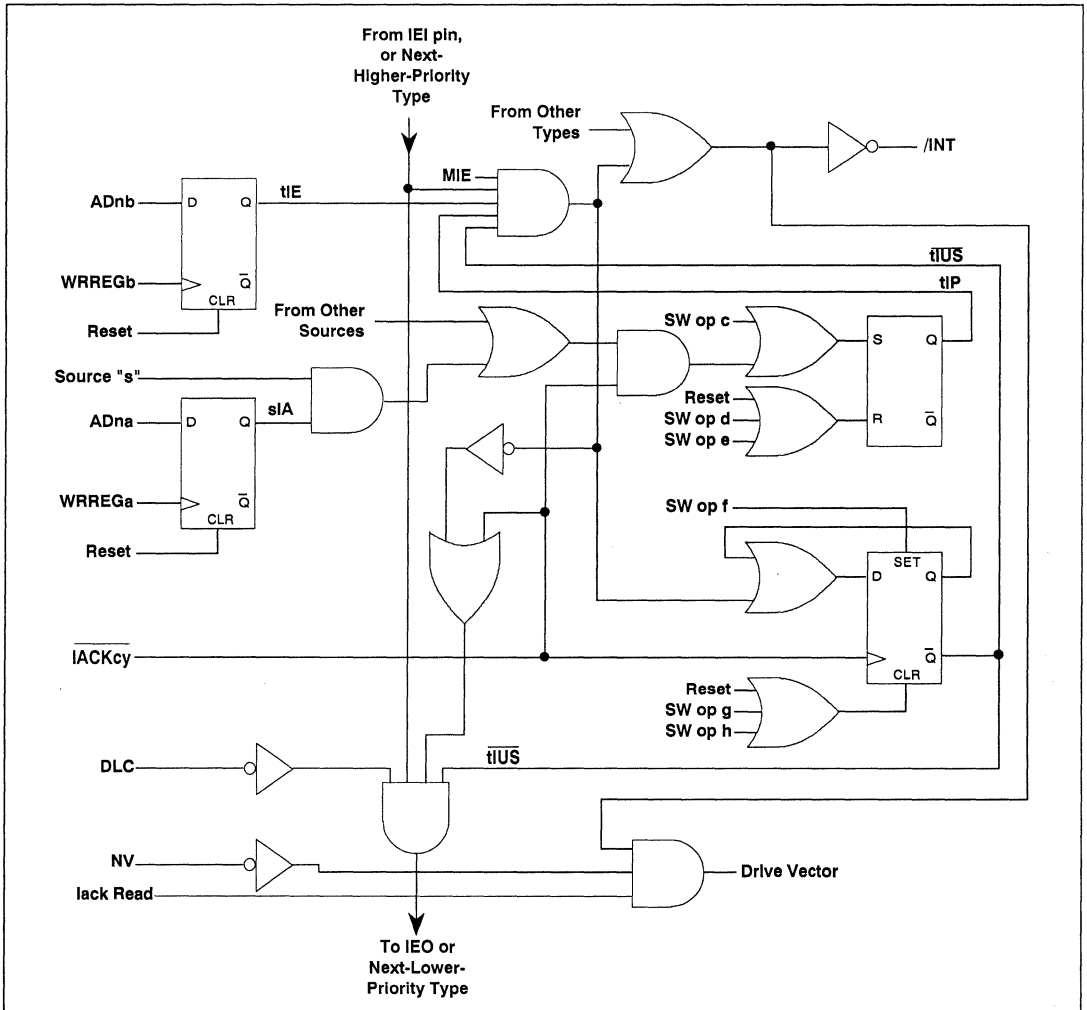


Figure 51. A Model of the Interrupt Logic for Source "s" and Type "t"

Details of the Model

The IA and IE bits appear near the left side of Figure 51, as D-type flip-flops that capture the state of an AD line when software writes a specific register. The IP bit appears as an SR-type latch that's set "by hardware" as described above; software can set and clear the latch. The signal labelled /IACKcy is Low for the duration of an interrupt acknowledge sequence. The IUS bit appears as a D-type flip-flop that can be set via its clock and D inputs at the end of an acknowledge cycle; again, software can set or clear IUS.

The various signals named "SW op x", that set and clear IP and IUS, represent software operations. These may reflect the writing of a "1" bit to a certain register bit position, or may represent the writing of an encoded command to a register. Since software always has to clear IUS and try to clear IP during an interrupt service routine, there are often several ways to do so, as shown by the multiple "SW op" signals for these functions in the Figure. One thing not shown in the Figure is how the typical command "Reset Highest IUS" is implemented -- including this function would have considerably increased the complexity of the logic, which is already complex enough!

The two downward-pointing gates in Figure 51 form the type's "IEO" output. They assert this output only if the type's incoming IEI is High and its IUS bit is 0. There is a register bit "Disable Lower Chain" (DLC) in each channel; if/when DLC is 1 the channel's IEO is forced false/low. The downward-pointing OR gate reflects the functional shift of the daisy-chain during interrupt-acknowledge cycles. Its output is High except during IACK cycles, at which time it allows IEO to be asserted High only if this type is not requesting an interrupt.

Finally, the signal labelled "Drive Vector" controls when the channel places an interrupt vector on the data bus during an interrupt acknowledge cycle. There is a register bit No Vector (NV) in each channel; NV=1 prevents driving a vector. The bus interface logic derives the signal "IACK Read" from R/W and /DS, /RD, or /PITACK. In most cases IACK Read is true during the latter part of the time that /IACKcy is true. The channel provides a vector on AD7-0 while IACK Read is true, if NV is 0 and any of the types in the channel is the highest priority interrupting type.

To keep its complexity reasonable, Figure 51 doesn't include the mechanism by which the content of a returned interrupt vector can reflect the identity of the channel's highest-priority interrupting type.

Software Requirements

While there's considerable variability and flexibility in the USC's interrupt subsystem, there are some common requirements in what an interrupt service routine must do to keep the hardware operating correctly:

1. If the ISR wants to allow nested interrupts, it can reenable processor interrupts near its start. The channel won't request another interrupt of the same type (or any lower-priority type) until software clears the type's IUS bit.
2. The service routine must figure out which type of interrupt it's servicing. This is automatic if the software enables the "Vector Includes Status" (VIS) option.
3. Next the service routine must choose which source(s) within the type it wants to deal with. For each such source that's both active and armed, it must clear the source signal (whatever that takes) or, less typically, clear the associated IA bit.
4. After dealing with as many sources for the type as it can, it must try to clear the IP bit, and clear the IUS bit for the type. This may involve writing one or two specific register bit(s) or writing one or two encoded command(s) to a register. The IP bit {remains set | is set again immediately} if the service routine left any sources for the type both active and armed.
5. Typically the service routine then returns to the interrupted process or program.

The USC channels provide register bits and/or commands to set the IP and/or IUS bits as well as clear them. Software can set IP to force an initial interrupt from a previously-inactive type. The ability to set IUS may be needed as part of simulating an interrupt acknowledge cycle.

Interrupt Option in the BCR

One bit in the Bus Configuration Register (BCR) affects the interrupt subsystem. The following is also presented in Chapter 2, *Bus Interfacing.*

2PulseIACK (Double-Pulse Interrupt Acknowledge; BCR1): software should program this bit to 0 if the /PITACK pin isn't used or if it carries a single pulse when the host processor acknowledges an interrupt, or to 1 if /PITACK carries two pulses when the host processor acknowledges an interrupt. (The latter mode is compatible with certain Intel processors.)

Interrupt Acknowledge Cycles

The USC doesn't require Interrupt Acknowledge cycles. The system designer can simply pull up the /SITACK and /PITACK pins, and software can read the Interrupt Pending (IP) bits in the Daisy Chain Control Register (DCCR), which are described in later sections.

Even if the host processor does Interrupt Acknowledge cycles, the USC doesn't have to provide a vector. If IEI is high and the NV bit in a channel's Interrupt Control Register (ICR) is 1, the channel sets the IUS bit of the highest priority interrupt then pending, but it does not return an interrupt vector.

But, since most microprocessors in use today perform interrupt acknowledge cycles to obtain an 8-bit interrupt vector, the rest of this section will assume vectored interrupts.

Figure 52 shows an interrupt acknowledge cycle that's signalled by /SITACK, on a bus with multiplexed addresses and data. (Actually there are two subcases of this kind of cycle, depending on whether the host processor uses /DS or /RD signalling. Since the timing is the same for either strobe, Figure 52 simply shows a trace labelled "/DS or /RD".)

If the channel samples /SITACK low at the rising edge of /AS, it "freezes" its internal interrupt state; if it is requesting an interrupt it forces its IEO output low regardless of the state of IEI, and starts resolving its internal interrupt priorities. If the IEI and IEO pins are part of an interrupt acknowledge daisy chain with other interrupting devices, this resolution occurs in concert with the interrupt logic in the other devices.

The IEI pin must be valid for a specified setup time before /DS or /RD goes low. The host CPU's strobe must be delayed if needed to guarantee this. If IEI is high and the channel is requesting an interrupt, it responds to /DS or /RD by setting the IUS bit of its highest requesting type of interrupt, driving a vector onto the AD7-0 pins, and driving /WAIT//RDY appropriately to signal when the vector is valid. If IEI is low at the leading/falling edge of /DS or /RD, and/or if the channel is not requesting an interrupt, it doesn't respond to the cycle.

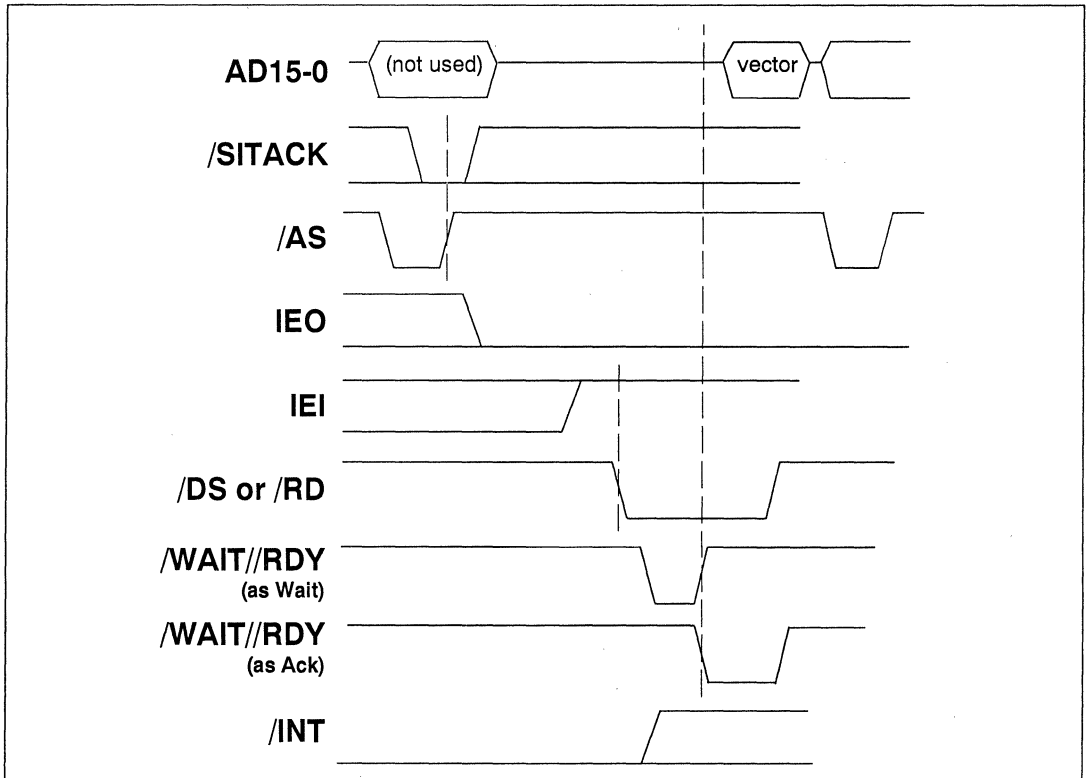


Figure 52. An Interrupt Acknowledge Cycle signalled by /SITACK, on a Multiplexed Bus

Figure 53 shows an interrupt acknowledge cycle that's signalled by /SITACK, on a bus with separate address and data lines. (As before there are two subcases of this kind of cycle, depending on whether the host processor uses /DS or /RD signalling. Since the timing is identical for either strobe, Figure 53 simply shows a trace labelled "/DS or /RD".)

Here the channel freezes its internal interrupt state in response to a falling edge on /SITACK; again, if it is requesting an interrupt it forces its IEO output low regardless of the state of IEI, and starts resolving its internal interrupt priorities.

In this mode /SITACK must stay low until after /DS or /RD goes low, and IEI must be valid for a specified setup time before /DS or /RD goes low. (The falling edge of /DS or /RD may have to be delayed to guarantee this.) If IEI is high and the channel is requesting an interrupt, it responds to /DS or /RD by setting the IUS bit of its highest priority requesting type of interrupt, driving a vector onto the AD7-0 pins, and driving /WAIT//RDY appropriately to signal when the vector is valid. If IEI is low at the leading/falling edge on /DS or /RD, and/or if the channel is not requesting an interrupt, it doesn't respond to the cycle.

