ZiLOG

Z16C32 IUSC™ Integrated Universal Serial Controller

UM014001-1002

User's Manual

Q1/95 DC 8350-00

Z16C32 User's Manual

Z16C32 IUSC™ USER'S MANUAL

Thank you for your interest in Zilog's high-speed Integrated Universal Serial Controller. To aid the designer, the following support material is available when designing a High Performance Serial Communication application based on Zilog's IUSC.

ABOUT THIS MANUAL

This 16C32 IUSC Integrated Universal Serial Controller User's Manual references the 16C32 SL1660 IUSC product (noted by referring to the topmarking on the device). The manual is also applicable to previous revisions of the 16C32 products. Any specific differences that will affect the use of the 16C32 product will be duly noted. These notices occur primarily in Chapters 5,6 and 8 of this User's Manual.

Z16C32 User's Manual

Zilog's IUSC User's Manual is a comprehensive breakdown of the functions and features of the IUSC which will aid in the development of your application. A good place to start is the *Overview* section at the beginning of the manual, which provides a short description of each feature and chapter. Then any chapter can be reviewed in more detail as necessary. This User's Manual provides in-depth descriptions of all functions and features of the IUSC as well as supporting block diagrams, timing diagrams, and sample applications.

ADDITIONAL DOCUMENTATION AND SUPPORT PRODUCT INFORMATION

Z16C32 Product Specification

The IUSC Product Specification provides descriptions of features, block diagrams, pin assignments, pin descriptions, register bit functions, and AC and DC specifications. This specification can be found in the High Speed Serial Communication Controllers Databook, DC-8314-01, which also includes a product specification on the Z16C30 USC as well as related Application Notes, support products, and a list of third party support vendors.

Application Notes

The following Application Notes are useful in demonstrating how the IUSC can be used in different applications.

Design a Serial Board to Handle Multiple Protocols
This Application Note details an approach to handling multiple serial communication protocols using Zilog's Z16C32 USC. This document is included in the High Speed SCC Databook, DC-8314-01 mentioned above.

Datacommunications with the IUSC™ Time Slot Assigner
This Application Note explains the functions of the TSA (Time Slot Assigner) section of Zilog's IUSC. This document is also included in the High Speed SCC Databook, DC-8314-01 mentioned above.

The Zilog Datacom Family with the 80186 CPU
This Application Note, DC-2560-03, explains and illustrates how Zilog's datacom family interfaces and communicates with the 80186 on an evaluation board.

Demonstration/Evaluation Boards

By selecting the board that most closely resembles your desired application, you may be able to use parts of the design in your implementation. These boards can be used as software platforms while the application hardware is being developed.

Z16C3200ZCO - Zilog's ZNW2000 Developer's Kit, DC-8315-00, includes an add-in board for AT/ISA systems that can be used for a wide variety of serial communications applications including Wide Area Network (WAN) access for PC-based LAN Server/Routers at speeds up to and above T1/E1 (1.544/2.048 Mbps) rates.

The kit contains software, hardware, and documentation that exemplify high speed serial designs: a ZNW2000 board, User's Manual (with engineering specification, test software documentation, schematic diagrams, bill of materials, PAL equations, and timing analysis), and disk (with fully documented source code, schematics, and PAL equations).

Demonstration/Evaluation Boards (Continued)

The ZNW2000 Developer's Kit can be used with Novell's Netware, and supports Frame Relay, Point-to-Point (PPP), and X.25 and has serial interfaces for V.35, RS-232, X.21, RS-449/422 and RS-449/423.

Z8018600ZCO - This kit contains an assembled circuit board, software, and documentation to support the evaluation and development of code for Zilog's Z16C30 USC, Z16C32 IUSC, Z85C30 SCC, Z85230 ESCC, and Z16C35 ISCC. The board illustrates how the datacom family interfaces and communicates with the 80186 CPU. A board-resident monitor program allows code to be downloaded and executed. A specification of this product is included in the High Speed Serial Communication Controllers Databook, DC-8314-01 mentioned above.

Electronics Programmer's Manual

The EPM is a software tool that provides on-line documentation on Zilog's USC Universal Serial Controller (Z16C30) and IUSC Integrated Universal Serial Controller (Z16C32) register sets and device operation. Its code generation features makes it a most valuable tool for the programmer. The EPM helps set the registers to ensure that the device operates with the specified settings. Use the "Function" option to develop .h files and to create an initialization sequence for your software. Once you have selected the field values as a series of C function calls, or as an assembler table, you can include this output in any software that utilizes the device. A specification of this product is included in Zilog's Master Selection Guide, DC-5634-01.

Third Party Software Support

These third party software vendor/contractors can supply useful drivers that have already been written for the IUSC. These contractors' experience writing software for the IUSC can also be used to help develop new code in a timely manner. A list of additional Third Party support vendors is is included in Zilog's Master Selection Guide, DC-5634-01.

Software Support

Forward Technology	**(516) 496-9033**
Software for data and telecommunications Extensive experience with the Z16C32 and Z85230	
GCOM	**(217) 337-4471**
Drivers for many common protocols including Frame Relay, X.25, LAPD, SDLC/HDLC, and ADCCP	
Novell	**1-800-NETWARE**
Software Multi-Protocol router (MPR 2.1) driver for the NW2000 Frame Relay.	
Telenetworks	**(707) 778-8737**
Software for data and telecommunications Specialists in ISDN protocols	
Trillium Digital Systems	**(310) 479-0500**
Protocol stacks for common WAN applications such as X.25, Frame Relay and ISDN	

Hardware Support

Computer Modules	**(408) 496-1881**

Question and Answer File

During the design cycle, this extensive list of commonly asked questions can provide helpful hints and save time in solving problems or questions that may arise. This is an extensive list of commonly asked questions and answers for engineers while designing a USC or IUSC based system. This list can be obtained from your local Zilog Sales office.

Zilog Bulletin Board Service

Zilog's electronic BBS can be used to get product updates on the products and download sample code from the factory. Zilog maintains an electronic BBS that is on-line 24 hours a day. The BBS is at 408-370-8024 (2400, 8, N, 1)

Z16C32 IUSC™ User's Manual

Table of Contents

CHAPTER TITLE AND SUBSECTIONS **PAGE**

CHAPTER TITLE AND SUBSECTIONS **PAGE**

CHAPTER TITLE AND SUBSECTIONS **PAGE**

CHAPTER TITLE AND SUBSECTIONS **PAGE**

CHAPTER TITLE AND SUBSECTIONS **PAGE**

Z16C32 IUSC™ User's Manual

Table of Contents

Figure Titles **Page**

FIGURE TITLES **PAGE**

Figure Titles **Page**

Z16C32 IUSC™ User's Manual

Table of Contents

Table Titles **Page**

CHAPTER 1
Z16C32 IUSC™ OVERVIEW

1.1 FEATURES

- Full-Duplex Multi-Protocol Serial Controller
- Two Multi-Mode DMA Channels with Peak Transfer Rates Up to 13.33 Mbytes/sec
- Serial Data Rates to 20 Mbits/sec
- Serial Modes Include Asynchronous, Synchronous, SDLC, HDLC, Ethernet, and Nine-Bit
- Two Baud Rate Generators
- Digital Phase Locked Loop for Clock Recovery
- Receive and Transmit Time Slot Assigners for ISDN and Fractional T1 Applications
- Ten General-Purpose I/O Lines Plus Carrier Detect, Clear to Send, and Two Clock I/Os
- Transmit and Receive Frame-Length Counters, Independent of the DMA Facility
- HDLC/SDLC Features Include 8-Bit Address Checking, Loop Mode, 16/32-Bit CRC, Programmable Idle State, Auto Preamble Option or Programmable Minimum Flag Count Between Frames, Real-Time or In-Data-Stream Abort Notification
- Sync Features Include 2-Bit to 16-Bit Sync Pattern, Sync Strip Option, 16/32-Bit CRC, Programmable Idle State, Auto Preamble Option, X.21 XMIT/RCV Slaving
- Async Features Include False-Start Filtering, Stop Bit Length Programmable by 1/16-Bit Steps, Parity Generation/Checking, Break Generation/Detection
- 32-Character Transmit and Receive FIFOs Between the Serial Controller and the DMA Channels
- Improved Bus/Serial Interlocks Prevent Extra Received DMA Characters and Misreporting of FIFO Fill Levels
- DMA Modes Include Single Buffer, Pipelined, Array Chained, and Linked-List
- 16-Bit and 32-Bit Addressing, 8-Bit or 16-Bit Data
- Received Frames can be Placed in Separate Memory Buffers or Stored Successively Without Regard for Buffer Boundaries
- Received-Frame Status can be Stored with DMA Control Info or After the End of Each Frame
- Transmit-Frame Control Info can Come from the DMA Control Structure or Before the Start of Each Frame
- Buffer Ring Wraparound Protection
- Programmable Throttling of DMA Bus Occupancy
- Flexible Adaptation to Various System Buses
- Flexible Interrupt and Bus-Arbitration Modes
- Interrupt and Bus-Acknowledge Daisy Chains
- Socket-and Software-Compatible with Z16C31 IUSC
- High-Speed, Low Power CMOS Technology
- 68-Pin PLCC

1.2 INTRODUCTION

The Z16C32 Integrated Universal Serial Controller (IUSC™) is the latest member of Zilog's large and popular family of multi-protocol serial controllers, which ranges from the original Z80-SIO through the industry standard SCC and the more recent ESCC, ISCC, and USC. Compared to the SCC family and most competing devices, the USC family features more serial protocols, a 16-data bus, higher data rates, larger FIFOs, better support for DMA operation, and more convenient software handling. The IUSC adds a Direct Memory Access (DMA) facility that has correspondingly powerful capabilities for transferring data to and from data buffers in memory.

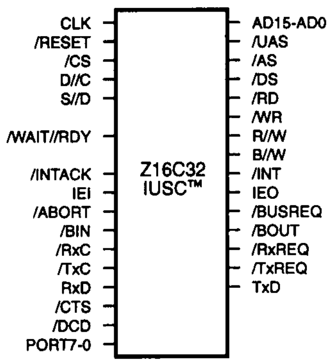


Figure 1-1. IUSC Logic Symbol

1.3 PACKAGING

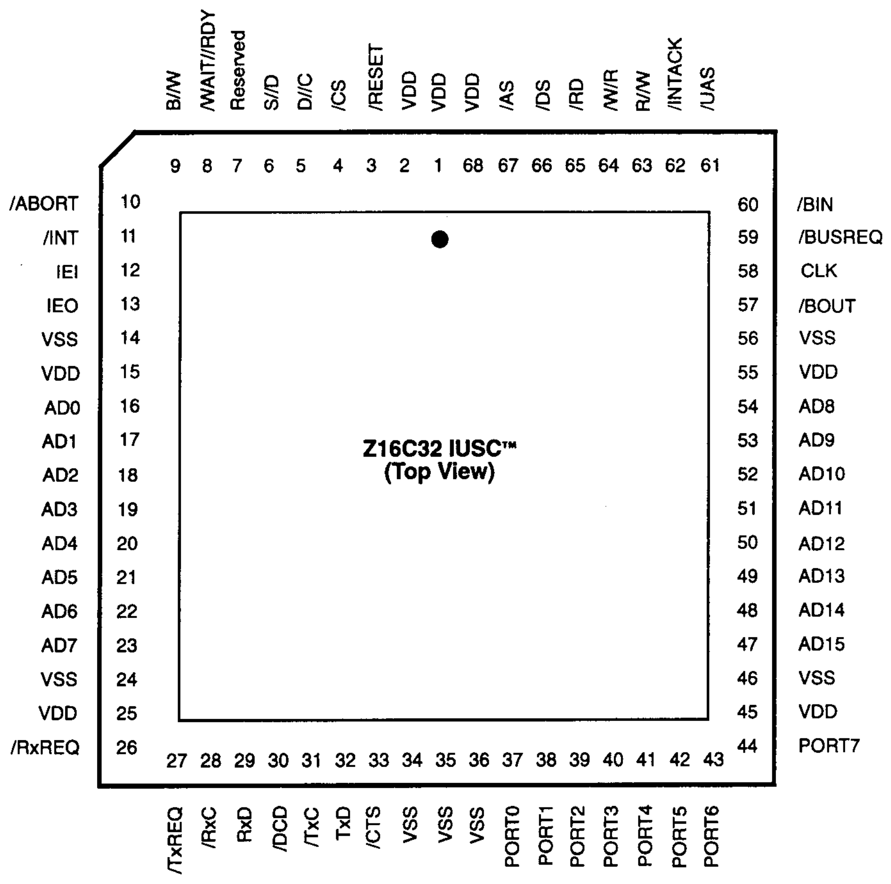


Figure 1-2. IUSC 68-Pin PLCC Pinout

1.4 GENERAL OVERVIEW

The following descriptions should be helpful in initial evaluation of the IUSC. Tables 1-1 through 1-6 give a brief overview of Chapters 2 and 4-8.

1.4.1 Bus Interfacing

Chapter 2 describes interfacing the IUSC to a processor or backplane bus. The IUSC includes several flexible interfacing options as described in Table 1-1. Many of these options are controlled by the Bus Configuration Register (BCR), which is implicitly the destination of the first write to the IUSC after a Reset, and is then no longer accessible to software.

1.4.2 Serial Interfacing

Chapter 4 covers Serial Interfacing, the "other side" of hardware design from Bus Interfacing. Table 1-2 summarizes the Serial Interfacing features of the IUSC, which include Clock Selection, Baud Rate Generation, serial data Encoding and Decoding, a Digital Phase Locked Loop for reconstructing clocking from received data, "modem control" and general-purpose Port pins, and Time Slot Assigner logic for ISDN/T1 applications.

1.4.3 Serial Modes and Protocols

Chapter 5 covers how to program the Transmitter and Receiver to handle many different protocols and applications. This Chapter focuses on software aspects of using the IUSC while Chapter 4 is more hardware-oriented. Tables 1-3 and 1-4 show the major subjects that you can find in Chapter 5.

1.4.4 DMA Operation

Chapter 6 covers both hardware and software aspects of using the IUSC's integrated DMA channels. These channels can be used in any of four basic modes with a number of options, as outlined in Table 1-5.

1.4.5 Interrupts

While Chapters 4-6 mention which conditions and events can be enabled/armed to interrupt the processor, Chapter 7 combines all aspects of the IUSC's extensive interrupt capabilities, including interrupt acknowledge cycles, vectors, and use of Interrupt Under Service bits to implement nested interrupts. Table 1-6 summarizes the subject.

1.4.6 Software Summary

Chapter 8 contains only a small amount of new material. The bulk of the Chapter is the Register Reference tables that summarize the use and function of each bit and field in each register in the IUSC.

Table 1-1. Bus Interfacing Features of the IUSC (Chapter 2)

Multiplexed or Separate Address and Data Bus(es)	can be selected for processor access to IUSC registers. Addresses and Data are always multiplexed when the IUSC's DMA channels fetch or store serial data in memory. External components can be used to demultiplex DMA addresses and data, if the host bus requires this.
Read/Write Control Signals	Separate Read and Write strobes, or Data strobe and Direction control can be used. Only one set of signals should be connected for processor access. Both are provided during DMA operation.
8-Bit or 16-Bit Data Bus	DMA efficiency and bandwidth are doubled by using a 16-bit bus, and software size and tediousness is improved as well. With an 8-bit data bus and non-multiplexed Address and Data, the bus pins that would otherwise be unused can be used for register addressing from the processor. When the IUSC is bus master it always drives addresses on all 16 AD pins.
Ready, Wait, or Acknowledge Handshaking	can be selected for both processor cycles and DMA operation. If Wait signaling is selected, the IUSC drives Wait for interrupt acknowledge cycles but not for register accesses — its 60 ns register access time is fast enough for no-Wait operation in almost all applications. If Acknowledge signaling is selected, the part drives Acknowledge for both interrupt acknowledge cycles and register accesses. The IUSC responds to Acknowledge or Wait signaling when it is the bus master.
Interrupt Acknowledge Cycles	Status line, single-pulse, or double-pulse cycles can be selected. The IUSC can also be used on buses that do not include Interrupt Acknowledge cycles.
Direct or Indirect Register Addressing	The board designer can conserve the address space occupied by the IUSC by requiring software to write register addresses into the IUSC, or can maximize software efficiency by presenting register addresses directly. On a non-multiplexed 16-bit data bus, the latter choice requires external components/logic to multiplex the address onto the AD pins.
Registers	There are 32 16-bit registers in the Serial Controller section, including four selectable subregisters in the MS byte of two of them. The DMA section includes eight "shareable" 16-bit registers that apply to both channels, eight 16-bit registers that are specific to the Transmit channel, and eight that are specific to the Receive channel.
Big-Endian or Little-Endian Byte Ordering	Motorola or Intel style addressing can be selected for serial data and for addresses in DMA buffer descriptors. Byte addressing within the IUSC's 16-bit registers is inherently Little-Endian/Intel style, as is addressing within the register pairs that hold 32-bit DMA addresses.
Context Indication Option	During Bus Master operation the IUSC can output signals to differentiate Transmit from Receive DMA channel operation, and serial data accesses from accesses to buffer descriptors.
Automatic One-Wait-State Option	For Bus Master operation, this option allows the IUSC to be used with slower memories without requiring external logic to generate Wait, Ready, or Acknowledge signaling.

Table 1-2. Serial Interfacing Features of the IUSC (Chapter 4)

Clock Selection	Clocking for the Transmitter and Receiver can come from any of the /RxC, /TxC, PORT0, or PORT1 pins, and can be used directly or can be divided by 4, 8, 16, or 32 by Counters 0 and 1, and/or by any value from 1 to 65,536 by Baud Rate Generators 0 and 1. Or, clocking can come from the Digital Phase Locked Loop (DPLL) module, which tracks transitions on the RxD pin.
Clock Output	Clocking can also be driven out on the /TxC and/or /RxC pin(s) for use by on-board logic, a modem or other interface.
CTR0, CTR1	These two 5-bit free-running counters can each divide /RxC, /TxC, PORT0, or PORT1 by 4, 8, 16, or 32. They can provide the Transmit or Receive bit clocks directly, or can act as "prescalers" for the Baud Rate Generators.
Baud Rate Generators	BRG0 and BRG1 are 16-bit counters, each of which can divide /RxC, /TxC, PORT0, PORT1, or the output of CTR0 or CTR1 by any value from 1 to 65,536. They can source the Transmit or Receive bit clocks, act as the reference clock for the DPLL, or can be used as timers on either a polled or interrupt-driven basis. They can be stopped and started by software, and can run continuously or stop when they reach zero. Their period (time constant) values can be reprogrammed dynamically, effective immediately or when the BRG counts down to zero.
Digital Phase Locked Loop	The DPLL can divide /RxC, /TxC, or the output of BRG0 or BRG1 by 8, 16, or 32, while resynchronizing to transitions on RxD, to recover a Receive clock from the Receive data signal. This can be done only when the received data stream includes enough transitions to keep the recovered clock synchronized to the data. NRZI-Space encoding of HDLC/SDLC frames, or Biphase (FM) encoding with any protocol, guarantees such data transitions.
Data Encoding	The IUSC can encode transmitted data and decode received data in NRZI-Mark, NRZI-Space, Biphase-Mark (FM1), Biphase-Space (FM0), Biphase-Level (Manchester), or Differential-Biphase-Level modes. These encodings are used in various applications to maintain synchronization between transmitting and receiving equipment.
Echoing and Looping	Received data can be repeated onto TxD, or transmitted data can be looped back to the Receiver for testing.
Modem Controls and Interrupts	Carrier Detect and Clear to Send inputs can auto-enable the Receiver and Transmitter, respectively. Rising and/or falling edges on these pins can cause interrupts, as can edges on the Transmit and Receive Clock pins (when they are not used for clocking), and/or the Transmit and Receive Request pins.
Port Pins	Eight "port" pins can be programmed as general-purpose inputs or outputs, or can carry a specific input or output signal for each pin. They differ from the pins mentioned in the above section in that they cannot cause interrupts. Thus they are more suitable for modem control outputs than for inputs.
Time Slot Assigners	These circuits allow integrated "gapping" or gating of a high-speed input clock, to restrict IUSC Transmit and Receive operation to only a selected number of "time slots" within a cyclic time-multiplexed frame as in ISDN and Fractional T1 applications. They can reduce the requirements for external components from a full framer to a simple Frame Sync detector.

Table 1-3. Serial Controller Features of the IUSC (Chapter 5)

Major Protocol Categories	Chapter 5 begins with a small tutorial on the differences between Asynchronous, Character-Oriented Synchronous, and Bit-Oriented Synchronous (Packet) protocols.
Asynchronous Protocols	In addition to classic Async, the IUSC can handle the following variations: ■ Isochronous (1X rather than 16-64X clock) ■ Nine-Bit (Address Wakeup – an extra bit signifies Address/Data)
Character-Oriented Synchronous Protocols	■ External Sync (Receive only: simple character assembly) ■ Monosync (1-character sync pattern, no hardware framing) ■ Bisync (2-character sync pattern, no hardware framing) ■ Transparent Bisync (Bisync + hardware support for Transparency) ■ Slaved Monosync (Xmit only; X.21 Tx character alignment to Rx) ■ IEEE 802.3 (Ethernet; requires external collision detect and backoff)
Bit-Oriented Synchronous Protocols	■ HDLC/SDLC ■ HDLC/SDLC Loop (RxD is repeated on TxD except when Xmit is enabled and triggered by a received Go Ahead/Abort sequence)
Character Length	Is programmable from 1-bit/character to: ■ 8 bits including Parity, if any, in synchronous modes ■ 8 bits plus Parity, if any, in Async mode ■ 8 bits plus Parity plus the Address/Data bit in Nine-Bit mode
CRC Generation/Checking	In synchronous modes, the IUSC will generate and check CRC-CCITT, CRC-16, or CRC-32 codes for each frame or message. For character-oriented modes other than 802.3, software can selectively control which characters are included in the CRC, for both transmitting and reception. For HDLC/SDLC and 802.3, CRC status can be stored in memory for each received frame.
Parity Checking	Asynchronous or Synchronous modes. Odd/Even/Mark/Space/None.
Transmit Status Reporting	Optional interrupt on: Preamble Sent, Idle Sent, Abort Sent, End of Frame/Message, CRC Sent, Underrun No interrupt: All Sent, Tx Empty
Receive Status Reporting	Optional Interrupt on: Exited Hunt, Idle Received, Break, Abort (immediate or synchronized with the RxFIFO), Rx Boundary (end of frame/message), Parity Error, Overrun No interrupt: Short Frame, Code Violation Type, CRC Error, Framing Error, Rx Character Available
Character Counters	These 16-bit counters decrement for each character received or fetched from memory for transmission. The Tx CC can control the length of Tx frames in synchronous modes using DMA. The Rx CC tracks the length of each Rx frame in synchronous modes using DMA, and optionally interrupts in case an Rx frame is too long.
RCC FIFO	Four-deep store for ending Rx Character Counter values for each frame. This can be read directly by software, as an alternative to storing these values in Receive Status Blocks.