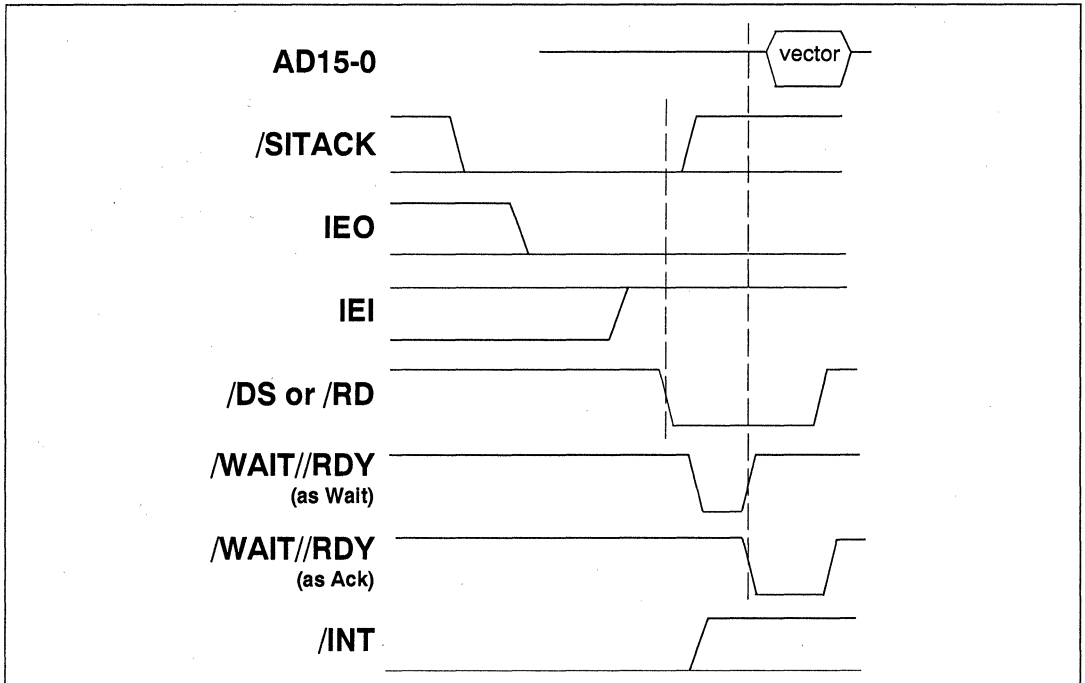


Figure 53. An Interrupt Acknowledge Cycle signalled by /SITACK, on a Non-Multiplexed Bus

Figure 54 shows the kind of interrupt acknowledge cycle that the USC expects when /PITACK goes low and the 2PulseIACK bit (BCR1) is 0. Here a single pulse on /PITACK substitutes for the pulse on /DS or /RD in the previous cases; the latter two signals must remain high throughout the cycle. For this case, operation on a non-multiplexed bus is identical with that on a multiplexed bus once the /AS strobe is over. The only distinction is that a multiplexed bus must meet minimum times between the pulse on /PITACK and the preceding and following pulses on /AS. These minima are similar to those required for register read and write cycles.

In this mode, an interrupt acknowledge daisy chain on IEI/IEO cannot be used to select whether an USC channel or another device should respond to each interrupt acknowledge cycle. Instead, external logic like that shown in Figure 50 must decide which requesting device/channel is to respond to an interrupt acknowledge cycle, if such a cycle occurs when more than one is requesting an interrupt. The external logic would typically consider the state of the individual devices'/channels' interrupt request lines in making this decision. (The lines cannot be OR-tied in this case.)

In this "single-pulse" mode, the IEI pin must set up and hold around the leading/falling edge on /PITACK. If IEI is high and the channel is requesting an interrupt at that point, it responds to /PITACK by driving a vector onto the AD7-0 pins and driving /WAIT//RDY appropriately to signal when the vector is valid. If IEI is low at the leading/falling edge of /PITACK, and/or if the channel is not requesting an interrupt at that point, it doesn't respond to the cycle.

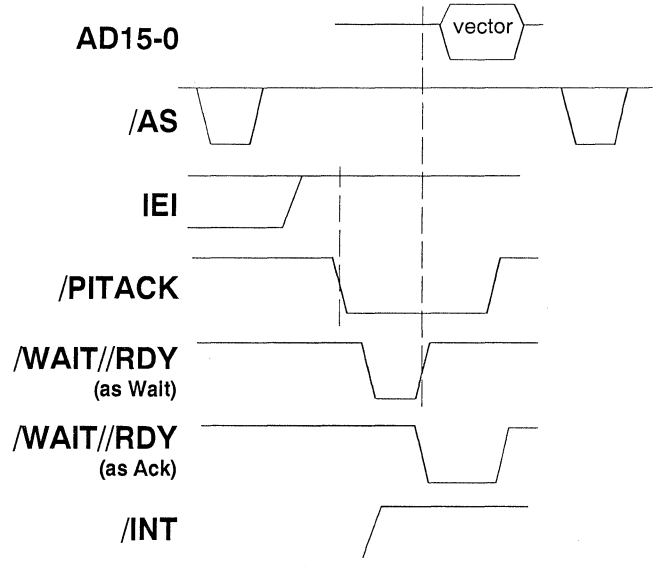


Figure 54. A /PITACK Interrupt Acknowledge Cycle with 2PulseIACK=0

Figure 55 shows the kind of interrupt acknowledge cycle that the USC expects when /PITACK goes low and the 2PulseIACK bit (BCR1) is 1. Here, two consecutive low pulses on /PITACK constitute the complete interrupt acknowledge cycle, and /DS and /RD should both stay high throughout the cycle. This mode is compatible with several microprocessors made by Intel Corp. and other companies. As in the preceding case, operation is similar whether the bus is multiplexed or non-multiplexed. The multiplexed bus must meet minimum times between the pulses on /AS and the pulses on /PITACK. These minima are similar to those between /AS and /DS or /RD in register read cycles.

In "double pulse mode" the channel keeps an internal state bit that distinguishes the two /PITACK pulses in each pair. The channel freezes its internal interrupt state in response to the first falling edge on /PITACK. If it is requesting an interrupt it forces its IEO output low regardless of the state of IEI, and starts resolving its internal interrupt priorities, but the channel does not otherwise respond to the first cycle.

In this mode the IEI pin must be valid for a specified setup time before /PITACK goes low for the second pulse. If IEI is high at this point and the channel is requesting an interrupt, it responds to the second /PITACK pulse by setting the IUS bit of its highest-priority requesting type of interrupt, driving a vector onto the AD7-0 pins, and driving /WAIT//RDY appropriately to signal when the vector is valid. If IEI is low at the leading edge of /PITACK, and/or if the channel is not requesting an interrupt, it doesn't respond to the cycle.

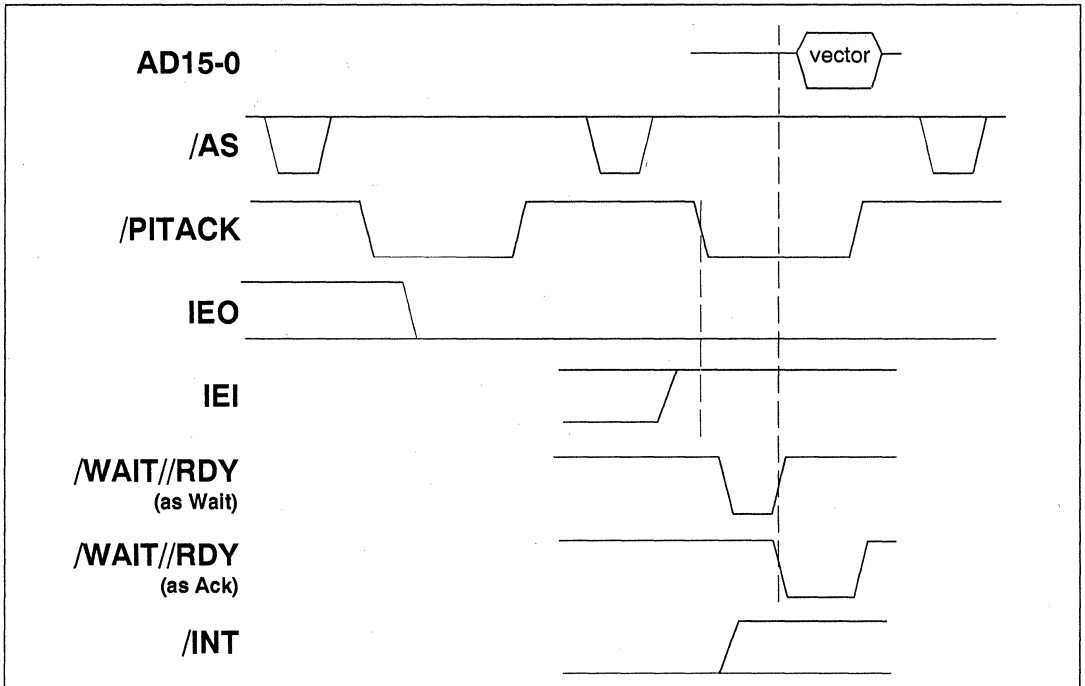


Figure 55. A /PITACK Interrupt Acknowledge Cycle with 2PulseIACK=1

Interrupt Acknowledge vs. Read Cycles

Interrupt Acknowledge cycles are similar to the cycles that occur when the host processor reads a USC register, which are discussed in Chapter 2. However, the user should note the following ways in which interrupt acknowledge cycles differ from read cycles:

* On a multiplexed bus, /SITACK acts like an address line. When a USC samples /SITACK low at a rising edge on /AS, it ignores the address on the AD lines.
* On a non-multiplexed bus, each leading edge of /RD or /DS captures the state of /SITACK.
* With /DS signalling, the state of R//W doesn't matter for a cycle in which the USC samples /SITACK low. (In other cycles R//W differentiates Read cycles from Writes.)
* When the /WAIT//RDY pin carries the Wait function, a USC channel asserts the pin during interrupt acknowledge cycles, but never does so during register Read or Write cycles.
* When /WAIT//RDY carries the Acknowledge function, a channel asserts it later in Interrupt Acknowledge cycles than in Reads. However, the relationship between the falling edge of /WAIT //RDY and the validity of data on the AD lines is similar in both kinds of cycles.

Interrupt Types

Each USC channel includes six types of interrupts, arranged on the internal interrupt daisy chain in the following priority order:

1. Receive Status (highest priority)
2. Receive Data
3. Transmit Status
4. Transmit Data
5. I/O Pin
6. Miscellaneous (lowest priority)

Each of these types has one each IE, IP, and IUS bit, as described in an earlier section of this chapter.

Receive Status Interrupt Sources and IA Bits

Any of six interrupt sources can set a channel's Receive Status IP bit. Software can read the status of each source in the LSByte of the Receive Command / Status Register (RCSR), which is shown in Figure 56. The following descriptions of the RCSR status bits are similar to those in the *Detailed Status in the RCSR* section of Chapter 4:

ExitedHunt The RS IP bit can be set when this bit (RCSR7) goes from 0 to 1 because the receiver has detected a Sync or Flag sequence in a synchronous mode.

IdleRcved The RS IP bit can be set when this bit (RCSR6) goes from 0 to 1 because the receiver has seen 15 or 16 consecutive one bits. In asynchronous modes with 16, 32, or 64X clocking, the receiver sets RCSR6 after one bit time or less, so this source's IA bit shouldn't be set in such modes.

Break/Abort The RS IP bit can be set when this bit (RCSR5) goes from 0 to 1 because the Receiver has detected a Break condition in an asynchronous mode or an Abort condition in an HDLC/SDLC mode.

RxBound If the IA bit for this source is 1, the interrupt logic sets the RS IP bit when software or the Receive DMA channel reads a character from the RxFIFO that's marked with RxBound status. Such marking reflects an address character in Nine-Bit mode, a word boundary in 1553B mode, negation of /DCD during the character in external sync mode, the last character of a frame in HDLC/SDLC and 802.3 modes, or one of five block terminating characters in Transparent Bisync mode.

PE If the IA bit for this source is 1, the interrupt logic sets the RS IP bit when software or the Receive DMA channel reads a character from the RxFIFO that failed parity checking.

RxOver If the IA bit for this source is 1, the interrupt logic sets the RS IP bit when software or the Receive DMA channel reads a character from the RxFIFO that's marked with Overrun status. A character so marked is the first one that arrived while the FIFO was full; the character before this one is lost, and an indeterminate number after it may have been lost as well.

RCmd (WO)		RxResidue	ShortF/ CVType	Exited Hunt	Idle Rcved	Break /Abort	Rx Bound	CRCE /FE	PE	Rx Over	Rx Avail
2ndBE	1stBE										
15 14 13 12		11 10 9	8	7	6	5	4	3	2	1	0

Figure 56. The Receive Command/Status Register (RCSR)

"RxFIFO fill level" if last RCSR15-12 command 4-7 was 5 "Rx Int Req level" if last RCSR15-12 command 4-7 was 6 "Rx DMA Req level" if last RCSR15-12 command 4-7 was 7								Exited Hunt IA	Idle Rcved IA	Break/ Abort IA	Rx Bound IA	Word Status	Parity Error IA	RxOver IA	TC0R Sel
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 57. The Receive Interrupt Control Register (RICR)

As described in Chapter 4, once an Interrupt-Armed RCSR bit has been set, it must be "unlatched" by writing a 1 to that bit position in RCSR. For Exited Hunt, Abort (in HDLC mode), RxBound, PE, and RxOver, this action also clears the RCSR bit. The IdleRcved and Break/Abort (in async modes) bits in RCSR don't become 0 until software has unlatched the bit and the line condition has ended.

Each of these six sources has a separate **Interrupt Arm (IA)** bit in the LSByte of the Receive Interrupt Control Register (RICR). Figure 57 shows the RICR. If an IA bit is 1, the interrupt logic sets the Receive Status IP bit as described above. If an IA bit is 0, the corresponding bit in RCSR has no effect on the IP bit and thus will not cause interrupts. The setting of the IA bits for the ExitedHunt, IdleRcved, and Break/Abort conditions has no effect on the bits in RCSR, while the IA bits for the RxBound, Parity Error, and Overrun conditions affect how the corresponding RCSR bits operate, as described in Chapter 4.

Receive Data Interrupts

This interrupt type has only one source, so there's no IA bit for it. The interrupt logic sets the RD IP bit when a character is received and the number of previously-received characters in the RxFIFO is equal to the number programmed as the "Receive Data Interrupt Request Level". That is, the IP bit is set for the character that makes the number of characters in the RxFIFO exceed the programmed value.

The RD IP bit is also set if the number of characters is less than the programmed threshold level, and the receiver places a character marked with RxBound status in the RxFIFO.

If received data is handled by either software polling or an external Receive DMA channel, disable the Receive Data interrupt by leaving its IE bit 0. (A later section discusses IE bits.)

To program the Receive Data Interrupt Request Level, first write the "Select RICRHi=/INT Level" command to the RCmd field of the Receive Command/Status Register (RCSR15-12). Then write the number of received characters at which the channel should start requesting a Receive Data interrupt, minus one, to the MSByte of the Receive Interrupt Control Register (RICR). For example, if the channel should request a Receive Data interrupt when its 32-byte RxFIFO becomes 3/4 full, write hex 60 to RCSR15-8, then write decimal 23 (hex 17) to RICR15-8.

Figure 58 shows a sample service routine for Receive Data interrupts. While it's not particularly fancy or efficient, it does illustrate several important points:

1. It reads the FIFO fill level to determine how many characters to read. The fact, that reception of an RxBound character (i.e., the last character of a frame, message, or ACV/1553B word) can set the Receive Data IP bit, means that a Receive Data interrupt service routine can't blindly read the number of characters implied by the Interrupt Request Level.
2. It explicitly clears the Receive Data IP and IUS bits by writing to the Daisy Chain Control Register (DCCR) as described in a later section. Neither bit is affected by reading data from the RxFIFO.
3. It re-reads the FIFO fill level after clearing the IP bit, and processes any characters that have been received while it was processing earlier characters. This procedure guards against losing an interrupt associated with a late-arriving End of Frame (RxBound) character.
4. It reads the status from RCSR "before" reading each character, and reads RCSR an extra time after reading out an End of Frame (RxBound) character, to clear the latching of the status that occurs when an RxBound character is read out.

   (This is not the only way to handle RxBound checking. Another way is to enable a Receive Status interrupt when the Receive Data interrupt service routine reads a RxBound character out of the RxFIFO, and not check RxBound status in this routine at all. Software that uses this method must ensure that an Receive Status interrupt can interrupt the Receive Data ISR in a "nested" fashion.)

Transmit Status Interrupt Sources and IA Bits

The interrupt logic can set the Transmit Status IP bit in response to any of six interrupt sources. Software can read the status of each source in the LSByte of the Transmit Command/Status Register (TCSR), which is shown in Figure 59. The following descriptions of the TCSR bits are similar to those in the *Detailed Status in the TCSR* section of Chapter 4:

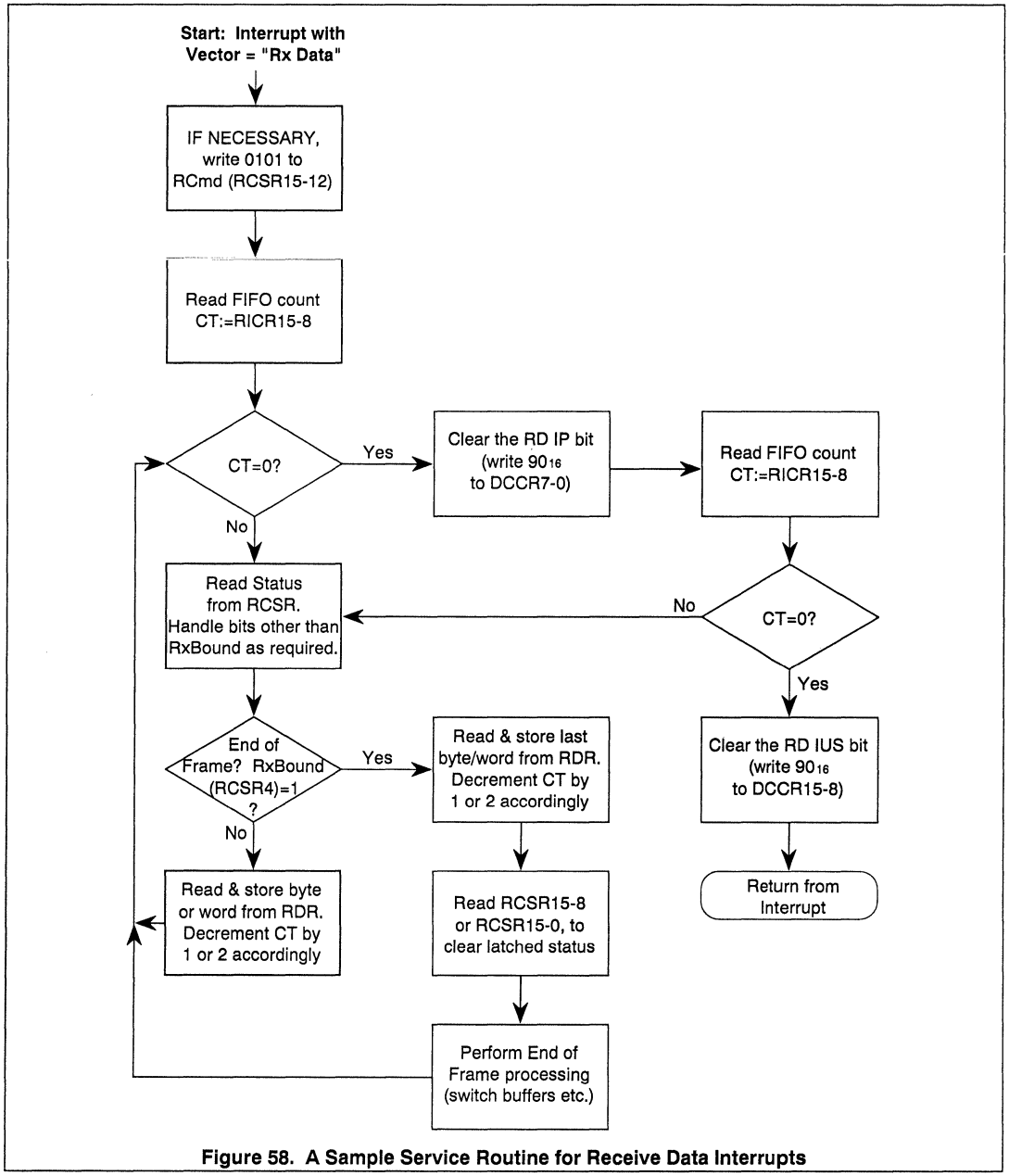


Figure 58. A Sample Service Routine for Receive Data Interrupts

TCmd				Rsrvd	TxIdle			Pre Sent	Idle Sent	Abort Sent	EOF/ EOM Sent	CRC Sent	All Sent	Tx Under	Tx Empty
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 59. The Transmit Command/Status Register (TCSR)

"TxFIFO Status" If last TCSR15-12 command 4-7 was 5 "Tx /INT Level" If last TCSR15-12 command 4-7 was 6 "/TxREQ Level" If last TCSR15-12 command 4-7 was 7								Pre Sent IA	Idle Sent IA	Abort Sent IA	EOF/ EOM Sent IA	CRC Sent IA	Wait2 Send	Tx Under IA	TC1R Sel
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 60. The Transmit Interrupt Control Register (TICR)

PreSent The interrupt logic can set the TS IP bit when this bit (TCSR7) goes from a 0 to a 1, because the transmitter has finished sending the "Preamble" selected in the Channel Control Register (CCR11-8) in a synchronous mode.

IdleSent The interrupt logic can set the TS IP bit when this bit (TCSR6) goes from a 0 to a 1, because the transmitter has sent the idle line state selected by the TxIdle field (TCSR10-8). If TxIdle and TxMode specify the condition as Flags or Syncs, this bit can be set for each one sent. Otherwise, for bit-oriented Idle conditions, it's set only after the first bit is sent.

AbortSent The interrupt logic can set the TS IP bit in HDLC/SDLC mode, when this bit (TCSR5) goes from 0 to 1 because the transmitter has sent an Abort character.

EOF/EOM Sent The interrupt logic can set the TS IP bit in a synchronous mode, when this bit (TCSR4) goes from 0 to 1 because the transmitter has sent the closing Flag or Sync character at the end of a message or frame.

CRCSent The interrupt logic can set the TS IP bit in a sync mode, when this bit (TCSR3) goes from 0 to 1 because the transmitter has sent the CRC sequence just before the end of a message or frame.

TxUnder The interrupt logic can set the TS IP bit when this bit (TCSR1) goes from 0 to 1, because the transmitter needed a character from the TxFIFO but it was empty.

All six of these sources operate differently from the general model described earlier, in that the interrupt logic sets the IP bit only when a TCSR bit goes from 0 to 1 and its associated IA bit is 1. Once one of these TCSR bits is 1, it must be cleared to 0 by writing a 1 to that bit position in TCSR.

Each of these six sources has a separate **Interrupt Arm (IA)** bit in the LSByte of the Transmit Interrupt Control Register (TICR). Figure 60 shows the TICR. If an IA bit is 1, the interrupt logic sets the Transmit Status IP bit when the corresponding bit in the Transmit Command/Status Register (TCSR) goes from 0 to 1. If an IA bit is 0, the corresponding TCSR bit has no effect on the IP bit and thus will not cause interrupts. The setting of the IA bits in TICR has no direct effect on the TCSR bits.

Transmit Data Interrupts

This interrupt type has only one source, so there's no need for an IA bit for it. The interrupt logic sets the Transmit Data IP bit whenever the number of **empty character positions** in the TxFIFO is **greater than** the number programmed as the "Transmit Data Interrupt Request Level". If transmitted data is to be handled by an external Transmit DMA channel, disable this interrupt by leaving its IE bit 0. (A later section discusses IE bits.)

To program the Transmit Data Interrupt Request Level, first write the "Select TICRHi=/INT Level" command (value 0110) to the TCmd field of the Transmit Command / Status Register (TCSR15-12). Then write the number of empty character positions at which the channel should start requesting a Transmit Data interrupt, minus one, to the MSByte of the Transmit Interrupt Control Register (TICR). For example, if the channel should request a Transmit Data interrupt when its 32-byte TxFIFO has only four characters left in it, write hex 60 to TCSR15-8, then write decimal 27 (hex 1B) to RICR15-8.

Note that a Purge Tx FIFO (or Purge Rx and Tx FIFO) command will typically make a channel immediately set its Transmit Data IP bit. This will, in turn, make the channel start requesting an interrupt on its /INT pin if:

* it hadn't been doing so,
* the channel's IEI pin is high,
* its TD IE and MIE bits are 1, and
* its TD IUS and all higher-priority IUS bits are 0.

RxCDn IA	RxCUp IA	TxCDn IA	TxCUp IA	RxRDn IA	RxRUp IA	TxRDn IA	TxRUp IA	DCDDn IA	DCDUp IA	CTSDn IA	CTSUp IA	RCC Under IA	DPLL DSync IA	BRG1 IA	BRG0 IA
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 61. The Status Interrupt Control Register (SICR)

RxCL/U	/RxC	TxCL/U	/TxC	RxRL/U	/RxR	TxRL/U	/TxR	DCDL/U	/DCD	CTSL/U	/CTS	RCC Under L/U	DPLL DSync L/U	BRG1 L/U	BRG0 L/U
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 62. The Miscellaneous Interrupt Status Register (MISR)

As with all USC interrupts, a Transmit Data interrupt service routine must explicitly clear the Transmit Data IP and IUS bits by writing to the Daisy Chain Control Register (DCCR) as described later; the bits aren't cleared by simply writing data into the TxFIFO.

I/O Pin Interrupt Sources and IA Bits

The interrupt logic can set the I/O Pin IP bit in response to rising and/or falling edges on any of six pins for each channel, namely /RxC, /TxC, /RxREQ, /TxREQ, /DCD, and /CTS. The following description is similar to that in the *Edge Detection and Interrupts* section of Chapter 3.

Software can program the channel to detect rising and/or falling edges on the /CTS, /DCD, /TxC, /RxC, /TxREQ, and /RxREQ pins, and to interrupt when such events occur. Figure 61 shows that the Status Interrupt Control Register (SICR) includes separate Interrupt Arm (IA) bits for rising and falling edges on each of these pins. A 1 in one of these bits makes the channel detect that kind of edge, while a 0 makes it ignore such edges. This edge detection and interrupt mechanism operates without regard for whether the various pins are programmed as inputs or outputs in the I/O Control Register (IOCR).

When a channel detects an edge that's enabled in the SICR, it records the event in an internal latch that's not directly accessible in the USC's register map. Instead, as shown in Figure 62, the Miscellaneous Interrupt Status Register (MISR) includes two bits for each of these six pins, one called a "Latched/Unlatch" or L/U bit, and the other being a "data bit" that has the same name as the pin itself.

A hardware or software Reset sequence clears all the L/U bits to zero. While the L/U bit for a pin is 0, the associated data bit reports and tracks the state of the pin in a "transparent" fashion, with a 1 indicating a low and a 0 indicating a high.

Whenever a pin's L/U bit is 0 and its internal edge-detecting latch is set, the channel sets the L/U bit to 1, clears the detection latch, and sets the IOP IP bit. IOP IP can be read and cleared (and if necessary set) in the Daisy Chain Control Register (DCCR1).

While an L/U bit is 1, the state of the associated data bit is frozen (latched). These two bits remain in this state, regardless of further transitions on the pin, until software writes a 1 to the L/U bit.. This clears the L/U bit to 0 and "opens" the data bit to once again report and track the state of the pin, at least for an "instant". If one or more enabled transitions occurred while the L/U bit was set, then L/U is set again right after software writes the 1 to it.

Writing a 0 to an L/U bit has no effect; it doesn't matter what value software writes in the "data" bits.

Miscellaneous Interrupt Sources and IA Bits

The interrupt logic can set the Miscellaneous IP bit in response to any of four interrupt sources. Software can read the status of these sources in the LSByte of the Miscellaneous Interrupt Status Register (MISR), which is shown in Figure 62. The following descriptions repeat some information that was presented in Chapters 3 and 4:

RCCUnder — If the RCCUnder IA bit is 1, a channel sets this bit (MISR3) and the Misc IP bit if the receiver has decremented the Receive Character Counter (RCC) to zero and then it receives another character (in the same frame / message).