Table 1-4. More Serial Controller Features of the IUSC (Chapter 4)

Transmit Control Blocks	The Tx DMA channel can fetch the Tx CC frame length and other control information for each frame/message, either before the frame in the memory buffer or from the Array/Linked List entry that describes the buffer.
Receive Status Blocks	The Rx DMA channel can store the Rx CC frame length residual and the frame status (including CRC status) for each frame/message, either after the frame in the memory buffer, or in the Array/Linked List entry that describes the buffer.
Serial Controller Commands	Software can write 36 different command codes to three different register fields to control the operation of the serial controller section of the IUSC. Commands can be divided into those that select a long-term configuration of the IUSC (like selecting which serial character in a 16-bit word comes first), those that make the part perform a time-sequenced action (like sending an Abort sequence), and those that change the state of the part immediately (like purging a FIFO).
Software Reset	Software can reset the serial controller section of the IUSC by writing a central register bit, similarly to a hardware-signaled Reset.
Rx and Tx FIFO Storage	32-character FIFOs stand between the Tx DMA channel and the Transmitter, and between the Receiver and the Rx DMA channel. They help pro-rate the timing overhead of exchanging bus control between the processor or central arbiter and the IUSC, over enough transfers to reduce the impact of this overhead. Fill level counters track how many characters are in each FIFO, and independently programmable threshold values determine when DMA operation will be triggered to fill or empty them, and/or when an interrupt will be requested.
Between Frames/Messages	In synchronous modes the Transmitter will do the following before the first data character of each frame or message, or after the last one: ■ optionally send a 8-64 bit Preamble for PLL synchronization or minimum inter-frame timing ■ send an "opening" Sync sequence or Flag ■ after the last character from memory, optionally send the CRC accumulated by the IUSC. Thus, when this option is not used, a CRC received with a frame can be sent back out without being regenerated. ■ send a "closing" Flag or Sync ■ send a selected "idle" pattern unless/until the next frame is ready to be sent
Waiting for Software Response	Software can select three optional interlocks between frames, to allow it to do real-time processing on a frame-by-frame basis.

Table 1-5. DMA Features of the IUSC (Chapter 6)

DMA Basics	The IUSC's integrated DMA channels handle addresses up to 32 bits wide, 16-bit buffer length fields allowing buffers up to 65,536 bytes long, and 16-bit or 8-bit data transfers. They can operate in the four major modes described below.
Single Buffer Mode	Software programs one buffer address and length into the DMA channel, which needs to be reprogrammed before it can transfer another buffer.
Pipelined Mode	Software starts the channel as in Single-Buffer mode, but while the channel is transferring a buffer, software can program the address and length of the next buffer into the channel. The channel needs to be restarted only when/if it comes to the end of a buffer, and software has not provided the address and length of another one.
Array Mode	Software programs into the DMA channel, the address of an Array in memory that contains the addresses and lengths of several buffers. The channel needs to be restarted only after it has read or written serial data in all of the buffers.
Linked List Mode	Like Array mode except that rather than a continuous Array of addresses and lengths, software provides a List, each entry of which includes the address and length of one memory buffer and the memory address of the next entry. Software can add new entries to the end of the list as the DMA channel is transferring previous buffers, and in this way can keep the Rx or Tx DMA channel going indefinitely.
Frame/Buffer Independence	A frame or message can span several buffers in memory, or one buffer can contain several frames/messages. The DMA section tracks the length of memory buffers while the serial section tracks frame lengths. An optional Early Termination feature makes a DMA channel finish with a buffer when the serial controller signals it that a frame is ending.
DMA Operation	The IUSC requests use of the bus when the RxFIFO or TxFIFO reaches a programmed level of fullness/emptiness, when the end of a frame or message is received, or when it needs to fetch the address and length of a new buffer in Array or Linked List mode.
DMA Status Reporting	Optional interrupt on: End of Array/List, End of Buffer, Hardware or Software Abort No interrupt for: Continue, Getting Link, Busy, Initializing.
DMA Commands	Allows software to Reset, Pause, or Abort channels, and to Start or Restart them in three different ways.
Bus Occupancy Throttling	An IUSC's use of the bus can be limited in terms of the number of transfers it will do per bus grant, or the number of clocks it will use the bus per grant.

Table 1-6. Interrupt Features of the IUSC (Chapter 7)

Interrupt Acknowledge Daisy Chaining	was one of Zilog's original contributions to microprocessor architecture. On the IUSC, its use (to determine which of several interrupting devices to service first) is optional, and performance is much improved compared to older devices.
External Interrupt Control	can be used instead of a daisy chain to implement interrupt priority schemes other than strict priority, such as "fairness," rotating, or first-come first-served.
Types of Interrupts	that can be selectively enabled or disabled include Receive Status, Receive Data, Transmit Status, Transmit Data, I/O Pin, miscellaneous, Tx DMA, and Rx DMA interrupts.
Receive Status Interrupt	sources that can be selectively armed or disarmed include Exited Hunt, Idle Received, Break, Abort (immediate and/or synchronized to received data), End of Frame/Message, Parity Error, and RxFIFO Overrun.
Receive Data Interrupt	can occur when the RxFIFO reaches a programmed level of fullness
Transmit Status Interrupt	sources that can be selectively armed or disarmed include Preamble Sent, Idle Sent, Abort Sent, End of Frame/Message Sent, CRC Sent, and Tx Underrun.
Transmit Data Interrupt	can occur when the TxFIFO reaches a programmed level of emptiness.
I/O Pin Interrupt	The interrupts that can be selectively armed or disarmed include rising and/or falling edges on the /DCD, /CTS, /RxREQ, /TxREQ, /RxC, and /TxC pins.
Miscellaneous Interrupt	sources that can be selectively armed or disarmed include Rx Character Counter Underflow, DPLL Sync Loss, Baud Rate Generator0 zero, and BRG1=0.
DMA Interrupt	sources that can be selectively armed or disarmed in each channel include End of Array/List, End of Buffer, Software Abort, and Hardware Abort.
Nested Interrupts	are fully supported in that the IUSC includes an Interrupt Pending and Interrupt Under Service bit for each type of interrupt.
Interrupt Acknowledge Cycles	The IUSC is compatible with a wide variety of processors in that the signal that identifies an acknowledge cycle can be sampled like an address bit, or can carry a single or double pulse similar to a read or write strobe.
Interrupt Vectors	The IUSC can include identification of the highest priority requesting type of interrupt in the vector that it returns during an interrupt acknowledge cycle.
Non-Acknowledging Buses	Software can simulate the effects of interrupt acknowledge cycles, so that the IUSC can be used on buses that do not provide such cycles, like the ISA (AT) bus.

1.5 DEVICE STRUCTURE

Figure 1-3 shows the basic structure of the IUSC. The Bus Interface module stands between the external bus pins and an on-chip 16-bit data bus that interconnects the Receiver, Transmitter, Receive and Transmit FIFO storage, Receive and Transmit DMA channels, Clock Generator, Interrupt logic, and I/O Port functions.

1.5.1 The Transmit Data Path

The host processor or the on-chip Transmit DMA channel can write transmit data into the Transmit First-In, First-Out (TxFIFO) memory. At any time, the TxFIFO can be empty or can contain from 1 to 32 characters to be transmitted. Characters written into the TxFIFO become available to the Transmitter in the order in which they were written.

While the host processor can itself write data into the TxFIFO, it is more efficient to use the Transmit DMA channel to fetch the data. Software can set up the Transmit DMA channel to operate in any of four major modes. In Single-Buffer mode, the channel transfers one block of consecutive bytes from host memory given a programmable location and length, delivering the data to the TxFIFO, and then notifies the host processor and stops. Software has to reprogram the channel before it can transfer another block, but in many applications there is time to do this because the TxFIFO holds 32 characters.

In Pipelined mode, there are two sets of buffer address and length registers: software can program one set while the DMA channel is using the other set. When the channel

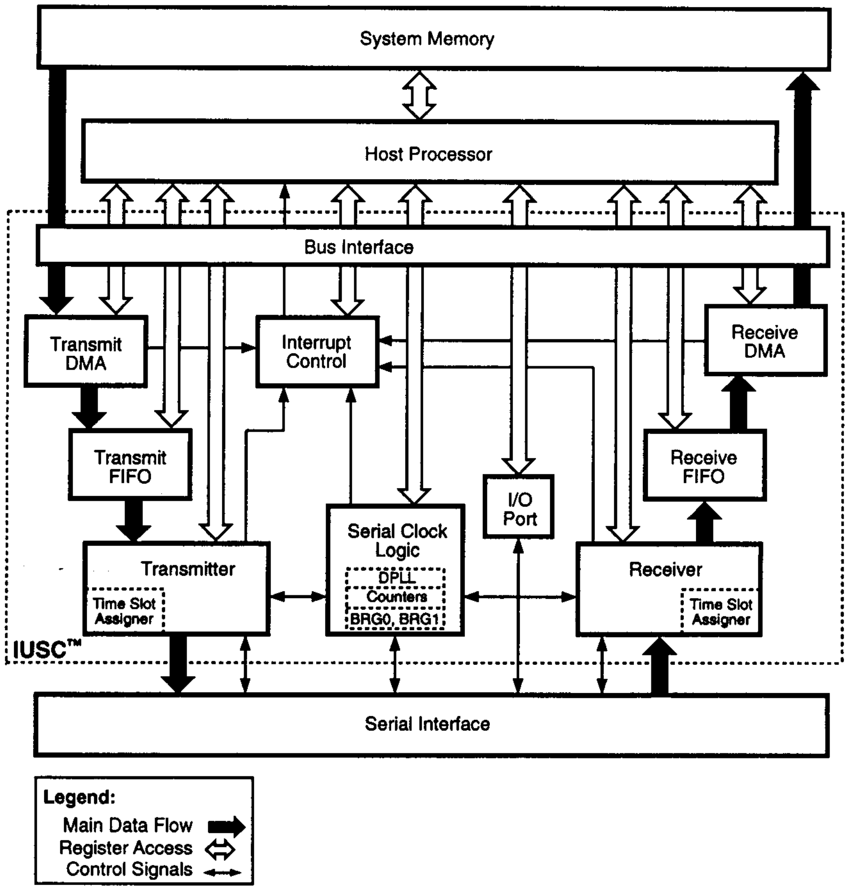


Figure 1-3. IUSC Block Diagram

finishes transferring one block, it notifies the host processor. If the host has set up the other register set, the channel proceeds to start transferring data from the next buffer.

In Array mode, the host processor programs the Tx DMA channel with the address of a table containing the addresses and lengths of the memory buffers. This table can also contain control information for each frame. When the channel finishes transferring the data from one memory buffer to the TxFIFO, it fetches the next buffer address and length from the table and begins to transfer the data from that buffer.

In Linked List mode, the host programs the channel with the address of the start of a linked list of buffer addresses and lengths, in which each entry also includes the address of the next entry. These entries can also contain control information for each frame. Channel operation is similar to operation in array-chained mode, but includes the extra steps of fetching the link addresses.

The host can program the Transmitter to trigger the 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 and release control of the host bus more often.

Finally, the Transmitter takes characters from the Transmit FIFO and converts them to serial data on the TxD pin. While this function is conceptually simple, the Transmitter may do any of the following in addition to parallel-serial conversion: start, stop, and/or parity bit generation, calculating and sending CRCs, automatic generation of opening and closing Sync or Flag characters, 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. The Z16C32 can also send a programmable minimum number of Flags between HDLC/SDLC frames.

For ISDN and Fractional T1 applications the Transmitter section includes Time Slot Assigner logic that can be used to enable the Transmitter only periodically and for specific bytes within a multiple-sourced, cyclically time-multiplexed data stream.

1.5.2 The Receive Data Path

In general, the functions of the Receiver section are the inverse of those of the Transmitter: it monitors the serial data on the RxD pin, recognizes its organization according to the serial mode selected by the software, and converts the data to parallel characters that it places in the Receive FIFO. Again, there is more to the process than just serial-parallel conversion. Depending on the serial mode the Receiver may have to detect and synchronize start bits, check parity and stop bits, calculate and check CRCs, detect Sync/Flag, Abort and/or Idle sequences, recognize control characters including transparency considerations, decode the serial data and extract clocking using any of several encoding schemes, and/or enable and disable reception based on the DCD input pin. The Receiver's checking functions generate several status bits associated with each character, that accompany the characters through the Receive FIFO. The Z16C32 can notify software of received HDLC/SDLC Abort sequences in real time and/or in the received data stream.