DPLLDSync — If the DPLLDSync IA bit is 1, a channel sets this bit (MISR2) and the Misc IP bit if software set up the Digital Phase Locked Loop circuit for Biphase encoding and the DPLL detects two consecutive missing clocks, indicating a loss of synchronization.

BRG1 — If the BRG1 IA bit is 1, a channel sets this bit (MISR1) and the Misc IP bit when Baud Rate Generator 1 counts down to zero.

BRG0 — If the BRG0 IA bit is 1, a channel sets this bit (MISR0) and the Misc IP bit when Baud Rate Generator 0 counts down to zero.

Once any of these bits is 1, software must write a 1 to that bit position to "unlatch" it. Writing a 1 to any of MISR3-0 clears the "read-side" bit unless the setting event recurred while the bit was latched, in which case the bit is set again immediately.

Each of these four sources has a separate **Interrupt Arm (IA)** bit in the LSByte of the Status Interrupt Control Register (SICR). Figure 61 shows the SICR. If an IA bit is 1, the interrupt logic sets the corresponding bit in MISR, and the Miscellaneous IP bit, when the indicated condition occurs. If an IA bit is 0, the logic won't set the corresponding MISR bit and thus the associated condition can't cause interrupts. Clearing an IA bit does not clear the corresponding bit in MISR.

Interrupt Pending and Under Service Bits

Software can read, set, and clear the **Interrupt Pending (IP)** and **Interrupt Under Service (IUS)** bits, for all six interrupt types in a USC channel, via the Daisy-Chain Control Register (DCCR). Figure 63 shows the DCCR. The MSByte deals only with the IUS bits, while the LSByte deals with the IP bits but can be used to clear the IP and IUS bits in one step.

Software can read the six IUS bits from DCCR13-8 and the six IP bits from DCCR5-0. The two MSBits of each byte always read as 00. When software writes the DCCR, the two MSBits of each byte can represent a command *that is applied to the type(s) selected by ones written in the six LSBits of that byte.* DCCR15-14 are an IUS Op field that the channel interprets as follows:

IUS Op	Operation
0x	No operation
10	Clear the IUS bit(s) of the type(s) selected in DCCR13-8
11	Set the IUS bit(s) of the type(s) selected in DCCR13-8

DCCR7-6 are an IP Op field that the channel interprets as follows:

IP Op	Operation
00	No operation
01	Clear both the IP and IUS bit(s) of the type(s) selected in DCCR5-0
10	Clear the IP bit(s) of the type(s) selected in DCCR5-0
11	Set the IP bit(s) of the type(s) selected in DCCR5-0

If software writes both bytes of the DCCR simultaneously on a 16-bit bus, the IUS command is "set", the IP command is "clear both", and a particular type is selected by ones in both the MSByte and LSByte, the channel clears the IUS bit for that type. On the other hand, if the IUS command says "set" for a type and the LSbyte says "clear both" but that type's bit in DDCR5-0 is 0, the channel sets that type's IUS bit.

In addition, one of the encoded commands that can be written to the Channel Command/Address Register (CCAR) allows for a general exit from an interrupt service routine, regardless of which type initiated the routine. If software writes the Reset Highest IUS command (00010) to a channel's RTCmd field (CCAR15-11), it clears the highest-priority IUS bit that's set in the channel. Unfortunately, the command doesn't also clear the corresponding IP bit, so that an interrupt service routine has to do this explicitly for the particular type that it's servicing.

Interrupt Enable Bits

Software can read, set, and clear the **Interrupt Enable** (IE) bits for all six interrupt types in a USC channel, in the LSByte of its Interrupt Control Register (ICR). Figure 64 shows the ICR. Software can read all six IE bits from ICR5-0; ICR7-6 always read as 00. When software writes the LSByte of the ICR, the IE Op field (ICR7-6) comprises a command that the channel applies to any and all IE bits selected by ones written to ICR5-0. The channel interprets IE Op as follows:

IE Op	Operation
0x	No operation
10	Clear the IE bit(s) of the type(s) selected in ICR5-0
11	Set the IE bit(s) of the type(s) selected in ICR5-0

Channel Interrupt Options

Figure 64 shows that the MSByte of the Interrupt Control Register (ICR) contains control bits that apply to all interrupts from a USC channel. These bits are fully under software control and can be read or written at any time.

The **Master Interrupt Enable** (MIE; ICR15) must be set to 1 to allow the channel to request an interrupt on its /INT pin.

Whenever the **Disable Lower Chain** bit (DLC; ICR14) is 1, the channel forces its IEO output low, so that devices further down the daisy chain can't request interrupts nor respond to interrupt acknowledge cycles.

IUSCmd (WO)		RS IUS	RD IUS	TS IUS	TD IUS	IOP IUS	Misc IUS	IPCmd (WO)		RS IP	RD IP	TS IP	TD IP	IOP IP	Misc IP
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 63. The Daisy Chain Control Register (DCCR)

MIE	DLC	NV	VIS				Rsrvd	IECmd (WO)		RS IE	RD IE	TS IE	TD IE	IOP IE	MIsc IE
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 64. The Interrupt Control Register (ICR)

If the **No Vector** bit (NV; ICR13) is 1, the channel neither provides a vector nor drives the /WAIT//RDY pin during an interrupt acknowledge cycle in which the highest-priority requesting type is in the channel. However, in such a case the channel still sets the IUS bit of the highest-priority requesting type.

The **Vector Includes Status** field (VIS; ICR12-9) controls whether the vector, that the channel returns during an interrupt acknowledge cycle in which the highest-priority requesting type is in the channel, identifies the type or not. Such vector modification can be enabled for all types in the channel, or only for those above a selected priority level:

VIS	Which types appear in vectors
0xxx	No types
100x	All types
1010	IOP and above (not Misc)
1011	Transmit Data and above
1100	Transmit Status and above
1101	Receive Data & Status
1110	Receive Status only
1111	No types

If the contents of VIS allow the highest-priority type, that's requesting at the time of an Interrupt Acknowledge cycle, to modify the interrupt vector, then bits 4-1 of the returned vector identify that type as described in the next section. If not, the channel returns the 8-bit vector exactly as the host software programmed it.

Interrupt Vectors

Software can read and write a channel's interrupt vector information in the Interrupt Vector Register (IVR). This register is also the basis of the vector that the channel returns during an interrupt acknowledge cycle in which the highest priority requesting type is in the channel.

Figure 65 shows the IVR. The basic vector can be written and read in its LSByte; software can read a modified version of the vector in its MSByte. (Writing the MSByte has no effect.) Bits 15-12 and 8 are the image of those in the corresponding bits of the LSByte, while the **TypeCode** field (IVR11-9) gives the identity of the highest priority interrupt type that has its IP bit set (the state of its IUS bit doesn't matter).

TypeCode	Meaning
000	No interrupt pending
001	Miscellaneous
010	I/O pin
011	Transmit Data
100	Transmit Status
101	Receive Data
110	Receive Status
111	(will not be read)

The state of the VIS field (ICR12-9) has no effect on reading the IVR. VIS simply controls how the channel decides whether to return IVR15-8 or IVR7-0 as the interrupt vector when it responds to an interrupt acknowledge cycle.

Interrupt Vector 7-4 (RO)				TypeCode (RO)			IV0 (RO)	Interrupt Vector (RW)							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 65. The Interrupt Vector Register (IVR)

7. Software Summary

Just about everything important about the USC has been said in previous chapters. This one simply pulls together some loose ends of interest to software types, as well as providing a unified reference to all the register fields.

About Resetting

The USC is placed in an initial inactive state whenever external hardware drives the /RESET pin low. In this state, it stores the next data written to it in the Bus Configuration Register (BCR), whichever register address within it software uses for the write operation. Chapter 2 describes how the address used for the BCR write is actually important, in the sense that the address line connected to the A//B pin (the one used for channel selection in normal operation) determines whether the USC drives and receives the /WAIT//RDY pin as a "wait" or "acknowledge" handshake.

Aside from requiring the BCR write, software can reset a channel just as thoroughly and completely as a hardware reset does. *Resetting a Channel* in Chapter 4 describes how to do this, by first writing a 1 to the RTReset bit in the Channel Command / Address Register (CCAR10), and then writing zeroes to the whole CCAR.

After either a hardware or a software reset, all register bits in the USC are zero except for the following:

1. The following bits reflect the state of pins. The USC treats these as inputs until and unless software programs them as outputs.

   MISR14 /RxC
   MISR12 /TxC
   MISR10 /RxREQ
   MISR8 /TxREQ
   MISR6 /DCD
   MISR4 /CTS
   CCSR1 /TxACK
   CCSR0 /RxACK

2. The following bits are 1 because the TxFIFO is empty:

   TCSR0 TxEmpty
   TICR13 (indicates 32 empty entries)

Programming Order

The USC and other USC family members aren't as particular about the order in which software programs their register fields as are the members of Zilog's SCC family. Still, initializing registers in the wrong order can thoroughly confuse the USC's internal logic and make it do strange things. Always initialize the USC in the following order:

1. Set the pin configurations in the IOCR. While it's OK to change the modes and even the direction of a signal dynamically, it should be fairly obvious that if you're going to use pins in certain ways, they ought to be pointing in the right direction before telling internal logic to use them.
2. Select the clocking scheme in the CMCR and HCR. (It's OK to enable a BRG at this point if it's only used for clocking, but if it's used for interrupts it's probably best to wait until later.)
3. Set up most or all of the other mode and control bits in the Transmitter, Receiver, DMA channels, etc., but don't enable anything to run or operate until all of the basic modes and controls are in place. This procedure avoids messy interactions when one internal unit is trying to signal another before the latter is ready to listen.
4. Set up the initial Interrupt Arm bits and Interrupt Enable bits; it might be a good superstition to clear all the IP and IUS bits after doing this.
5. Enable whichever units need to run and operate initially. Some units might not want to be enabled until later, like enabling the Transmitter and Receiver after a call is established.
6. Finally, set the Master Interrupt Enable (MIE) bit. In general, you want to do this last so that interrupt service routines can assume that everything's set up in its starting configuration.

Using DMA to Initialize a Channel

Instead of software initializing a channel by writing the various registers itself, it can initialize a channel's Transmit DMA controller first and then use the DMA controller to initialize the serial channel. To do this:

1. Initialize the transmit DMA controller, including giving it the address of a sequence of bytes or 16-bit words that will initialize the channel. If there's only an 8-bit bus, structure this string as a series of byte pairs. The first byte of each pair goes into the LSByte of the Channel Command / Address Register (CCAR) to identify the destination (register address) of the second byte of the pair. If there's a 16-bit bus, structure the sequence as pairs of 16-bit words. The first word of each pair goes into CCAR to identify the destination of the second word of the pair.
2. Arrange the string/sequence to initialize the channel registers in the order described in the previous section. Make the **ChanLoad** bit (bit 7) of the first byte or word of each pair be 1, except make it 0 in the last entry of the sequence. If the RegAddr field in that last entry is non-zero, that is, if it doesn't point to the CCAR, the USC will request that the DMA controller fetch the second byte or word of the last pair and write it into the indicated register before finishing the initializing operation. If the RegAddr is zero, the USC will release /TxREQ and stop without accessing a following byte or word.
3. Program the DMA controller with the length of the initializing string. This should include at least the first byte or word of the last entry, and optionally the second word or byte, as described above.
4. Start the DMA controller so that it will respond to the channel's /TxREQ output.
5. Write a "Trigger Channel Load DMA" command (hex 20) to the MSByte of the CCAR.
6. Assuming the processor is set up to grant use of the bus to the DMA controller, the operation should complete very quickly. This should be verified by checking the DMA controller status.

Register Reference

The following pages include all of the fields in all of the registers in one of the USC's channels, plus the Bus Configuration Register which is common to both channels. They are arranged in alphabetical order by register name, like Table 2 in Chapter 2. (If you want to look up a register by its address/register number, look in Table 1 in Chapter 2 and then come back here...)

Register Addresses

These are located to the right of the name of each register on the following pages, and are shown as

d b aaaaa, where:

d	represents the state of D//C (1=high=data)
b	is 1 for a byte access on a 16-bit bus (it's just shown as "b" in all cases, like a placeholder);
aaaaa	is the actual register address, from AD5-1, AD13-9, or CCAR5-1.

Conditions/Context

Entries in this column indicate the conditions under which descriptions to their right apply or can validly be used. If an entry is blank, the description to the right always applies.

Description

Often entries in this column consist of one or more subentries of the form "value=description". If some possible values aren't shown, it may mean they are reserved (and should not be written) or that they will never be read. Or, particularly for single Read-Write bits, if the other case is obvious, it's left out. For example, for an entry like "1=dog is dead" we didn't feel obliged to add "0=dog is alive".

The following abbreviation is used in some entries in this column and "Conditions/Context":

:=	this "assignment operator" indicates that the value on its right is written to the field or bit on its left.

RW Status

This column includes the following codes for each register field:

RW	The field is fully under the control of software, and can be read and written.
RO	The field is read only; writing to it has no effect.
ROC	The bit is read-only; the USC clears it automatically after software reads it as 1.
WO	The field is write-only; reading it will either return zeroes or an unrelated item that's described next in the list.
WOC	The field is write only. After using its value the USC will clear it to zero, so that it points back to the indirect address register.
R,W1C	The bit is set by the USC hardware, writing a 1 to it clears it.
R,W1U	The bit is controlled by the USC hardware, writing a 1 to it "unlatches" it.

Bus Configuration Register (BCR)

No Address (First Write after /RESET)

SepAd	Reserved (must be 0)												16Bit	2Pulse IACK	SRight A
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
BCR15	SepAd	8-bit bus	1 if AD13-8 carry register addresses	WO	2: Bus Configuration Register (pp.11-12)
		16-bit bus	Must be 0		
BCR2	16Bit		0=8 bit data on AD7-0; 1=16 bit data on AD15-0	WO	
BCR1	2PulseIACK	/PITACK used	0=one pulse on /PITACK per interrupt acknowledge; 1=two pulses (Intel compatible)	WO	
BCR0	SRightA	Muxed AD	1=use AD6-0 as B/W, RegAddr, U/L 0=use AD7-1	WO	

Channel Command/Address Register (CCAR)

Register Address 0 b 00000

RTCmd					RT Reset	RTMode		Chan Load	B//W	RegAddr					U//L
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
CCAR15-12	RTCmd		00000=no operation 00001=Reserved 00010=Reset Highest IUS 00100=Trigger Channel Load DMA 00101=Trigger Rx DMA 00110=Trigger Tx DMA 00111=Trigger Rx and Tx DMA 01001=Purge Rx FIFO 01010=Purge Tx FIFO 01011=Purge Rx and Tx FIFO 01101=Load RCC 01110=Load TCC 01111=Load RCC and TCC 10001=Load TC0 10010=Load TC1 10011=Load TC0 and TC1 10100=Select Serial Data LSBit First 10101=Select Serial Data MSBit First 10110=Select D15-8 First 10111=Select D7-0 First 11xxx=Reserved	WO	4: Commands (pp.63-66)
CCAR10	RTReset		1=put channel in software Reset state 0=release it from Reset state	RW	4: Resetting a USC Channel (p.66)
CCAR9-8	RTMode		00=normal mode: Tx and Rx are independent 01=echo RxD to TxD 10=Local Loop TxD to RxD 11=internal Local Loop	RW	3: The RxD and TxD Pins (pp.27-28)
CCAR7	ChanLoad	Channel Load DMA	1=continue Channel Load operation; 0=terminate it	RW	7: Using DMA to Initialize a Channel (p.94)
CCAR6	B//W	16 bit bus	0=16-bit access to register selected by RegAddr 1=access MS or LS byte of register	WOC	2: Register Addressing (pp.12-15)
CCAR5-1	RegAddr		register address for next access to CCAR (see Table 1)	WOC	
CCAR0	U//L		1=access MSByte of reg selected by RegAddr 0=access LSByte or whole 16-bit register	WOC	

Channel Command/Status Register (CCSR)

Register Address 0 b 00010

RCCF Ovflo	RCCF Avail	Clear RCCF	DPLL Sync	DPLL 2Miss	DPLL 1Miss	DPLLEdge	On Loop	Loop Send	Resrvd	TxResidue	/TxACK	/RxACK
15	14	13	12	11	10	9 8	7	6	5	4 3 2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
CCSR15	RCCF Ovflo	RCC Enabled	1=RCC FIFO overflow (4+1 frames)	RO	4: DMA Support Features: The RCC FIFO (p.60)
CCSR14	RCCF Avail		1=RCC FIFO not empty	RO	
CCSR13	Clear RCCF		1=purge RCC FIFO, clear RCCF Ovflo and RCCF Avail to 0	WO	
CCSR12	DPLL Sync		1=DPLL in sync	R,W1C	3: More About the DPLL (pp.26-27)
CCSR11	DPLL2Miss	Biphase	1=DPLL has seen 2 consecutive missing clocks	R,W1C	
CCSR10	DPLL1Miss	Biphase, CVOK=0	1=DPLL has seen a missing clock	R,W1C	
CCSR9-8	DPLL Edge		00=DPLL resyncs on rising and falling edges	RW	
		NRZ modes only	01=DPLL sees rising edges only; 10=DPLL sees falling edges only; 11=DPLL free-runs like CTR1,0		
CCSR7	OnLoop	Slaved Monosync	1=Transmit is or has been active (cleared only by leaving Slave Monosync mode)	RO	4: Slaved Monosync Mode (p.45) 4: HDLC/SDLC Loop Mode (pp.49-50)
		H/SDLC Loop	1=USC has inserted itself in the loop		
CCSR6	LoopSend	H/SDLC Loop	1=Transmit actively sending; 0=Transmit repeating Receive	RO	4: HDLC/SDLC Loop Mode (pp.49-50)
CCSR4-2	TxResidue	H/SDLC, H/SDLC Loop	000=last character of Transmit frame contains 8 bits; 001-111= last character contains 1-7 bits	RW	4: HDLC/SDLC Mode: Frame Length Residuals (pp.48-49)
CCSR1	/TxACK	TxAMode (HCR7-6) =00	1=/TxACK pin is low	RO	3: The /TxACK and /RxACK Pins (p.32)
CCSR0	/RxACK	RxAMode (HCR3-2) =00	1=/RxACK pin is low		

Channel Control Register (CCR)

Register Address 0 b 00011

TxCtrlBlk	Wait4 Tx Trig	Rsrvd	Async:TxShaveL		RxStatBlk	Wait4 Rx Trig	Reserved (0)
			Sync:TxPreL	Sync:TxPrePat			
15 14	13	12	11 10	9 8	7 6	5	4 3 2 1 0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
CCR15-14	TxCtrlBlk		00=don't use Transmit Control Blocks; 01=use 16-bit TCB's; 10=use 32-bit TCB's	RW	4: DMA Support Features: Transmit Control Blocks (pp.60-61)
CCR13	Wait4TxTrig	Sync	1=hold Transmit DMA Request between frames/ messages until software issues "Trigger Tx DMA" command	RW	4: Synchronizing Frames/ Messages with Software Response (p.70)
CCR11-8	TxShaveL	Async, CMR15=1	shave the number of Stop bits specified by TxSubMode CMR14 by (15 minus the value in this field)/16 bit times	RW	4: Asynchronous Mode (pp.38-39)
CCR11-10	TxPreL	Sync w/ Preamble	00=send 8-bit Preamble; 01=16-bit; 10=32-bit; 11=64-bit	RW	4: Between Frames, Messages, or Characters (pp.68-70)
CCR9-8	TxPrePat	Sync w/ Preamble	00=all-zero Preamble; 01=all ones; 10=101010...; 11=010101...	RW	
CCR7-6	RxStatBlk		00=don't use Receive Status Blocks;	RW	4: DMA Support Features: Receive Status Blocks (pp.61-63)
		Ext Sync, T. Bisync, H/SDLC, 802.3, ACV (1553B)	01=use 16-bit RSB's; 10=use 32-bit RSB's		
CCR5	Wait4RxTrig	Sync	1=hold Receive DMA Request between frames/ messages until software issues "Trigger Rx DMA" command	RW	4: Synchronizing Frames/ Messages with Software Response (p.70)

Channel Mode Register (CMR)

Register Address 0 b 00001

TxSubMode	TxMode	RxSubMode	RxMode

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Because the content of the SubMode fields depends on the Mode fields, the following descriptions are grouped by mode. TxSubMode and RxSubMode bits that are not shown for a particular Mode value are Reserved in that mode and should be programmed with zeroes.

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
CMR11-8	TxMode		**0000=Asynchronous**	RW	4: Asynchronous Mode (pp.38-39)
CMR15-14	TxSubMode	TxMode=0	00=send one stop bit; 01=two stop bits; 10=1 shaved stop bit (per CCR11-8); 11=2 shaved stop bits	RW	
CMR13-12			00=16 TxCLKs/Tx bit; 01=32 TxCLKs/Tx bit; 10=64 TxCLKs/Tx bit		
CMR3-0	RxMode		**0000=Asynchronous**	RW	
CMR5-4	RxSubMode	RxMode=0	00=16 RxCLKs/Rx bit; 01=32 RxCLKs/Rx bit; 10=64 RxCLKs/Rx bit	RW	
CMR11-8	TxMode		**0001=Reserved**	RW	
CMR3-0	RxMode		**0001=External Sync**		4: External Sync Mode (p.42)
CMR11-8	TxMode		**0010=2=Isochronous**	RW	4: Isochronous Mode (p.39)
CMR14	TxSubMode	TxMode=2	0=send one stop bit; 1=two stop bits	RW	
CMR3-0	RxMode		**0010=2=Isochronous**	RW	
CMR11-8	TxMode		**0011=3=Async w/Code Violations (1553B)**	RW	4: Async w/Code Violations Mode (pp.40-42)
CMR15-14	TxSubMode	TxMode=3	00=send one stop bit; 01=two stop bits; 10=no stop bits	RW	
CMR13			0=Tx length <= 8 bits per TxLength (TMR4-2); 1=Tx length is 8 more than indic. by TxLength		
CMR12			0=send Data words; 1=send Command/Status words		
CMR3-0	RxMode		**0011=3=Async w/Code Violations (1553B)**	RW	
CMR4	RxSubMode	RxMode=3	0=Rx length <= 8 bits per RxLength (RMR4-2); 1=Rx length is 8 more than indic. by RxLength	RW	
CMR11-8	TxMode		**0100=4=Monosync**	RW	4: Monosync and Bisync Modes (pp.42-43)
CMR15	TxSubMode	TxMode=4	1=send CRC on Tx Underrun	RW	
CMR13			1=send Preamble before opening Sync		
CMR12			0=send 8-bit Syncs; 1=send Syncs per TxLength		
CMR3-0	RxMode		**0100=4=Monosync**	RW	
CMR5	RxSubMode	RxMode=4	1=strip received Syncs; 0=include them in RxFIFO and CRC calculation	RW	
CMR4			0=expect 8-bit Syncs; 1=expect Syncs per RxLength		
CMR11-8	TxMode		**0101=5=Bisync**	RW	
CMR15	TxSubMode	TxMode=5	1=send CRC on Tx Underrun	RW	
CMR14			0=send closing/idle SYNs from TSR15-8; 1=send closing/idle SYN0/SYN1 (TSR7-0/15-8)		
CMR13			1=send Preamble before opening Sync		
CMR12			0=send 8-bit Syncs; 1=send Syncs per TxLength		
CMR3-0	RxMode		**0101=5=Bisync**	RW	
CMR5	RxSubMode	RxMode=5	1=strip received Syncs; 0=include them in RxFIFO and CRC calculation	RW	
CMR4			0=expect 8-bit Syncs; 1=expect Syncs per RxLength		

Channel Mode Register (CMR) -- Continued

Because the content of the SubMode fields depends on the Mode fields, the following descriptions are grouped by mode. TxSubMode and RxSubMode bits that are not shown for a particular Mode value are Reserved in that mode and should be programmed with zeroes.

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
CMR11-8	TxMode		**0110=6=HDLC/SDLC**	RW	4: HDLC/SDLC Mode (pp.46-49)
CMR15-14	TxSubMode	TxMode=6	00=send 7-bit Abort on Tx Underrun; 01=send 15-bit Abort; 10=send Flag; 11=send CRC then Flag	RW	
CMR13			1=send Preamble before opening Flag		
CMR12			1=consecutive idle Flags share a 0 (11111101111111...); 0=(11111100111111...)		
CMR3-0	RxMode		**0110=6=HDLC/SDLC**	RW	

CMR7-4	RxSubMode	RxMode=6	xx00=no Address or Control field handling;
			xx01=1-byte Address only;
			x010=1-byte Address, 1-byte Control;
			x110=1-byte Address, 2-byte Control;
			0011=Extended Address, 1-byte Control;
			0111=Extended Address, 2-byte Control;
			1011=Extended Address, Control >= 2 bytes;
			1111=Extended Address, Control >= 3 bytes
				RW	

CMR11-8	TxMode		**0111=7=Transparent Bisync**	RW	4: Transparent Bisync Mode (pp.43-44)
CMR15	TxSubMode	TxMode=7	1=send CRC on Tx Underrun	RW	
CMR14			0=send closing/idle SYNs; 1=send closing/idle DLE-SYNs		
CMR13			1=send Preamble before opening DLE-SYN		
CMR12			0=send ASCII control characters; 1=send EBCDIC		
CMR3-0	RxMode		**0111=7=Transparent Bisync**	RW	
CMR4	RxSubMode	RxMode=7	0=look for ASCII control characters; 1=look for EBCDIC	RW	
CMR11-8	TxMode		**1000=8=Nine Bit**	RW	4: Nine Bit Mode (pp.39-40)
CMR15	TxSubMode	TxMode=8	0=send 9th bit 0 (data); 1=send 9th bit 1 (address)	RW	
CMR14			0=send eight data bits; 1=send seven data bits plus parity		
CMR13-12			00=16 TxCLKs/Tx bit; 01=32 TxCLKs/Tx bit; 10=64 TxCLKs/Tx bit		
CMR3-0	RxMode		**1000=8=Nine Bit**	RW	
CMR5-4	RxSubMode	RxMode=8	00=16 RxCLKs/Rx bit; 01=32 RxCLKs/Rx bit; 10=64 RxCLKs/Rx bit	RW	
CMR11-8	TxMode		**1001=9=802.3 (Ethernet)**	RW	4: 802.3 (Ethernet) Mode (pp.45-46)
CMR15	TxSubMode	TxMode=9	1=send CRC on Tx Underrun	RW	
CMR3-0	RxMode		**1001=9=802.3 (Ethernet)**	RW	
CMR4	RxSubMode	RxMode=9	0=receive all frames; 1=match 16-bit Destination Address vs. RSR	RW	
CMR11-8	TxMode		**101x=10-11=Reserved**		
CMR3-0	RxMode				
CMR11-8	TxMode		**1100=12=Slaved Monosync**	RW	4: Slaved Monosync Mode (p.45)
CMR15	TxSubMode	TxMode	1=send CRC on Tx Underrun	RW	
CMR13		=12	0=don't send (stop sending at EOM); 1=send a(nother) message		
CMR12			0=send 8-bit Syncs; 1=send Syncs per TxLength		
CMR3-0	RxMode		**1100=12=Reserved** (use RxMode=0100=4= Monosync with TxMode=1100=12)		