The Receiver section also includes a Time Slot Assigner that can be used to enable reception only for specific bytes within a multiple-destination, cyclically time-multiplexed data stream like an ISDN or Fractional T1 link.

The Receive FIFO can hold up to 32 characters and their associated status bits. As the receiver writes entries into their FIFOs, they become available to either the host processor or the Receive DMA channel in the order in which they were received. Similarly to the transmit side, the Receive FIFO includes detection logic for various degrees of "fullness." Separate thresholds control when the Receive DMA channel starts refilling the FIFO, and at which the IUSC requests an interrupt.

While the host processor can access data from the Receive FIFOs, it is more efficient to use the Receive DMA channel to transfer the data directly into buffer areas in memory. As on the transmit side, software can program the Receive DMA channel to operate in Single-Buffer mode, Pipelined mode, Array mode, or Linked List mode. The Z16C32 can store the status and length of each frame after the last character of each frame, or, in Array and Linked List modes, in the DMA control structure.

1.5.3 Clocking

The Serial Clocking Logic section 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 more external clock(s) for each channel. An on-chip Digital Phase Locked Loop (DPLL) circuit can recover clocking from encoded data on RxD.

1.5.4 Interrupts

The Interrupt Control section gathers the various "request" lines from the Transmitter, Receiver, and the DMA channels, 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, Transmitter, and DMA channels.

1.5.5 I/O Port

The I/O port section provides eight pins that can be used for modem control lines or any other purpose. Each pin can be individually controlled as an input or output, and most of them can optionally be used for a specific/dedicated input or output signal.

1.6 ABOUT THIS DOCUMENT

This manual provides a staged and gradual introduction to the IUSC. The manual is structured according to the IUSC'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 8 reviews the general programming model and includes a concise description of each register bit and field for quick reference.

Refer to the IUSC Product Specification for actual timing parameters and electrical specifications.

CHAPTER 2
Z16C32 IUSC™ BUS INTERFACING

2.1 INTRODUCTION

The IUSC can be used in systems with various microprocessor and 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 and DMA controllers. This chapter describes how to use these facilities to interface the IUSC to a variety of host microprocessors and buses.

2.2 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. As a DMA bus master the IUSC always operates in the latter (multiplexed) fashion. If the host bus does not multiplex addresses and data, addresses and data from the IUSC can be easily demultiplexed as described later. If it does (as with a Zilog 16C01), the AD pins of the IUSC can be directly connected to those of the host.

As a DMA master, the IUSC maintains 32-bit addresses. It presents the MS and LS 16 bits of an address as it drives /UAS and /AS Low, respectively, and this information is valid at the following rising edge. As a slave on a multiplexed bus, the IUSC captures addressing at rising edges on /AS. If this signaling is the same as that used on the host bus (as with a Zilog 16C0x), then the IUSC's /AS pin can be directly connected to the corresponding bus signal. Figure 2-1 shows such a system.

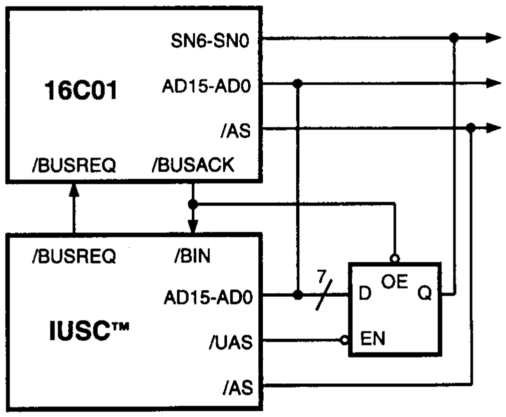


Figure 2-1. Simple Multiplexed System

2.2 MULTIPLEXED/NON-MULTIPLEXED OPERATION (Continued)

If the host's address strobe signaling is different from that of the IUSC (as with an 8086), then external logic must generate a compatible /AS signal for the IUSC.

Unless the rest of the system can use this /AS signal, external logic must also transform the /AS and /UAS issued by the IUSC as a bus master, to signaling that is compatible with the host bus. Figure 2-2 shows such an application.

If the host bus does not multiplex addresses and data, external devices must be added to latch the address when the IUSC is the bus master. Figures 2-3 and 2-4 illustrate two ways to interface the IUSC to a non-multiplexed bus. Figure 2-3 includes minimum hardware but requires that software write the register address into the IUSC each time it is going to access a register. In this mode, the IUSC's /AS pin should be pulled up to ensure a constant high logic level. The enhanced interface of Figure 2-4 includes drivers to sequence the low-order bits of the host address onto the IUSC'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 IUSC's registers.

The IUSC 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 IUSC captures the state of the low order AD lines, S//D, 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 or DMA Command/Address Register (CCAR or DCAR) before reading or writing a register. (If the host bus only includes eight data lines, AD13-AD8 can carry register addresses.)

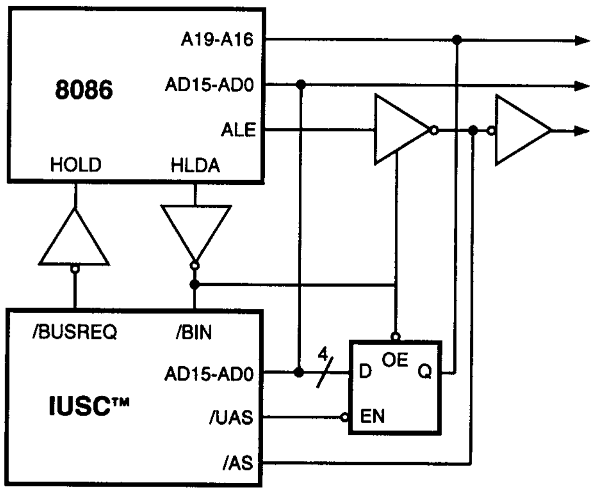


Figure 2-2. Multiplexed System with ALE-/AS-ALE Remapping

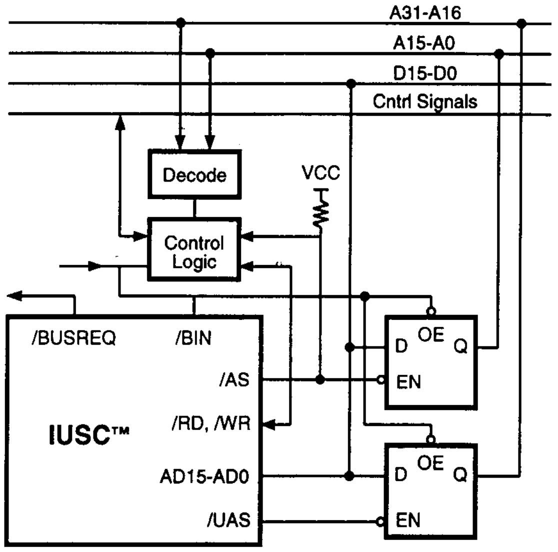


Figure 2-3. Simple Bus Demultiplexing

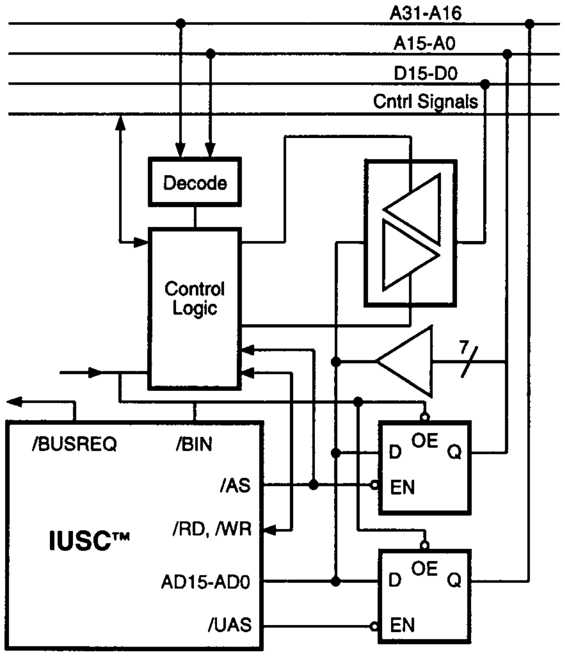


Figure 2-4. User-Friendly Bus Demultiplexing

2.3 READ/WRITE DATA STROBES

Another difference among host buses is the way in which read and write cycles are signaled and differentiated. Figures 2-6 and 2-7 show two standard methods supported by the IUSC. In Figure 2-6, the bus includes separate strobe lines for read and write cycles, commonly called /RD and /WR. In Figure 2-7, 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 IUSC 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 with resistors of about 10 kOhms.

There is no programmable option for the distinction between /RD-/WR and R//W-/DS operation. As a master the IUSC simply drives all four lines as shown in Figures 2-5 and 2-6. As a slave the IUSC responds to either pair of lines, which is why it is important to pull up the unused pair.

Also, as a slave the IUSC does not 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 of the bus signals /DS, /RD, and /WR may be active at a time. This restriction also includes /INTACK if it carries a strobe rather than a sampled level (see Chapter 6 for more information about interrupts). If the IUSC detects more than one of these inputs active simultaneously, it enters an inactive state from which the only escape is through the /RESET pin. When using multiple IUSCs, do not connect their /DS pins as well as their /RD and /WR pins, because the first time one of them becomes bus master and drives /DS and /RD or /DS and /WR Low, it will inactivate the others. Instead, provide separate pull-up resistors for each of the /DS pins, or for each of the /RD and /WR pins, whichever signals are not used.

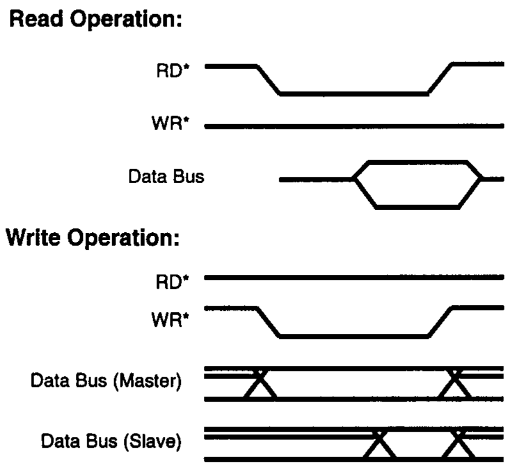


Figure 2-5. /RD and /WR Signaling

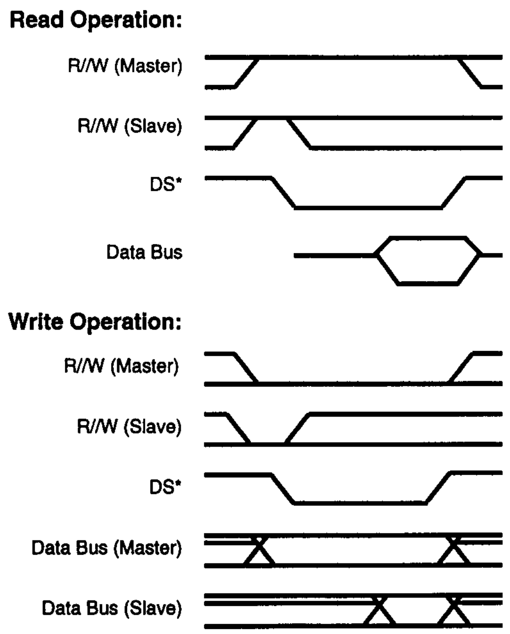


Figure 2-6. R/W and /DS Signaling

2.4 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 IUSC to transfer 16 bits at a time, in which case it is still possible to transfer eight bits when this is necessary or desirable. On a 16-bit data bus, the DMA channels can transfer either two data characters per cycle, or one per cycle with alternating cycles using AD15-AD8 and AD7-AD0.

Software can also restrict both master and slave operations to transferring only eight bits at a time on the AD7-AD0 pins. This leaves the AD15-AD8 pins unused during slave cycles: another BCR option allows them to carry register addresses. The latter option allows software to directly address IUSC registers even on a non-multiplexed bus, without having to write an address into the IUSC before it accesses a register.

2.5 ACK VS WAIT HANDSHAKING

The last major difference among host buses involves the handshaking signals that slave devices use for speed-matching with masters. Figure 2-7 illustrates the three variations in common use. In the first, called Wait signaling, if a master selects a slave and the slave cannot capture write data or provide read data within the time required to keep the master operating at full speed, it quickly (combinatorially) drives a Wait output Low, and then returns it to High when it is ready to complete the cycle. Some peripheral devices have Wait outputs that are open-collector or open-drain, which can be tied together for a negative logic wired-OR function. The IUSC drives its Ready/Wait output high or low at all times, so that a logic gate must be used to negative-logic OR (positive logic AND) its Wait line with Wait signal(s) from other devices, to produce the /WAIT input to the processor.

In the second scheme, "Acknowledge" signaling, all slaves must respond when a 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, some peripherals provide slave ACK outputs that are open-collector or open-drain, which can be tied together for a negative logic wired-OR function, while with the IUSC a logic gate must be used to negative-logic OR separate ACK lines to produce the Acknowledge input to the masters.

In the third scheme, "Ready" signaling, 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 differs from Wait signaling in the default state of the handshaking signal between cycles (High for Wait signaling, Low for Ready). It has similar timing as ACK signaling, but differs in the polarity of the handshaking signal. With Ready signaling, 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 IUSC supports Acknowledge and Ready signaling for all cycles, and Wait signaling for interrupt acknowledge cycles. The IUSC's register access times should be short enough to avoid the need for Wait signaling on all but the fastest processors. The board designer can combine the IUSC's /WAIT//RDY output with similar signals from other slaves, by means of an external logic gate or (for Acknowledge and Wait) by using an external tri-state or open-collector driver.

If software writes the Bus Configuration Register (BCR) at an address that makes the S//D pin Low, the IUSC drives /WAIT//RDY Low as an "Acknowledge" signal. If software writes the BCR with S//D High, the IUSC drives /WAIT//RDY as a "Wait" signal.

Ready signaling can be handled by using Acknowledge signaling and inverting the sense of the signal. When doing this, remember that /WAIT//RDY is bidirectional to the IUSC because of the on-chip DMA channels.

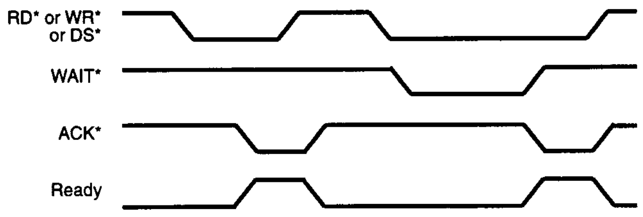


Figure 2-7. A Fast and Slow Cycle, with Three Kinds of Handshaking

2.6 BUS INTERFACE PIN DESCRIPTIONS

/RESET. *Reset* (input, active Low). A Low on this line places the IUSC 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.

CLK. *System Clock* (input). This signal is the timing reference for the DMA channels. (The serial controller section is clocked by the selected sources of receive and transmit clocking.)

AD15-AD0. *Address/Data Bus* (inputs/tri-state outputs). After Reset, these lines carry data between the controlling microprocessor and the IUSC, and may also carry multiplexed addresses of registers within the IUSC. Operation between the host processor and the IUSC is called Slave mode. Once the software has set up the IUSC™ and placed it into operation, these lines also carry multiplexed addresses and data between the IUSC and system memory; such operation is called master mode. AD15-AD0 can be used in a variety of ways based on whether the IUSC 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 IUSC. The IUSC ignores /CS when a Low on /INTACK indicates that the current bus operation is an interrupt acknowledge cycle. On a multiplexed bus, the IUSC 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.

S//D. *Serial/DMA* (input/tri-state output, input High indicates "serial"). Cycles with /CS Low, and /INTACK and S//D both High, access registers in the serial controller section. Cycles with /INTACK High, and /CS and S//D both Low, access registers in the DMA controller section. The state of this line when the Bus Configuration Register is written determines "wait vs acknowledge" operation, as described in a later section. On a multiplexed bus, the IUSC latches the state of this pin at rising edges of /AS, while on a non-multiplexed bus it latches the state at the leading/falling edges of /DS, /RD, or /WR.

Software can program the IUSC so that when it is acting as a bus master, it drives this line High to indicate a DMA cycle for serial data and Low to indicate an "array" or "list" access.

D//C. *Data/Control* (input/tri-state output, input High indicates Data). A slave read cycle with /CS Low, and /INTACK, S//D, and D//C High, fetches data from the serial controller's receive FIFO through the Receive Data Register (RDR). A slave write cycle with the same conditions writes data into the transmit FIFO through its Transmit Data Register (TDR). Slave cycles with /INTACK and S//D High, and /CS and this pin Low, read or write registers in the serial controller. On a multiplexed bus the IUSC latches the state of this pin at rising edges of /AS, while on a non-multiplexed bus it latches the state at leading/falling edges of /DS, /RD, or /WR.

For slave cycles using direct register addressing, with /INTACK High and both /CS and S//D Low, the state of this line at rising edges of /AS selects between the registers of the transmit DMA channel (Low) and those of the receive DMA channel (High). Using register pointer addressing with /INTACK High and /CS and S//D both Low, the IUSC ignores this line, taking the DMA channel selection from a bit in the DMA Command/Address Register. Software can program the IUSC so that when it is acting as a bus master, it drives this line High to indicate a DMA cycle for the receiver and Low to indicate a cycle for the transmitter.

/AS. *Address Strobe* (input/tri-state output, active Low). After a reset, the IUSC's bus interface logic monitors this signal to see if the host bus multiplexes addresses and data on AD15-AD0. If the logic sees activity on /AS before (or as) software writes the Bus Configuration Register, then in subsequent slave cycles directed to the IUSC, it captures register selection from the AD lines, S//D, and C//D on the rising edges of /AS.

When external logic generates pulses on /AS, it should not qualify them so that /AS goes low only on cycles directed to the IUSC. Since the IUSC captures the state of /CS only on rising edges of /AS, such a scheme renders the IUSC "always selected", so that it responds to every cycle on /RD, /WR, or /DS.

When the IUSC takes control of the bus and operates as a master, it always uses the bus in a multiplexed fashion, driving /AS Low when it places the least significant 16 bits of an address on the AD15-AD0 lines. External devices can be used to de-multiplex the address and data, if this is necessary to match the characteristics of the host processor or host bus.

For a non-multiplexed bus, this pin should be pulled up to +5V using a 10 kOhm resistor. If a processor uses a nonmultiplexed bus, yet has an output called Address Strobe (e.g., 680x0 devices), this pin should not be tied to the processor output.

/UAS. *Upper Address Strobe* (tri-state output, active Low). When the IUSC takes control of the bus and operates as a master, it drives /UAS Low when it places the more significant 16 bits of an address on AD15-AD0. External memory and other slave devices (or de-multiplexing latches) should capture the MS address at each rising edge on this line.

R//W. *Read/Write Control* (input/tri-state output, Low signifies "write"). R//W and /DS indicate read and write cycles on the bus, for host processors/buses using this kind of signaling. When the IUSC has taken control of the bus and is operating in Master mode, this pin is an output that remains valid throughout the Low time of /DS. In slave cycles, the IUSC samples R//W at each leading/falling edge of /DS.

/DS. *Data Strobe* (input/tri-state output, active Low). R//W and /DS indicate read and write cycles on the bus for host processors/buses having this kind of signaling. It is an output when the IUSC has taken control of the bus and is operating in Master mode, otherwise it is an input that is qualified by /CS Low or /INTACK Low. In Master mode, the R//W line remains valid throughout the Low time of this line. In Slave mode, the IUSC samples R//W at each leading/falling edge on this line. For slave write cycles and master read cycles, the IUSC captures data at the rising (trailing) edge on this line. For slave read cycles, the IUSC provides valid data on the AD lines within the specified access time after this line goes Low, and keeps the data valid until after the master releases this line to High. For master write cycles, the IUSC places valid data on the AD lines before it drives this signal to Low, and keeps the data valid until after it drives this line back to High.

/RD. *Read Strobe* (input/tri-state output, active Low). This line indicates a read cycle on the bus for host processors/buses having this kind of signaling. It is an output when the IUSC has taken control of the bus and is operating in Master mode, otherwise it is an input that is qualified by /CS Low or /INTACK Low. For master read cycles, the IUSC captures data at the rising (trailing) edge of this line. For slave read cycles, the IUSC provides valid data on the AD lines within the specified access time after this line goes Low, and keeps the data valid until after the master drives this line back to High.

/WR. *Write Strobe* (input/tri-state output, active Low). This line indicates write cycles on the bus, for host processors/buses having this kind of signaling. It is an output when the IUSC has taken control of the bus and is operating in Master mode, otherwise it is an input that is qualified by /CS Low. For slave write cycles, the IUSC captures write data at the rising (trailing) edge of this line. For master write cycles, the IUSC places valid data on the AD lines before it drives this signal to Low, and keeps the data valid until after it drives this line back to High.

Note: Only one of /DS, /RD, or /WR may be driven in one bus cycle.

B//W. *Byte/Word Select* (tri-state output, High indicates 8-bit transfer). When the IUSC takes control of the bus and operates as a master, a High on this line indicates that a byte is to be transferred, and a Low indicates that 16 bits are to be transferred. The IUSC ignores this signal during slave cycles: it takes the byte/word distinction from an AD line at the rising edge of /AS, or from a bit in the Channel Command/Address Register or DMA Command/Address Register.

/WAIT//RDY. *Wait, Ready, or Acknowledge Handshaking* (input/tri-state output, active Low). The IUSC drives this line full-time after Reset, except that it releases the line to act as an input when it has taken control of the bus and is operating in Master mode. In both directions, the line can carry "Wait" or "Acknowledge" signaling depending on the state of the S//D input during the initial BCR write. If S//D is High when the BCR is written, this line operates thereafter as a Ready/Wait line for Zilog and some Intel processors. In this mode the IUSC will not complete a master cycle while this line is Low, and it asserts this line Low until it is ready to complete an interrupt acknowledge cycle, but it never asserts this line when the host accesses one of the IUSC registers.

If S//D 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 IUSC will not complete a master cycle until this line is Low. It asserts this line Low for register read and write cycles, and when it is ready to complete an interrupt acknowledge cycle.

In any case /WAIT//RDY is a tri-state (not open-drain) output. The board designer can combine this signal with similar signals from other slaves by using an external logic gate or a tri-state or open-collector driver.

2.6 BUS INTERFACE PIN DESCRIPTIONS (Continued)

/INT. *Interrupt Request* (output, active Low). The IUSC drives this line Low when (1) its IEI pin is High, (2) one or more of its interrupt type (s) is (are) enabled and pending, and (3) the Interrupt Under Service flag is not set for its highest priority enabled/pending type, nor for any higher-priority internal type within the IUSC. Software can program whether the bus interface drives this pin in a totem-pole or an open-drain fashion.

/INTACK. *Interrupt Acknowledge* (input, active Low). A Low on this line indicates that the host processor is performing an interrupt acknowledge cycle. In some systems a Low on this line may further indicate that external logic has selected this IUSC as the device to be acknowledged, or as a potential device to be acknowledged. A field in the Bus Configuration Register selects whether this line carries a level-sensitive "status" signal that the IUSC should sample at the leading edge of /AS or /DS, or a single-pulse or double-pulse protocol. The IUSC will respond to an interrupt acknowledge cycle in a variety of ways depending on this programming and the state of the /INT and IEI lines, as described in Chapter 7.