Channel Mode Register (CMR) -- Continued

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
CMR11-8	TxMode		**1101=13=Reserved**		
CMR3-0	RxMode				
CMR11-8	TxMode		**1110=14=HDLC/SDLC Loop**	RW	4: HDLC/SDLC Loop Mode (pp.49-50)
CMR15-14	TxSubMode	TxMode =14	00=send 7-bit Abort on Tx Underrun; 01=send 15-bit Abort; 10=send Flag; 11=send CRC then Flag	RW	
CMR13			(initially) 0=Transmit disabled; 1=insert into loop; (once inserted) 0=repeat Rx to Tx; 1=send	RW	
CMR12			1=consecutive idle Flags share a 0 (11111101111111...); 0=(11111100111111...)	RW	
CMR3-0	RxMode		**1110=14=Reserved** (use RxMode=0110=6= HDLC/SDLC with TxMode=1110=14)		
CMR11-8	TxMode		**1111=15=Reserved**		
CMR3-0	RxMode				

Clock Mode Control Register (CMCR)

Register Address 0 b 01000

CTR1Src		CTR0Src		BRG1Src		BRG0Src		DPLLSrc		TxCLKSrc			RxCLKSrc		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
CMCR15-14	CTR1Src		00=CTR1 disabled; 01=CTR1 input is PORT1/CLK1; 10=/RxC pin; 11=/TxC pin	RW	3: Tx and Rx Clocking: CTR0 and CTR1 (p.21)
CMCR13-12	CTR0Src		00=CTR0 disabled; 01=CTR0 input is PORT0/CLK0; 10=/RxC pin; 11=/TxC pin	RW	
CMCR11-10	BRG1Src		00=BRG1 input is CTR0 output; 01=CTR1 output; 10=/RxC pin; 11=/TxC pin	RW	3: Tx and Rx Clocking: The Baud Rate Generators (pp.21-23)
CMCR9-8	BRG0Src		00=BRG0 input is CTR0 output; 01=CTR1 output; 10=/RxC pin; 11=/TxC pin	RW	
CMCR7-6	DPLLSrc		00=DPLL input is BRG0 output; 01=BRG1 output; 10=/RxC pin; 11=/TxC pin	RW	3: Tx and Rx Clocking: Intro to the DPLL (pp.23-24)
CMCR5-3	TxCLKSrc		000=no TxCLK (Transmit disabled); 001=TxCLK is /RxC; 010=/TxC; 011=DPLL Tx output; 100=BRG0 output; 101=BRG1 output; 110=CTR0 output; 111=TxCLK is CTR1 output	RW	3: Tx and Rx Clocking: TxCLK and RxCLK Selection (p.24)
CMCR2-0	RxCLKSrc		000=no RxCLK (Receive disabled); 001=RxCLK is /RxC; 010=/TxC; 011=DPLL Rx output; 100=BRG0 output; 101=BRG1 output; 110=CTR0 output; 111=RxCLK is CTR1 output	RW	

Daisy Chain Control Register (DCCR)

Register Address 0 b 01101

IUS Op (WO)		RS IUS	RD IUS	TS IUS	TD IUS	IOP IUS	Misc IUS	IP Op (WO)		RS IP	RD IP	TS IP	TD IP	IOP IP	Misc IP
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
DCCR15-14	IUS Op	write	0x=no operation; 10=clear IUS bits selected by 1s in DCCR13-8; 11=set IUS bits selected by 1s in DCCR13-8	WO	6: Interrupt Pending and Under Service Bits (p.90)
DCCR13	RS IUS	read	1=Receive Status interrupt under service	RO	6: Interrupt Pending and Under Service Bits (p.90); 6: Rx Status Interrupt Sources and IA Bits (pp.85-86)
		write	1=set or clear Receive Status IUS per IUS Op; 0=no change	WO	
DCCR12	RD IUS	read	1=Receive Data interrupt under service	RO	6: Interrupt Pending and Under Service Bits (p.90); 6: Rx Data Interrupts (p.86)
		write	1=set or clear Receive Data IUS per IUS Op; 0=no change	WO	
DCCR11	TS IUS	read	1=Transmit Status interrupt under service	RO	6: Interrupt Pending and Under Service Bits (p.90); 6: Tx Status Interrupt Sources and IA Bits (pp.86-88)
		write	1=set or clear Transmit Status IUS per IUS Op; 0=no change	WO	
DCCR10	TD IUS	read	1=Transmit Data interrupt under service	RO	6: Interrupt Pending and Under Service Bits (p.90); 6: Transmit Data Interrupts (pp.88-89)
		write	1=set or clear Transmit Data IUS per IUS Op; 0=no change	WO	
DCCR9	IOP IUS	read	1=I/O Pin interrupt under service	RO	6: Interrupt Pending and Under Service Bits (p.90); 6: I/O Pin Interrupt Sources and IA Bits (p.89)
		write	1=set or clear I/O Pin IUS per IUS Op; 0=no change	WO	
DCCR8	Misc IUS	read	1=Miscellaneous interrupt under service	RO	6: Interrupt Pending and Under Service Bits (p.90); 6: Miscellaneous Int. Sources and IA Bits (pp.89-90)
		write	1=set or clear Miscellaneous IUS per IUS Op; 0=no change	WO	
DCCR7-6	IP Op	write	00=no operation; 01=clear IP and IUS bits sel by 1s in DCCR5-0; 10=clear IP bits selected by 1s in DCCR5-0; 11=set IP bits selected by 1s in DCCR5-0	WO	6: Interrupt Pending and Under Service Bits (p.90)
DCCR5	RS IP	read	1=Receive Status interrupt pending	RO	6: Interrupt Pending and Under Service Bits (p.90); 6: Rx Status Interrupt Sources and IA Bits (pp.85-86)
		write	1=set or clear Receive Status IP/IUS per IP Op; 0=no change	WO	
DCCR4	RD IP	read	1=Receive Data interrupt pending	RO	6: Interrupt Pending and Under Service Bits (p.90); 6: Rx Data Interrupts (p.86)
		write	1=set or clear Receive Data IP/IUS per IP Op; 0=no change	WO	
DCCR3	TS IP	read	1=Transmit Status interrupt pending	RO	6: Interrupt Pending and Under Service Bits (p.90); 6: Tx Status Interrupt Sources and IA Bits (pp.86-88)
		write	1=set or clear Transmit Status IP/IUS per IP Op; 0=no change	WO	
DCCR2	TD IP	read	1=Transmit Data interrupt pending	RO	6: Interrupt Pending and Under Service Bits (p.90); 6: Transmit Data Interrupts (pp.88-89)
		write	1=set or clear Transmit Data IP/IUS per IP Op; 0=no change	WO	
DCCR1	IOP IP	read	1=I/O Pin interrupt pending	RO	6: Interrupt Pending and Under Service Bits (p.90); 6:I/O Pin Interrupt Sources and IA Bits (p.89)
		write	1=set or clear I/O Pin IP/IUS per IP Op; 0=no change	WO	
DCCR0	Misc IP	read	1=Miscellaneous interrupt pending	RO	6: Interrupt Pending and Under Service Bits (p.90); 6: Miscellaneous Int, Sources and IA Bits (pp.89-90)
		write	1=set or clear Miscellaneous IP/IUS per IP Op; 0=no change	WO	

Hardware Configuration Register (HCR)

Register Address 0 b 01001

CTR0Div		CTR1 DSel	CVOK	DPLLDiv		DPLLMode		TxAMode		BRG1S	BRG1E	RxAMode		BRG0S	BRG0E
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
HCR15-14	CTR0Div		00=CTR0 divides by 32; 01=/16; 10=/8; 11=/4	RW	3: Tx and Rx Clocking: CTR0 and CTR1 (p.21)
HCR13	CTR1DSel		0=CTR0Div determines CTR1 divisor; 1=DPLLDiv determines CTR1 divisor	RW	
HCR12	CVOK	Biphase	1=don't report single code violations	RW	3: More About the DPLL (pp.26-27)
HCR11-10	DPLLDiv		00=DPLL divides by 32; 01=/16; 10=/8; 11=don't use for DPLL (/4 for CTR1)	RW	3: Tx and Rx Clocking: Intro to the DPLL (pp.23-24)
HCR9-8	DPLLMode		00=disable DPLL; 01=run DPLL for NRZ modes; 10=run DPLL for Biphase-Mark or -Space; 11=run DPLL for either Biphase-Level mode	RW	3: More About the DPLL (pp.26-27)
HCR7-6	TxAMode		00=/TxACK pin is a general purpose input; 01=/TxACK is a Tx DMA Acknowledge input; 10=drive /TxACK Low; 11=drive /TxACK High	RW	3: The /TxACK and /RxACK Pins (p.32)
HCR5	BRG1S		1=BRG1 single cycle mode; 0=continuous	RW	3: Tx and Rx Clocking: The Baud Rate Generators (pp.21-23)
HCR4	BRG1E		1=enable BRG1	RW	
HCR3-2	RxAMode		00=/RxACK pin is a general purpose input; 01=/RxACK is a Rx DMA Acknowledge input; 10=drive /RxACK Low; 11=drive /RxACK High	RW	3: The /TxACK and /RxACK Pins (p.32)
HCR1	BRG0S		1=BRG0 single cycle mode; 0=continuous	RW	3: Tx and Rx Clocking: The Baud Rate Generators (pp.21-23)
HCR0	BRG0E		1=enable BRG0	RW	

Input/Output Control Register (IOCR)

Register Address 0 b 01011

CTSMode	DCDMode	TxRMode	RxRMode	TxDMode	TxCMode	RxCMode
15 14	13 12	11 10	9 8	7 6	5 4 3	2 1 0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
IOCR15-14	CTSMode		0x=/CTS pin is low-active Clear To Send input; 10=drive /CTS Low; 11=drive /CTS High	RW	3: The /CTS Pin (pp.30-31)
IOCR13-12	DCDMode		00=/DCD is low-active Rx Carrier Detect input; 01=/DCD is low-active Rx Sync Detect input; 10=drive /DCD Low; 11=drive /DCD High	RW	3: The /DCD Pin (pp.29-30)
IOCR11-10	TxRMode		00=/TxREQ pin is an input; 01=drive /TxREQ with Transmit DMA Request; 10=drive /TxREQ Low; 11=drive /TxREQ High	RW	3: The /RxREQ and /TxREQ Pins (pp.31-32)
IOCR9-8	RxRMode		00=/RxREQ pin is an input; 01=drive /RxREQ with Receive DMA Request; 10=drive /RxREQ Low; 11=drive /RxREQ High	RW	
IOCR7-6	TxDMode		00=drive /TxD with Transmitter output; 01=release /TxD to high impedance; 10=drive /TxD Low; 11=drive /TxD High	RW	3: The /RxD and /TxD Pins (pp.27-28)
IOCR5-3	TxCMode		000=/TxC pin is an input; 001=drive /TxC with TxCLK; 010=drive /TxC with Transmit char clock; 011=drive /TxC with Transmit Complete; 100=drive /TxC with output of BRG0; 101=drive /TxC with output of BRG1; 110=drive /TxC with output of CTR1; 111=drive /TxC with Tx output of DPLL	RW	3: The /RxC and /TxC Pins (p.31)
IOCR2-0	RxCMode		000=/RxC pin is an input; 001=drive /RxC with RxCLK; 010=drive /RxC with Receive char clock; 011=drive /RxC with /RxSYNC; 100=drive /RxC with output of BRG0; 101=drive /RxC with output of BRG1; 110=drive /RxC with output of CTR0; 111=drive /RxC with Rx output of DPLL	RW	

Interrupt Control Register (ICR)

Register Address 0 b 01100

MIE	DLC	NV	VIS	Rsrvd	IE Op (WO)	RS IE	RD IE	TS IE	TD IE	IOP IE	Misc IE
15	14	13	12 11 10 9	8	7 6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
ICR15	MIE		1=enable interrupts from this channel	RW	6: Channel Interrupt Options (pp.90-91)
ICR14	DLC		1=disable Interrupt Enable Out (IEO)	RW	
ICR13	NV		1=don't return a vector during /INTACK cycle	RW	
ICR12-9	VIS		0xxx=interrupt vectors never include status; 100x=interrupt vectors always include status; 1010=vectors include status except for Misc; 1011=vectors include status only for TD, TS, RD and RS 1100=vectors include status only for TS, RD, and RS 1101=vectors include status only for RD and RS 1110=vectors include status only for RS 1111=interrupt vectors never include status	RW	
ICR7-6	IE Op	write	0x=no operation; 10=clear the IE bits selected by 1s in ICR5-0; 11=set the IE bits selected by 1s in ICR5-0	WO	6: Interrupt Enable Bits (p.90)
ICR5	RS IE	read	1=Receive Status interrupt enabled	RO	
		write	1=set or clear Receive Status IE per IE Op; 0=no change	WO	
ICR4	RD IE	read	1=Receive Data interrupt enabled	RO	
		write	1=set or clear Receive Data IE per IE Op; 0=no change	WO	
ICR3	TS IE	read	1=Transmit Status interrupt enabled	RO	
		write	1=set or clear Transmit Status IE per IE Op; 0=no change	WO	
ICR2	TD IE	read	1=Transmit Data interrupt enabled	RO	
		write	1=set or clear Transmit Data IE per IE Op; 0=no change	WO	
ICR1	IOP IE	read	1=I/O Pin interrupt enabled	RO	
		write	1=set or clear I/O Pin IE per IE Op; 0=no change	WO	
ICR0	Misc IE	read	1=Miscellaneous interrupt enabled	RO	
		write	1=set or clear Miscellaneous IE per IE Op; 0=no change	WO	

Interrupt Vector Register (IVR)

Register Address 0 b 01010

Interrupt Vector7-4 (RO)				Type Code (RO)			IV0 (RO)	Interrupt Vector (RW)							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
IVR15-12		read	as software wrote IVR7-4	RO	6: Interrupt Vectors (p.91)
IVR11-9	TypeCode	IVR15-8, or IAck w/ highest pending type enabled by ICR12-9	highest pending interrupt type: 000=no interrupt type pending; 001=Misc: 010=I/O Pin; 011=Transmit Data; 100=Transmit Status; 101=Receive Data; 110=Receive Status	RO	
IVR8			as software wrote IVR0	RO	
IVR7-0		read/write IVR7-0, or IAck w/ highest pending type blocked by ICR12-9	basic 8-bit interrupt vector (reads back as software wrote it)	RW	