IEI. *Interrupt Enable In* (input, active High). This signal and the IEO pin can be part of an interrupt-acknowledge daisy-chain with other devices that may request interrupts. If IEI is High outside of an interrupt acknowledge cycle, and one or more IUSC interrupt type(s) is (are) enabled and pending, and the Interrupt Under Service flag is not set for the (highest priority such) type nor for any higher-priority type within the IUSC, then the IUSC requests an interrupt by driving its /INT pin Low. If the IEI pin is High during an interrupt acknowledge cycle, and one or more IUSC interrupt type(s) is (are) enabled and pending, and the Interrupt Under Service flag is not set for the (highest priority such) type, nor for any higher-priority type within the IUSC, then the IUSC keeps IEO Low and responds to the cycle.

IEO. *Interrupt Enable Out* (output, active High). This signal and/or IEI can be part of an interrupt acknowledge daisy chain with other devices that may request interrupts. The IUSC drives its IEO pin Low whenever its IEI pin is Low, and/or whenever the Interrupt Under Service flag is set for any condition. The IUSC drives this signal slightly differently during an interrupt acknowledge cycle, in that it also forces IEO Low if it is (has been) requesting an interrupt.

/BUSREQ. *Bus Request* (output, active Low). The DMA controller section drives this line Low to request control of the host bus. /BUSREQ can be an open-drain or totem-pole output depending on a bit in the Bus Configuration Register. In open-drain mode, the IUSC samples the pin as an input and only drives it Low after sampling it High.

/BIN. *Bus Acknowledge In* (input, active Low). When the IUSC receives a falling edge on this input, it samples whether it has been driving (or has just begun to drive) /BUSREQ. If so, it keeps /BOUT High and takes control of the host bus. If not, it "passes the bus grant" by driving /BOUT Low. This signal can be used with /BOUT to form a bus-grant daisy chain for arbitration of bus control. Alternatively, it can be connected to a direct, positive grant from an external arbiter, and the /BOUT pin can be left unconnected.

/BOUT. *Bus Acknowledge Out* (output, active Low). As noted above, this signal can be used with /BIN to form a bus-grant daisy chain for arbitration of bus control.

/ABORT. *Abort Master Cycle* (input, active Low). A Low on this line during a master cycle makes the currently active DMA channel terminate its activity and enter a disabled state. Note that /ABORT is only effective during a DMA cycle, so that the IUSC knows which channel should be aborted. Also note that external logic must set /WAIT//RDY to the right state for the cycle to complete, in order for /ABORT to have an effect.

V_{CC}, V_{SS}. *Power and Ground.* The inclusion of seven pins for each power rail insures good signal integrity, helps prevent transients on outputs, and improves noise margins on inputs. The IUSC's internal power distribution network requires that all these pins be connected appropriately.

2.7 PULL-UP RESISTORS AND UNUSED PINS

All unused input pins should be pulled up, either by connecting them directly to Vcc or with a resistor. This may include /INTACK, IEI, and /ABORT.

Bi-directional pins should typically be pulled up with a 10 kOhm resistor, to allow the IUSC to drive them as outputs. This always includes /AS, R//W, /DS, /RD, and /WR, and may also include /TxREQ, /RxREQ, /TxC, /RxC, /CTS, /DCD, and PORT7-0. Furthermore, when multiple IUSCs are used in a design, whichever pin(s) is (are) unused among /DS, /RD, and /WR should be pulled up individually (using one resistor per unused device input) rather than being connected among all the devices.

Tri-statable output pins may need to be pulled up to protect external logic from the effects of having a floating input. Again, a 10 kOhm resistor is recommended. This may include /BUSREQ, /UAS, B//W, TxD, and /INT.

2.8 THE BUS CONFIGURATION REGISTER (BCR)

The BCR is a 16-bit register as shown in Figure 2-8. All the bits in the BCR reset to zero. Software's first access to the IUSC after a hardware reset, must be a write to the BCR. If the host processor handles 16-bit data, and the data bus between it and the IUSC is at least 16 bits wide, then the software's initial access to the IUSC 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 IUSC, all data should be transferred on the AD7-AD0 pins. In such a system, pull-down resistors of about 10 kOhms should be attached to the AD15-AD8 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.)

2.8.1 Wait vs Ready Selection

The following paragraphs describe the significance of the various bits and fields in the BCR. Besides these data bits, the IUSC captures the state of the S//D pin when the software writes the BCR. It uses this captured state after the BCR write, such that if S//D was Low, it drives the /WAIT //RDY pin as an "acknowledge" (or an inverted "ready") signal during register accesses and interrupt acknowledge cycles, while if S//D 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.

2.8.2 Bits and Fields in the BCR

SepAd (Separate Address; BCR15). This bit should only be written as 1 with 16 Bit=0. This combination conditions the IUSC to use AD7-AD0 for data and to take register addressing from AD13-AD8. In this mode the IUSC takes the Upper/Lower byte indication (U//L) from AD8 and the register address from AD13-AD9. The external drivers for these signals must be tri-stated when the IUSC is the bus master.

With this interfacing technique, the BCR must be written at an address such that AD13-AD8 are Low (0). Further, AD15 must be High (1) and AD14 must be Low (0) 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 pull-down resistor to AD14, both being about 10 kOhms.

This mode is useful with a non-multiplexed bus, to avoid making the software write a register address to CCAR or DCAR before each register access. In this mode the IUSC captures the state of AD13-AD8 on each leading/falling edge on /DS, /RD, or /WR. But software can still program SepAd=1 (with 16 Bit=0) when the IUSC has detected early activity on /AS. In this case the IUSC captures addressing from AD13-AD8 on each rising edge of /AS, rather than from the Low-order AD lines as would be true with SepAd=0.

2.8.2 Bits and Fields in the BCR (Continued)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SepAd	Reserved									IAckMode		BRQTP	16-Bit	/IRQTP	SRight A

Figure 2-8. Bus Configuration Register (BCR)

The predecessor Z16C31 device used BCR7-6 as a "ByteSwap" field that controlled how the Transmit DMA channel captured bytes from the D15-D0 lines when it was reading bytes over a 16-bit bus. On the Z16C32 these bits are Reserved—software written for the Z16C31 may program them with 10 or 11, but new software should write 00 to this field. In effect, the Z16C32's Transmit DMA channel uses 16 Bit (BCR2) in place of BCR7, to control whether it fetches bytes from the two halves of the bus alternately, and uses the state bit that's controlled by "Select D15-D8 or D7-D0 First" commands in place of BCR6, to control which half of the bus corresponds to even and odd addresses.

The **IAckMode** field (BCR5-4) controls how the host processor drives the /INTACK pin. 00 indicates that the IUSC should capture the state of /INTACK at the start of each bus cycle. On a multiplexed bus it does this at the rising edges of /AS, while on a bus with separate address and data lines it does so at falling edges on /DS or /RD.

This field should be 01 if /INTACK carries a single Low-active pulse during interrupt acknowledge cycles.

The 10 value in this field is reserved and should not be programmed.

IAckMode should be 11 if /INTACK carries two pulses during an interrupt acknowledge sequence. This mode is compatible with several Intel microprocessors.

BRQTP (Bus Request Totem-Pole; BCR3). If this bit is 1, the IUSC drives its /BUSREQ pin in a totem-pole fashion (both High and Low). If it is 0, the IUSC drives /BUSREQ in an open-drain fashion (Low only), in which case an external pull-up resistor should be provided. In the latter case, the IUSC samples /BUSREQ before driving it; if the pin is Low, the logic waits until it goes High before driving it back to Low.

16-Bit (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 three main effects: it restricts the IUSC to using byte operations on AD7-AD0 when it is the bus master, it restricts the host to using byte transfers on AD7-AD0 when reading and writing the IUSC's registers, and it makes the IUSC ignore the state of the "B//W" signal or bit for register accesses. This bit also controls whether "implicit" accesses to the CCAR, TDR, RDR, and DCAR are 8 or 16 bit wide.

/IRQTP (Interrupt Request Totem-Pole; BCR1). If this bit is 0, the IUSC drives its /INT pin in a totem-pole fashion (both High and Low). If /IRQTP is 1, the IUSC drives /INT in an open-drain fashion (Low only), in which case it should have an external pull-up resistor.

SRightA (Shift Right Addresses; BCR0). This bit is significant only for a multiplexed bus—the IUSC ignores it for a nonmultiplexed bus. If SRightA is 1, the IUSC captures slave register addressing from the AD6-AD0 pins and ignores the AD7 pin. In this mode, AD0 carries the Upper/Lower byte indication (U//L), AD5-AD1 carry the actual register address, and AD6 carries the Byte/Word indication (B//W). If SRightA is 0, the IUSC captures addressing from AD7-AD1 and ignores AD0. It takes U//L from AD1, the register address from AD6-AD2, and B//W from AD7. This bit applies to accesses to both the serial and DMA sections of the IUSC, but it has no effect on the use of the S//D and D//C pins.

SRightA should be 0 in order to use the IUSC as an 8-bit peripheral on a 16-bit bus, which is not likely to be a common application. Some sections of this manual assume that SRightA is 1.

All bits in the BCR, other than those described above, are reserved and should be programmed as 0. If the processor can only write bytes to the IUSC, software can only write the 8 LS Bits of the BCR, on the AD7-AD0 lines. In this case, the state of AD15-AD8, when software writes the BCR, must be ensured by connecting these pins to pull-down resistors of about 10 kOhms or, if SepAd=1, to host address lines.

2.9 REGISTER ADDRESSING

The flowchart of Figures 2-9 and 2-10 shows the complete process by which the IUSC determines which register to access when a host processor cycle asserts /CS and one of /RD, /WR, or /DS.

In all accesses to IUSC registers, the S//D pin selects between registers in the serial controller and those in the DMA controllers. The IUSC samples S//D, and other pins as described below, at the rising/trailing edge of /AS, or, if /AS is pulled up so that it's always High, at the falling/leading edge of /DS, /RD, or /WR.

2.9.1 Implicit Serial Data Register Addressing

If the IUSC samples S//D and the D//C pin both High, a write operation accesses the Transmit Data Register (TDR) and a read operation selects the Receive Data Register (RDR). The access is implicitly 16 bits wide if the 16-bit in the Bus Configuration Register (BCR2) is 1 (indicating a 16-bit data bus) or eight bits wide if BCR2 is 0.

This means that, on a 16-bit bus, 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 LS byte of the TDR/RDR, either directly or by writing it to the CCAR.

2.9.2 Direct Serial Register Addressing on AD13-AD8

If the IUSC samples S//D High and D//C Low, it accesses a serial controller register. If the SepAd bit in the Bus Configuration Register (BCR15) is 1 (which should only be the case with an 8-bit data bus) the IUSC samples the AD13-AD9 pins as a register address to select which serial controller register to access, and samples AD8 as U//L to select which byte of the register to access. The IUSC always interprets a U//L bit in the "Little-Endian" fashion, with a 1 indicating the more-significant eight bits of the register. If the IUSC samples AD13-AD8 as all 0's in this mode, indicating the Channel Command/Address Register (CCAR), the IUSC uses the contents of the CCAR to select which register to access, as described in Indirect Serial Register Addressing 2.9.4.

2.9 REGISTER ADDRESSING (Continued)

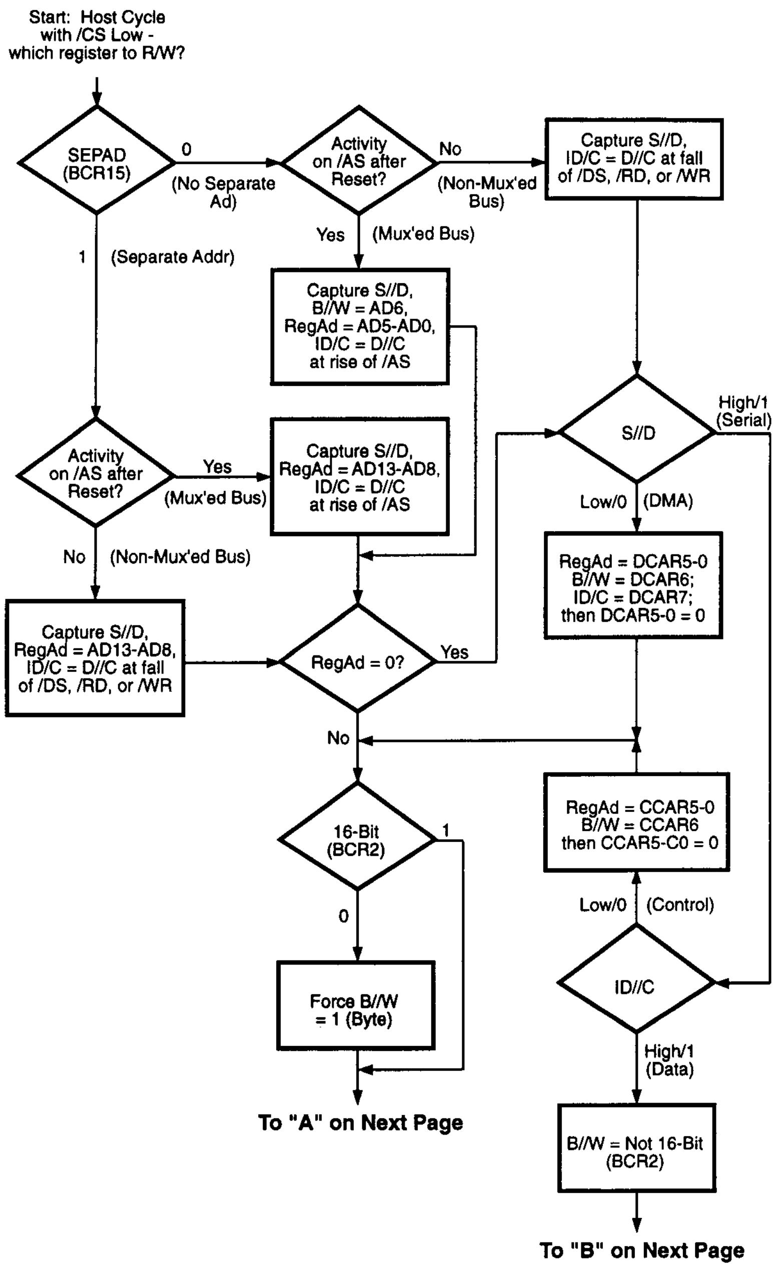

Start: Host Cycle with /CS Low - which register to R/W?
SEPAD (BCR15)
0
(No Separate Ad)
1
(Separate Addr)
Activity on /AS after Reset?
No
(Non-Mux'ed Bus)
Yes
(Mux'ed Bus)
Capture S//D, ID/C = D//C at fall of /DS, /RD, or /WR
Capture S//D, B//W = AD6, RegAd = AD5-AD0, ID/C = D//C at rise of /AS
Activity on /AS after Reset?
Yes
(Mux'ed Bus)
No
(Non-Mux'ed Bus)
Capture S//D, RegAd = AD13-AD8, ID/C = D//C at rise of /AS
Capture S//D, RegAd = AD13-AD8, ID/C = D//C at fall of /DS, /RD, or /WR
RegAd = 0?
Yes
No
S//D
High/1 (Serial)
Low/0
(DMA)
RegAd = DCAR5-0 B//W = DCAR6; ID/C = DCAR7; then DCAR5-0 = 0
RegAd = CCAR5-0 B//W = CCAR6 then CCAR5-C0 = 0
Low/0
(Control)
ID//C
High/1 (Data)
B//W = Not 16-Bit (BCR2)
16-Bit (BCR2)
1
0
Force B//W = 1 (Byte)
To "A" on Next Page
To "B" on Next Page

Figure 2-9. IUSC Register Addressing (1 of 2)

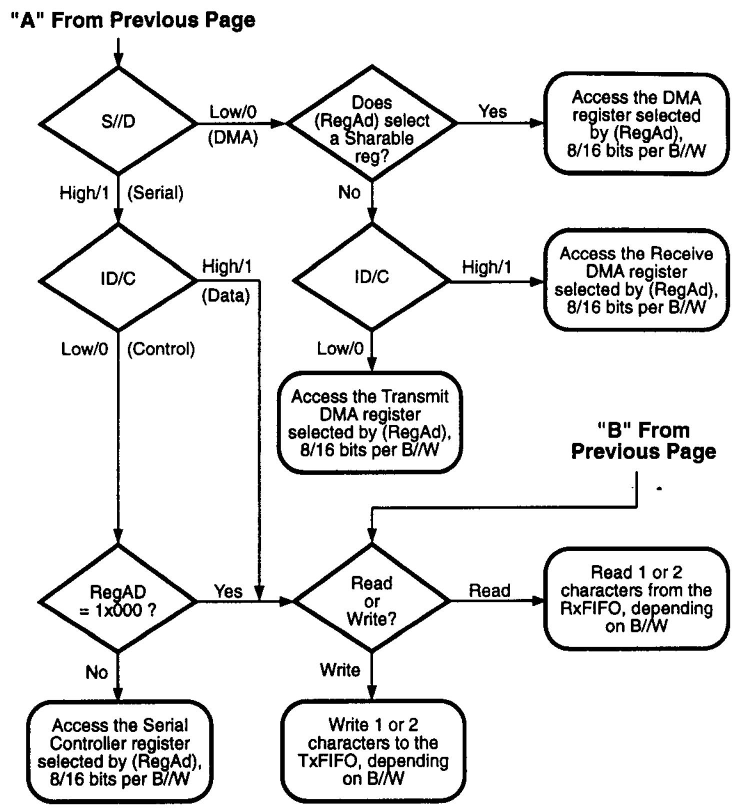

"A" From Previous Page
S//D
Low/0
(DMA)
Does (RegAd) select a Sharable reg?
Yes
Access the DMA register selected by (RegAd), 8/16 bits per B//W
High/1
(Serial)
No
ID/C
High/1
(Data)
ID/C
High/1
Access the Receive DMA register selected by (RegAd), 8/16 bits per B//W
Low/0
(Control)
Low/0
Access the Transmit DMA register selected by (RegAd), 8/16 bits per B//W
"B" From Previous Page
RegAD = 1x000 ?
Yes
Read or Write?
Read
Read 1 or 2 characters from the RxFIFO, depending on B//W
No
Write
Access the Serial Controller register selected by (RegAd), 8/16 bits per B//W
Write 1 or 2 characters to the TxFIFO, depending on B//W

Figure 2-10. IUSC Register Addressing (2 of 2)

2.9.3 Direct Serial Register Addressing on AD6-AD0 or AD7-AD1

If the IUSC samples S//D High and D//C Low, SepAd (BCR15) is 0, and the IUSC detected activity on /AS before or as the BCR was written, the IUSC samples the Low-order AD pins to determine what kind of serial controller register access it should do. It takes the register selection (RegAd) from AD5-AD1 if SRightA (BCR0) is 1, or from AD6-AD2 if SRightA is 0. If 16-bit (BCR2) is 1, the IUSC samples AD6 (or AD7 if SRightA/BCR0 is 0) as **B//W** to determine whether to access all 16 bits of the register (if B//W is 0) or just eight bits. If 16-bit is 0 or B//W is 1, it samples AD0 (or AD1 if SRightA is 0) as U//L to select which byte of the register to access. The IUSC always interprets a U//L bit in the "Little-Endian" fashion, with a 1 indicating the more-significant eight bits of the register. U//L should be 0 for all 16-bit accesses.

If the IUSC samples AD6-AD0 (or AD7-AD1 if SRightA is 1) as all 0's in this mode, indicating the Channel Command/ Address Register (CCAR), the IUSC uses the contents of the CCAR to select which register to access, as described in the next section.

2.9.4 Indirect Serial Register Addressing in the CCAR

If the IUSC samples S//D High and D//C Low, and:

1. SepAd (BCR15) is 1 and the IUSC samples AD13-AD8 as all zero indicating the CCAR, or
2. SepAd is 0, the IUSC detected activity on /AS before or as the BCR was written, and it samples AD6-AD0 as all zero indicating the CCAR, or
3. SepAd is 0 and the IUSC did not detect activity on /AS before nor as the BCR was written,

then it uses the less-significant byte of the CCAR to select how to access a serial controller register.

Figure 2-11 shows the CCAR. When the IUSC takes indirect register addressing from it, the **RegAd** field (CCAR5-1) selects which register to access. If 16-bit (BCR2) is 1, the IUSC uses CCAR6 as B//W to determine whether to access all 16 bits if the register (if B//W is 0) or just 8. If 16-bit is 0 or B//W is 1, it uses CCAR0 as U//L to select which byte of the register to access. The IUSC always interprets a U//L bit in the "Little-Endian" fashion, with a 1 indicating the more-significant eight bits of the register. U//L should be 0 for all 16-bit accesses.

Whenever it uses CCAR as an indirect address, the IUSC thereafter clears CCAR6-0 to zero, so that the next access to the CCAR address again references all 16 bits of the CCAR itself. Thus, after writing a register address to the CCAR, reading or writing the CCAR address accesses the register selected by the address written, but another write to the CCAR address thereafter again writes an address into the CCAR.

CCAR or DCAR can always be used to select a register for a subsequent access to the CCAR or DCAR address, even if the IUSC detected activity on /AS after Reset, and regardless of the state of SepAd (BCR15).

Typically when software uses indirect register addressing, the CCAR and DCAR are the only register addresses it reads and writes, every other access being to write a register address. Note that the CCAR itself can be accessed in a single read or write operation: for example, to write a command to the RTCmd field, software does not have to first write the address of the CCAR (which is zero). Specifying a register address for the next access to the CCAR can be done in the same write operation with issuing a command in RTCmd and/or changing the RTMode field.

'The RxD and TxD Pins' in Chapter 4 describes how the RTMode field in the CCAR controls echoing and looping between the Transmitter and Receiver. Typically this field is zero, but in applications using indirect register addressing and non-zero RTMode values, software must take care to preserve the current value of RTMode when it writes register addresses to the CCAR.

When using indirect addressing, some hardware/software mechanism has to prevent an IUSC interrupt, or any interrupt that leads to a context switch away from an interrupted IUSC task, from occurring between the time an address is written into the CCAR and when the subsequent read or write is done. This is because an address that has been written into the CCAR is part of the interrupted task's context that would want to be saved, but there is no way to read such an address out of the IUSC—reading the CCAR returns the contents of the addressed register!

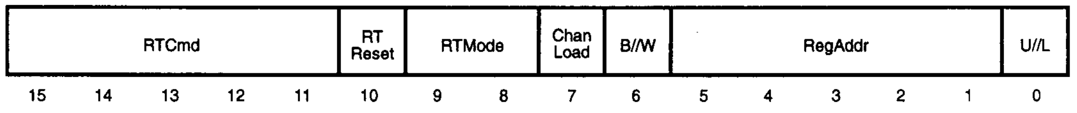


Figure 2-11. Channel Command/Address Register (CCAR)

2.9.5 Direct DMA Register Addressing on AD13-AD8

If the IUSC samples S//D Low, it accesses a DMA controller register. If the SepAd bit in the Bus Configuration Register (BCR15) is 1 (which should only be the case with an 8-bit data bus) the IUSC samples the AD13-AD9 pins as a RegAd to select which DMA controller register to access.