Miscellaneous Interrupt Status Register (MISR)

Register Address 0 b 01110

RxCL/U	/RxC	TxCL/U	/TxC	RxRL/U	/RxR	TxRL/U	/TxR	DCDL/U	/DCD	CTSL/U	/CTS	RCC Under L/U	DPLL DSync L/U	BRG1 L/U	BRG0 L/U
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
MISR15	RxCL/U	Read	1=one or more transition(s) enabled by SICR15-14 has (have) occurred on the /RxC pin	R,W1U	3: The /RxC and /TxC Pins (p..31)
		Write	1=open the latches for /RxC and for this bit		
MISR14	/RxC	RxCL/U=1	1=the (first such) enabled transition was a rising edge; 0=it was a falling edge	RO	
		RxCL/U=0	1=the /RxC pin is low; 0=it's high		
MISR13	TxCL/U	Read	1=one or more transition(s) enabled by SICR13-12 has (have) occurred on the /TxC pin	R,W1U	
		Write	1=open the latches for /TxC and for this bit		
MISR12	/TxC	TxCL/U=1	1=the (first such) enabled transition was a rising edge; 0=it was a falling edge	RO	
		TxCL/U=0	1=the /TxC pin is low; 0=it's high		
MISR11	RxRL/U	Read	1=one or more transition(s) enabled by SICR11-10 has (have) occurred on the /RxREQ pin	R,W1U	3: /RxREQ and /TxREQ Pins (pp.31-32)
		Write	1=open the latches for /RxR and for this bit		
MISR10	/RxR	RxRL/U=1	1=the (first such) enabled transition was a rising edge; 0=it was a falling edge	RO	
		RxRL/U=0	1=the /RxREQ pin is low; 0=it's high		
MISR9	TxRL/U	Read	1=one or more transition(s) enabled by SICR9-8 has (have) occurred on the /TxREQ pin	R,W1U	
		Write	1=open the latches for /TxR and for this bit		
MISR8	/TxR	TxRL/U=1	1=the (first such) enabled transition was a rising edge; 0=it was a falling edge	RO	
		TxRL/U=0	1=the /TxREQ pin is low; 0=it's high		
MISR7	DCDL/U	Read	1=one or more transition(s) enabled by SICR7-6 has (have) occurred on the /DCD pin	R,W1U	3: The /DCD Pin (pp.29-30)
		Write	1=open the latches for /DCD and for this bit		
MISR6	/DCD	DCDL/U=1	1=the (first such) enabled transition was a rising edge; 0=it was a falling edge	RO	
		DCDL/U=0	1=the /DCD pin is low; 0=it's high		
MISR5	CTSL/U	Read	1=one or more transition(s) enabled by SICR5-4 has (have) occurred on the /CTS pin	R,W1U	3: The /CTS Pin (pp.30-31)
		Write	1=open the latches for /CTS and for this bit		
MISR4	/CTS	CTSL/U=1	1=the (first such) enabled transition was a rising edge; 0=it was a falling edge	RO	
		CTSL/U=0	1=the /CTS pin is low; 0=it's high		
MISR3	RCC Under L/U		1=RCC FIFO has counted down past 0 (Receive frame/message longer than max allowed)	R,W1U	4: DMA Support Features: The RCC FIFO (p.60)
MISR2	DPLLDSync L/U		1=DPLL has lost sync	R,W1U	3: More About the DPLL (pp.26-27); 6: Miscellaneous Interrupt Sources and IA Bits (p.89-90)
MISR1	BRG1 L/U		1=BRG1 has counted down to 0	R,W1U	3: Tx and Rx Clocking:
MISR0	BRG0 L/U		1=BRG0 has counted down to 0	R,W1U	The Baud Rate Generators (pp.21-23)

Receive Character Count Register (RCCR)

Register Address 0 b 10110

ending count of oldest received frame/message in RCC FIFO															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
RCCR15-0		RCCAvail (CCSR14) =1	final RCC value of oldest received frame/ message in the RCC FIFO	RO	4: DMA Support Features: The RCC FIFO (p.60)

Receive Command/Status Register (RCSR)

Register Address 0 b 10010

RCmd (WO)				RxResidue			ShortF/ CVType	Exited Hunt	Idle Rcved	Break /Abort	Rx Bound	CRCE /FE	PE	Rx Over	Rx Avail
2ndBE	1stBE														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
RCSR15-12	RCmd	Sync	0000=no operation; 0001=Reserved 0010=Clear Receive CRC Generator 0011=Enter Hunt Mode; 0100=Reserved 0101=Select RICRHi=TxFIFO Status 0110=Select RICRHi=/INT Level 0111=Select RICRHi=/RxREQ Level 1xxx=Reserved	WO	4: Commands (pp.63-66)
RCSR15	2ndBE	Last RDR read was 16 bits	1=2nd-oldest byte in RxFIFO had RxBound, PE, or RxOver when RDR was last read	RO	4: Status Reporting: Detailed Status in the RCSR (pp.55-56)
RCSR14	1stBE		1=oldest byte in RxFIFO had RxBound, PE, or RxOver when RDR was last read	RO	
RCSR11-9	RxResidue	H/SDLC	000=frame ended at character boundary 001-111=number of extra bits at end	RO	4: HDLC/SDLC Mode: Frame Length Residuals (pp.48-49)
RCSR8	ShortF/ CVType	H/SDLC, CMR7-4 <>xx00	1=received frame ended before Address/Control fields (see Note 1)	R,W1U or RO	4: Status Reporting: Detailed Status in the RCSR (pp.55-56)
		ACV (1553B)	0=received Data word 1=received Command/Status word (see Note 1)		
RCSR7	ExitedHunt		1=receiver has left Hunt mode	R,W1U	
RCSR6	IdleRcved		1=15 or 16 ones received	R,W1U	
RCSR5	Break/Abort	Async	1=Break received	R,W1U	
		H/SDLC	1=Abort received		
RCSR4	RxBound	Nine Bit	1=address character (see Note 2)	R,W1C or RO	
		ACV (1553B)	1=2nd (or only) byte of word (see Note 2)		
		Ext Sync, T. Bisync	1=end of message (see Note 2)		
		H/SDLC, 802.3	1=end of frame (see Note 2)		
RCSR3	CRCE/FE	Sync	1=CRC not correct (at this point; see Note 1)	RO	
		Async	1=framing error (Stop bit = zero/space; see Note 1)		
RCSR2	PE		1=parity error (see Note 2)	R,W1C or RO	
RCSR1	RxOver		1=RxFIFO overflow (see Note 2)	R,W1C RO??	
RCSR0	RxAvail		1=RxFIFO is not empty	RO	

Note 1: the USC carries these bits through the RxFIFO with data characters; they may represent the status of the oldest character or two currently in the FIFO, or of the last one or two read from it, as described in the referenced Chapter/Section.

Note 2: the USC carries these bits through the RxFIFO with data characters; they may represent the status of the oldest character or two currently in the FIFO, of the last one or two read from it, or may be a cumulative/latched bit, as described in the referenced Chapter/Section.

Receive Count Limit Register (RCLR)

Register Address 0 b 10101

starting value for Receive Character Counter															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
RCLR15-0			starting value for RCC: 0=disable RCC; FFFF=enable RCC, no set max frame/message length; else maximum allowed length	RW	4: DMA Support Features: The Character Counters (pp.57-59)

Receive Data Register (RDR)

Register Address 0 b 1x000 or 1 b xxxxx

received character: read only using 16-bit operation								received character: 8- or 16-bit read							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
RDR15-8		16 bit bus	the "other" received character in a 16-bit read (may be the oldest or 2nd-oldest per "Select D15-8 First" or "Select D7-0 First" commands in RTCmd [CCAR15-11])	RO	4: The Data Registers and the FIFOs (pp.66-68)
RDR7-0			received character	RO	

Receive Interrupt Control Register (RICR)

Register Address 0 b 10011

"RxFIFO Status" If last RCSR15-12 command 4-7 was 5 "Rx /INT level" If last RCSR15-12 command 4-7 was 6 "/RxREQ level" If last RCSR15-12 command 4-7 was 7	Exited Hunt IA	Idle Rcved IA	Break/ Abort IA	Rx Bound IA	Word Status	Parity Error IA	RxOver IA	TC0R Sel
15 14 13 12 11 10 9 8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
RICR15-8		5 written to RCmd (RCSR15-12) or Reset since 6 or 7 written to RCmd	the number of characters/bytes/octets currently in the RxFIFO	RO	4: The Data Registers and the FIFOs (pp.66-68)
RICR15-8		6 written to RCmd (RCSR15-12) since 5 or 7 written to RCmd	number of characters/bytes/octets in the RxFIFO, above which to request a Receive Data interrupt	RW	6: Receive Data Interrupts (p.86)
RICR15-8		7 written to RCmd (RCSR15-12) since 5 or 6 written to RCmd	number of characters/bytes/octets in the RxFIFO, above which to request Receive DMA transfer	RW	5: DMA Requests by the Receiver and Transmitter (pp.73-75)
RICR7	ExitedHunt IA		1=arm interrupts on ExitedHunt (RCSR7)	RW	6: Receive Status Interrupt Sources and IA Bits (pp.85-86)
RICR6	IdleRcvedIA		1=arm interrupts on IdleRcved (RCSR6)	RW	
RICR5	Break/Abort IA		1=arm interrupts on Break/Abort (RCSR5)	RW	
RICR4	RxBound IA		1=arm interrupts on RxBound (RCSR4)	RW	
RICR3	WordStatus		0="queued" status in RCSR reflects oldest character in RxFIFO; 1=two oldest characters	RW	4: Status Reporting (pp.52-56)
RICR2	PE IA		1=arm interrupts on PE (RCSR2)	RW	6: Receive Status Interrupt Sources & IA Bits (pp.85-86)
RICR1	RxOver IA		1=arm interrupts on RxOver (RCSR1)	RW	
RICR0	TC0R Sel		0=select Time Constant value for reading TC0R 1=capture current count for reading TC0R	RW	3: Tx and Rx Clocking: The Baud Rate Generators (pp.21-23)

Receive Mode Register (RMR)

Register Address 0 b 10001

RxDecode	RxCRCType	RxCRC Start	RxCRC Enab	Rsrvd	RxParType	RxPar Enab	RxLength	RxEnable
15 14 13	12 11	10	9	8	7 6	5	4 3 2	1 0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
RMR15-13	RxDecode		000=RxD not encoded ("NRZ"); 001=invert polarity of RxD ("NRZB"); 010=decode RxD NRZI-Mark; 011=decode RxD NRZI-Space; 100=decode RxD Biphase-Mark (FM1); 101=decode RxD Biphase-Space (FM0); 110=decode RxD Biphase-Level (Manchester); 111=decode RxD Differential Biphase-Level	RW	3: Data Formats and Encoding (pp.25-26)
RMR12-11	RxCRCType	Sync	00=use 16-bit CRC-CCITT for Rx; 01=use CRC-16 for Rx; 10=use 32-bit Ethernet CRC for Rx	RW	4: Cyclic Redundancy Checking (pp.50-51)
RMR10	RxCRCStart	Sync	0=start Receive CRC generator as all-zeroes; 1=all ones	RW	
RMR9	RxCRCEnab	Sync	1=include Receive characters in CRC	RW	
RMR7-6	RxParType		00=Receive Parity Even; 01=Odd; 10=Zero (Space); 11=One (Mark)	RW	4: Parity Checking (p.52)
RMR5	RxParEnab		1=accumulate & check Parity bits	RW	
RMR4-2	RxLength		000=receive eight bit characters; 001-111=receive 1-7 bit characters	RW	4: The Mode Registers: Character Length (pp.45-46)
RMR1-0	RxEnable		00=disable Receiver (immediately); 01=disable Rx at end of message/frame/char; 10=enable Rx unconditionally; 11=auto-enable Rx per /DCD pin	RW	

Receive Sync Register (RSR)

Register Address 0 b 10100

Receive Sync, SYN1, or 9th-16th bits of Ethernet address	Receive SYN0 or 1st-8th bits of address
15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
RSR15-8		Monosync	Receive Sync match character	WR	4: Monosync and Bisync Modes (pp.42-43)
		Bisync	second half of Receive sync match (SYN1)		
		802.3	match against last-received 8 bits of address		4: 802.3 (Ethernet) Mode (pp.45-46)
RSR7-0		Bisync	first half of Receive sync match (SYN0)	WR	4: Monosync and Bisync Modes (pp.42-43)
		H/SDLC, (CMR7-4) <>xx00, 802.3	match against first-received 8 bits of address		4: HDLC/SDLC Mode (pp.46-49) 4: 802.3 (Ethernet) Mode (pp.45-46)

Status Interrupt Control Register (SICR)

Register Address 0 b 01111

RxCDn IA	RxCUp IA	TxCDn IA	TxCUp IA	RxRDn IA	RxRUp IA	TxRDn IA	TxRUp IA	DCDDn IA	DCDUp IA	CTSDn IA	CTSUp IA	RCC Under IA	DPLL DSync IA	BRG1 IA	BRG0 IA
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
SICR15	RxCDn IA		1=set MISR15/interrupt on fall of /RxC	RW	3: The /RxC and /TxC Pins (p.31)
SICR14	RxCUp IA		1=set MISR15/interrupt on rise of /RxC	RW	
SICR13	TxCDn IA		1=set MISR13/interrupt on fall of /TxC	RW	
SICR12	TxCUp IA		1=set MISR13/interrupt on rise of /TxC	RW	
SICR11	RxRDn IA		1=set MISR11/interrupt on fall of /RxREQ	RW	3: /RxREQ and /TxREQ Pins (pp.31-32)
SICR10	RxRUp IA		1=set MISR11/interrupt on rise of /RxREQ	RW	
SICR9	TxRDn IA		1=set MISR9/interrupt on fall of /TxREQ	RW	
SICR8	TxRUp IA		1=set MISR9/interrupt on rise of /TxREQ	RW	
SICR7	DCDDn IA		1=set MISR7/interrupt on fall of /DCD	RW	3: The /DCD Pin (pp.29-30)
SICR6	DCDUp IA		1=set MISR7/interrupt on rise of /DCD	RW	
SICR5	CTSDn IA		1=set MISR5/interrupt on fall of /CTS	RW	3: The /CTS Pin (pp.30-31)
SICR4	CTSUp IA		1=set MISR5/interrupt on rise of /CTS	RW	
SICR3	RCC Under IA	RCC used	1=interrupt on RCC underflow (Receive frame/message longer than max allowed)	RW	4: DMA Support Features: The RCC FIFO (p.60)
SICR2	DPLLDSync IA	Biphase	1=interrupt on DPLL sync loss	RW	3: More About the DPLL (pp.26-27); 6: Miscellaneous Int. Sources and IA Bits (pp.89-90)
SICR1	BRG1 IA		1=interrupt on BRG1 zero	RW	3: Tx and Rx Clocking: The Baud Rate Generators (pp.21-23)
SICR0	BRG0 IA		1=interrupt on BRG0 zero	RW	

Test Mode Control Register (TMCR)

Register Address 0 b 00111

Reserved (0)	Test Register Address
15 14 13 12 11 10 9 8 7 6 5	4 3 2 1 0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
TMCR4-0			address of test register to read and write in TMDR	?	*USC Family Test Modes* (forthcoming separate document)

Test Mode Data Register (TMDR)

Register Address 0 b 00110

Test Register selected by TMCR4-0
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
TMCR15-0			test register selected by TMCR4-0	varies	*USC Family Test Modes* (forthcoming separate document)

Time Constant 0 Register (TC0R)

Register Address 0 b 10111

divisor for (or current count in) Baud Rate Generator 0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
TC0R15-0		write, or read w/ TC0RSel (RICR0)=0	divisor/starting value for BRG0: 0=input=output; 1=divide by 2; n=divide by n+1	RW	4: DMA Support Features: The Character Counters (pp.57-59)
		read w/ TC0RSel (RICR0)=1	value of BRG0 counter last time TC0RSel:=1	RO	

Time Constant 1 Register (TC1R)

Register Address 0 b 11111

divisor for (or current count in) Baud Rate Generator 1															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
TC1R15-0		write, or read w/ TC1RSel (TICR0)=0	divisor/starting value for BRG1: 0=input=output; 1=divide by 2; n=divide by n+1	RW	4: DMA Support Features: The Character Counters (pp.57-59)
		read w/ TC1RSel (TICR0)=1	value of BRG1 counter last time TC1RSel:=1	RO	

Transmit Character Count Register (TCCR)

Register Address 0 b 11110

current value of Transmit Character Counter															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
TCCR15-0			0=TCC disabled; else number of bytes (left) to send in current/next Transmit frame/message	RO	4: DMA Support Features: The Character Counters (pp.57-59)

Transmit Command/Status Register (TCSR)

Register Address 0 b 11010

TCmd				Rsrvd	TxIdle			Pre Sent	Idle Sent	Abort Sent	EOF/ EOM Sent	CRC Sent	All Sent	Tx Under	Tx Empty
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
TCSR15-12	TCmd		0000=no operation; 0001=reserved	WO	4: Commands (pp.63-66)
		Sync	0010=Clear Tx CRC Generator		
			0011, 0100=reserved 0101=Select TICRHi=TxFIFO Status 0110=Select TICRHi=/INT Level 0111=Select TICRHi=/TxREQ Level		
		TICR2=1	1000=Send Frame/Message		
		H/SDLC	1001=Send Abort		
			101x=reserved		
		T.Bisync	1100=Enable DLE Insertion 1101=Disable DLE Insertion		
		Sync	1110=Clear EOF/EOM 1111=Set EOF/EOM		
TCSR10-8	TxIdle		selects the Transmit idle line condition: 000=the default for TxMode (sync/Flag/Mark) 001=alternating zeroes and ones 010=continuous zeroes 011=continuous ones 100=reserved 101=alternating Mark and Space 110=continuous Space (TxD low) 111=continuous Mark (TxD high)	RW	4: Between Messages, Frames, or Characters (pp.68-70)
TCSR7	PreSent	Sync	1=Transmitter has finished sending Preamble	R,W1U	4: Status Reporting: Detailed Status in the TCSR (p.54)
TCSR6	IdleSent		1=Transmitter has sent Idle condition	R,W1U	
TCSR5	AbortSent	H/SDLC	1=Transmitter has sent Abort	R,W1U	
TCSR4	EOF/EOM Sent	Sync	1=Transmitter has sent End of Frame/End of Message	R,W1U	
TCSR3	CRCSent	Sync	1=Transmitter has sent a CRC code	R,W1U	
TCSR2	AllSent	Async	1=last bit has gone out onto TxD	RO	
TCSR1	TxUnder		1=Transmitter has Underflowed	R,W1U	
TCSR0	TxEmpty		1=TxFIFO is empty	RO	

Transmit Count Limit Register (TCLR)

Register Address 0 b 11101

starting value for Transmit Character Counter															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
TCLR15-0			starting value for TCC: 0=disable TCC; else length of next frame/message	RW	4: DMA Support Features: The Character Counters (pp.57-59)

Transmit Data Register (TDR)

Register Address 0 b 1x000 or 1 b xxxxx

Transmit character: write only using 16-bit operation	Transmit character: 8- or 16-bit write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
TDR15-8		16 bit bus	the "other" Transmit character in a 16-bit write (may be sent 1st or 2nd per "Select D15-8 First" or "Select D7-0 First" command in RTCmd [CCAR15-11])	WO	4: The Data Registers and the FIFOs (pp.66-68)
TDR7-0			Transmit character	WO	

Transmit Interrupt Control Register (TICR)

Register Address 0 b 11011

"TxFIFO Status" if last TCSR15-12 command 4-7 was 5 "Tx /INT Level" if last TCSR15-12 command 4-7 was 6 "/TxREQ level" if last TCSR15-12 command 4-7 was 7	Pre Sent IA	Idle Sent IA	Abort Sent IA	EOF/ EOM Sent IA	CRC Sent IA	Wait2 Send	Tx Under IA	TC1R Sel

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
TICR15-8		5 written to TCmd (TCSR15-12), or Reset, since 6 or 7 written there	the number of character/byte/octet entries currently empty in the TxFIFO	RO	4: The Data Registers and the FIFOs (pp.66-68)
TICR15-8		6 written to TCmd (TCSR15-12) since 5 or 7 written there	the number of empty character/byte/octet entries in the TxFIFO, above which to request a Transmit Data interrupt	RW	6: Transmit Data Interrupts (pp.88-89)
TICR15-8		7 written to TCmd (TCSR15-12) since 5 or 6 written there	the number of empty character/byte/octet entries in the TxFIFO, above which to request Transmit DMA transfer	RW	5: DMA Requests by the Receiver and Transmitter (pp.73-75)
TICR7	PreSent IA	Sync	1=arm interrupts on Preamble Sent (TCSR7)	RW	6: Transmit Status Interrupt Sources and IA Bits (pp.86-88)
TICR6	IdleSent IA		1=arm interrupts on IdleSent (TCSR6)	RW	
TICR5	AbortSentIA	H/SDLC	1=arm interrupts on AbortSent (TCSR5)	RW	
TICR4	EOF/EOM Sent IA	Sync	1=arm interrupts on EOF/EOM Sent (TCSR4)	RW	
TICR3	CRCSent IA	Sync	1=arm interrupts on CRCSent (TCSR3)	RW	
TICR2	Wait2Send	Sync	1=hold Transmitter from sending each frame/message until software issues "Send Message/Frame" command	RW	4: Synchronizing Frames/ Messages with Software Response (p.70)
TICR1	TxUnder IA		1=arm interrupts on TxUnder (TCSR1)	RW	6: Transmit Status Interrupt Sources and IA Bits (pp.86-88)
TICR0	TC1R Sel		0=select Time Constant value for reading TC1R 1=capture current count for reading TC1R	RW	3: Tx and Rx Clocking: The Baud Rate Generators (pp.21-23)

Transmit Mode Register (TMR)

Register Address 0 b 11001

TxEncode			TxCRCType		TxCRC Start	TxCRC Enab	TxCRC atEnd	TxParType		TxPar Enab	TxLength			TxEnable	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
TMR15-13	TxEncode		000=don't encode TxD ("NRZ"); 001=invert polarity of TxD ("NRZB"); 010=encode TxD NRZI-Mark; 011=encode TxD NRZI-Space; 100=encode TxD Biphase-Mark (FM1); 101=encode TxD Biphase-Space (FM0); 110=encode TxD Biphase-Level (Manchester); 111=encode TxD Differential Biphase-Level	RW	3: Data Formats and Encoding (pp.25-26)
TMR12-11	TxCRCType	Sync	00=use 16-bit CRC-CCITT for Tx; 01=use CRC-16 for Tx; 10=use 32-bit Ethernet CRC for Tx	RW	4: Cyclic Redundancy Checking (pp.50-51)
TMR10	TxCRCStart	Sync	0=start Transmit CRC generator as all-zeroes; 1=all ones	RW	
TMR9	TxCRCEnab	Sync	1=include Transmit characters in CRC	RW	
TMR8	TxCRCatEnd	Sync	1=send accumulated CRC at EOF/EOM	RW	
TMR7-6	TxParType		00=Transmit Parity Even; 01=Odd; 10=Zero (Space); 11=One (Mark)	RW	4: Parity Checking (p.52)
TMR5	TxParEnab		1=accumulate & send Parity bits	RW	
TMR4-2	TxLength		000=send eight bit characters; 001-111=send 1-7 bit characters	RW	4: The Mode Registers: Character Length (pp.45-46)
TMR1-0	TxEnable		00=disable Transmitter (immediately); 01=disable Tx at end of message/frame/char; 10=enable Tx unconditionally; 11=auto-enable Tx per /CTS pin	RW	4: The Mode Registers: Enabling and Disabling (p.37)

Transmit Sync Register (TSR)

Register Address 0 b 11100

Transmit SYN1								Transmit Sync or SYN0							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit(s)	*Field/Bit Name*	*Conditions /Context*	*Description*	*RW Status*	*Ref Chapter: Section*
RSR15-8		Bisync	second half of Transmit sync (SYN1)	WR	4: Monosync and Bisync Modes (pp.42-43) 4: Slaved Monosync Mode (p.45)
RSR7-0		Monosync, Slaved Monosync	Transmit Sync character	WR	
		Bisync	first half of Transmit sync (SYN0)		

Appendix: Name Changes

This is for the reader of previous USC documentation. It summarizes the changes in the names of registers and commands that were made in this document, with a few words about why they were changed.

Transmit Status Blocks-->Transmit Control Blocks

The names of registers and other USC features, in past documentation, maintained the distinction between "status" info as flowing from the USC to the host, and "control" information as flowing from the host to the USC pretty strictly -- all except this one.

Interrupt Enable (for individual sources) --> Interrupt Arm

There was no distinction between the enabling of a whole interrupt type and the enabling of an individual source within a type, and it seemed important to distinguish between these, so we kept the former as "enabling" and called the latter "arming" instead. Vague memories of early minicomputer terminology say the same terms were used.

Commands

Reload RCC / TCC --> Load RCC/TCC

It wasn't clear why RCC and TCC were "reloaded" while TC0 and TC1 were just "loaded".

Select Straight/Swapped Memory Data --> Select D15-8/D7-0 First

"Straight" means whichever way your microprocessor wants it, while "swapped" is the way the other guys' part works...

Preset CRC --> Clear Tx/Rx CRC Generator

More descriptive of the function: "preset" seemed to carry the possibility that you might be able to load in any arbitrary starting value...

Bit/Field Names

There weren't really bit and field names in the old Technical Manual -- they were more like text titles. But for those bits and fields that had fairly short titles, the names in this manual may or may not be the same. One change of note is that RCSR4 has been changed from "CV/EOF/EOM" to "RxBound", after it was noted that the bit has a fourth use: in Nine-Bit mode it flags address bytes. ("CV/EOF/EOM/Addr" seemed a little long...)

Another such change is that CCSR14 is now called RCCF Avail rather than RCC Valid. (It's perfectly *valid* for the RCC FIFO to be empty, in which case there's nothing *available* to be read from it.)

The bit and field names in this book are similar to, but not identical with, those in the Electronic Programmer's Manual.

Index

In the following index:

Bold page numbers identify the definition or main explanation of a term.

Italic page numbers identify a Figure that illustrates the term.

Bold Italic page numbers identify a section about the term, that includes both text and pictorial information.

ZILOG DOMESTIC SALES OFFICES AND TECHNICAL CENTERS

CALIFORNIA
Agoura 818-707-2160
Campbell 408-370-8120
Tustin 714-838-7800

COLORADO
Boulder 303-494-2905

FLORIDA
Largo 813-585-2533

GEORGIA
Norcross 404-448-9370

ILLINOIS
Schaumburg 708-517-8080

MINNESOTA
Minneapolis 612-944-0737

NEW HAMPSHIRE
Nashua 603-888-8590

NORTH CAROLINA
Raleigh 919-790-7706

OHIO
Independence 216-447-1480

PENNSYLVANIA
Ambler 215-653-0230

TEXAS
Dallas 214-987-9987

WASHINGTON
Seattle 206-523-3591

INTERNATIONAL SALES OFFICES

CANADA
Toronto 416-673-0634

GERMANY
Munich 49-89-672-045
Sömmerda 37-626-23906

JAPAN
Tokyo 81-3-3587-0528

HONG KONG
Kowloon 852-7238979

KOREA
Seoul 82-2-552-5401

SINGAPORE
Singapore 65-2357155

TAIWAN
Taipei 886-2-741-3125

UNITED KINGDOM
Maidenhead 44-628-392-00

Zilog, Inc. 210 East Hacienda Ave.
Campbell, CA 95008-6600
Telephone (408) 370-8000
Telex 910-338-7621
FAX 408 370-8056