DMA controller registers fall into two categories. One copy of each "shareable" register (e.g., DCAR) applies to both channels, while other registers are duplicated in each channel (e.g., TDMR and RDMR). When the address on AD13-AD9 indicates a shareable register, the IUSC ignores the D//C pin, while for non-shareable registers it samples D//C to select whether to access the Transmit or Receive channel register. D//C High (1) selects the Receive channel.

The IUSC samples AD8 as U//L to select which byte of the register to access. The IUSC always interprets a U//L bit in the "Little-Endian" fashion, with a 1 indicating the more-significant eight bits of the register.

If the IUSC samples AD13-AD8 as all zero in this mode, indicating the DMA Command/Address Register (DCAR), it uses the contents of the DCAR to select which register to access, as described in **Indirect DMA Register Addressing** 2.11.7.

2.9.6 Direct DMA Register Addressing on AD6-AD0 or AD7-AD1

If the IUSC samples S//D Low, SepAd (BCR15) is 0, and it detected activity on /AS before or as the BCR was written, the IUSC samples the low-order AD pins to determine what kind of DMA controller register access it should do. It takes the register selection (RegAd) from AD5-AD1 if SRightA (BCR0) is 1, or from AD6-AD2 if SRightA is 0.

DMA controller registers fall into two categories. There is only one copy of each "shareable" register (e.g., DCAR), while other registers are duplicated in each channel (e.g., TDMR and RDMR). When RegAd indicates a shareable register, the IUSC ignores the D//C pin, while for non-shareable registers it samples D//C to select whether to access the Transmit or Receive channel register. D//C High (1) selects the Receive channel.

If 16-Bit (BCR2) is 1, the IUSC samples AD6 (or AD7 if SRightA/BCR0 is 0) as B//W to determine whether to access all 16 bits of the register (if B//W is 0) or just 8 bits. If 16-Bit is 0 or B//W is 1, it samples AD0 (or AD1 if SRightA is 0) as U//L to select which byte of the register to access. The IUSC always interprets a U//L bit in the "Little-Endian" fashion, with a 1 indicating the more-significant eight bits of the register. U//L should be 0 for all 16-bit accesses.

If the IUSC samples AD6-AD0 (or AD7-AD1 if SRightA is 1) as all zero in this mode, indicating the DMA Command/Address Register (DCAR), it uses the contents of the DCAR to select which register to access, as described in the next section.

2.9.7 Indirect DMA Register Addressing in the DCAR

If the IUSC samples S//D Low, and:

1. SepAd (BCR15) is 0 and the IUSC did not detect activity on /AS before nor as the BCR was written, or

2. SepAd is 1 and the IUSC samples AD13-AD8 as all zero indicating the DCAR, or

3. SepAd is 0, the IUSC detected activity on /AS before or as the BCR was written, and it samples AD6-AD0 as all zero indicating the DCAR,

then it uses the less-significant byte of the DCAR to select how to access a DMA controller register.

Figure 2-12 shows the DCAR. When the IUSC takes indirect addressing from it, the RegAd field (DCAR5-1) selects which register to access.

DMA controller registers fall into two categories. There is only one copy of each "shareable" register (e.g., DCAR), while other registers are duplicated in each channel (e.g., TDMR and RDMR). When RegAd indicates a shareable register, the IUSC ignores the **Rx/Tx Reg** field (DCAR7), while for non-shareable registers it uses Rx/Tx Reg to select whether to access the Transmit or Receive channel register. A 1 selects the Receive channel. (The IUSC ignores the D//C pin when taking indirect addressing from the DCAR.)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DCmd				Reserved		Rx/Tx Cmd	MBRE	Rx/Tx Reg	B//W	RegAddr					U//L

Figure 2-12. DMA Command/Address Register (DCAR)

2.9.7 Indirect DMA Register Addressing in the DCAR (Continued)

If 16-bit (BCR2) is 1, the IUSC uses DCAR6 as B//W to determine whether to access all 16 bits of the register (if B//W is 0) or just eight bits. If 16-bit is 0 or B//W is 1, it uses DCAR0 as U//L to select which byte of the register to access. The IUSC always interprets a U//L bit in the "Little-Endian" fashion, with a 1 indicating the more-significant eight bits of the register. U//L should be 0 for all 16-bit accesses.

Whenever it uses DCAR as an indirect address, the IUSC thereafter clears DCAR6-0 to zero, so that the next access to the DCAR address again refers to all 16 bits of the DCAR itself. Thus, after writing a register address to the DCAR, reading or writing the DCAR address accesses the register selected by the address written, but writing to the DCAR address thereafter writes another address into the DCAR.

Typically when software uses indirect register addressing, the DCAR and CCAR are the only register addresses it reads and writes, every other access being to write a register address. Note that the DCAR itself can be accessed in a single read or write operation: for example, to write a command to the DCmd field, software does not have to first write the address of the DCAR (which is zero). Specifying a register address for the next access to the DCAR can be done in the same write operation with issuing a command in DCmd and/or changing the MBRE bit.

Commands and /BUSREQ Enable in Chapter 6 describes how the MBRE bit in the DCAR controls whether the IUSC can request control of the host bus to do DMA transfers. Typically this bit is set. **When indirect register addressing is used, software must take care to preserve the current state of MBRE when it writes register addresses to the DCAR.**

When using indirect addressing, some hardware/software mechanism has to prevent an IUSC interrupt, or any interrupt that leads to a context switch away from an interrupted IUSC task, from occurring between the time an address is written into the DCAR and when the subsequent read or write is done. This is because an address that has been written into the DCAR is part of the interrupted task's context that would want to be saved, but there is no way to read such an address of the IUSC—reading the DCAR returns the contents of the addressed register.

2.9.8 About the Register Address Tables

Tables 2-1 and 2-2 show the names and addresses of the addressable registers in the IUSC, in address and alphabetical order. The Direct Address columns assume that SRightA (BCR0) is 1. The RegAddr column in the Tables reflects the state of AD5-AD1, AD13-AD9, CCAR5-1, or DCAR5-1 as applicable.

If 16-Bit (BCR2) is 1, the B//W bit from AD6, AD14, CCAR6, or DCAR6 selects between a 16-bit transfer (if 0/Low) and an 8-bit transfer (if 1). If "16-bit" is 0, the IUSC ignores AD6, AD14, CCAR6, or DCAR6 (as applicable). Note that the values in the "8-bit data" columns of Tables 2-1 and 2-2 include the B//W bit 1 for both direct and indirect addressing, as is required on a 16-bit bus. When 16-bit (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-bit or 16-bit bus, the state of AD0, AD8, CCAR0, or DCAR0 selects the more-significant eight bits of the register (if 1/High) or the less-significant eight bits. In this regard, and in the register addresses of the two halves of 32-bit DMA address registers, the IUSC is "Little-Endian" like Intel microprocessors. (The next section describes the IUSC's byte-ordering flexibility in the TDR and RDR, and in address fetching in DMA Array or Linked List mode.) For 16-bit transfers, AD0, AD8, CCAR0 or DCAR0 should be 0/Low.

The Direct Address columns of the Tables assume:

1. SRightA (BCR0) is 1,
2. the processor's multiplexed AD6-AD0 lines are connected to AD6-AD0, or its A5-A0 lines are connected to AD13-AD8, depending on SepAd (BCR15),
3. the processor's A7 line is connected to D//C, and
4. the processor's A8 line is connected to S//D.

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

2.9.9 Serial Data Registers RDR and TDR

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

The MS bytes 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 353 or 369 (161 or 171 hex) to select the LS byte for byte transfers. On an Intel-based system, use direct addresses 352 or 368 (160 or 170 hex) to select the LS byte for byte transfers.

Table 2-1. IUSC Registers, in Address Order

Register Name	Acronym	S//D	D//C	Reg Addr	Direct Address: 16-Bit Data	Direct Address: 8-Bit Data	DCAR7-0 or *CCAR6-0:* 16-Bit Data	DCAR7-0 or *CCAR6-0:* 8-Bit Data
DMA Command/Address	DCAR	L (0)	x	00000	0/0	64,5/40,1	0/0	64,5/40,1
Transmit DMA Mode	TDMR	L (0)	L (0)	00001	2/2	66,7/42,3	2/2	66,7/42,3
DMA Control	DCR	L (0)	x	00011	6/6	70,1/46,7	6/6	70,1/46,7
DMA Array Count	DACR	L (0)	x	00100	8/8	72,3/48,9	8/8	72,3/48,9
Burst/Dwell Control	BDCR	L (0)	x	01001	18/12	82,3/52,3	18/12	82,3/52,3
DMA Interrupt Vector	DIVR	L (0)	x	01010	20/14	84,5/54,5	20/14	84,5/54,5
DMA Interrupt Control	DICR	L (0)	x	01100	24/18	88,9/58,9	24/18	88,9/58,9
Clear DMA Interrupt	CDIR	L (0)	x	01101	26/1A	90,1/5A,B	26/1A	90,1/5A,B
Set DMA Interrupt	SDIR	L (0)	x	01110	28/1C	92,3/5C,D	28/1C	92,3/5C,D
Transmit DMA Interrupt Arm	TDIAR	L (0)	L (0)	01111	30/1E	94,5/5E,F	30/1E	94,5/5E,F
Transmit Byte Count	TBCR	L (0)	L (0)	10101	42/2A	106,7/6A,B	42/2A	106,7/6A,B
Transmit Address (Lower)	TARL	L (0)	L (0)	10110	44/2C	108,9/6C,D	44/2C	108,9/6C,D
Transmit Address (Upper)	TARU	L (0)	L (0)	10111	46/2E	110,1/6E,F	46/2E	110,1/6E,F
Next Transmit Byte Count	NTBCR	L (0)	L (0)	11101	58/3A	122,3/7A,B	58/3A	122,3/7A,B
Next Transmit Address (L)	NTARL	L (0)	L (0)	11110	60/3C	124,5/7C,D	60/3C	124,5/7C,D
Next Transmit Address (U)	NTARU	L (0)	L (0)	11111	62/3E	126,7/7E,F	62/3E	126,7 /7E,F
Receive DMA Mode	RDMR	L (0)	H (1)	00001	130/82	194,5/C2,3	130/82	194,5/C2,3
Receive DMA Interrupt Arm	RDIAR	L (0)	H (1)	01111	158/9E	222,3/DE,F	158/9E	222,3/DE,F
Receive Byte Count	RBCR	L (0)	H (1)	10101	170/AA	234,5/EA,B	170/AA	234,5/EA,B
Receive Address (Lower)	RARL	L (0)	H (1)	10110	172/AC	236,7/EC,D	172/AC	236,7/EC,D
Receive Address (Upper)	RARU	L (0)	H (1)	10111	174/AE	238,9/EE,F	174/AE	238,9/EE,F
Next Receive Byte Count	NRBCR	L (0)	H (1)	11101	186/BA	250,1/FA,B	186/BA	250,1/FA,B
Next Receive Address (L)	NRARL	L (0)	H (1)	11110	188/BC	252,3/FC,D	188/BC	252,3/FC,D
Next Receive Address (U)	NRARU	L (0)	H (1)	11111	190/BE	254,4/FE,F	190/BE	254,5/FE,F
Channel Command/Address	CCAR	H (1)	L (0)	00000	256/100	320,1/140,1	***0/0***	***64,65/40,1***
Channel Mode	CMR	H (1)	L (0)	00001	258/102	322,3/142,3	***2/2***	***66,7/42,3***
Channel Command/Status	CCSR	H (1)	L (0)	00010	260/104	324,5/144,5	***4/4***	***68,9/44,5***
Channel Control	CCR	H (1)	L (0)	00011	262/106	326,7/146,7	***6/6***	***70,1/46,7***
Port Status	PSR	H (1)	L (0)	00100	264/108	328,9/148,9	***8/8***	***72,3/48,9***
Port Control	PCR	H (1)	L (0)	00101	266/10A	330,1/14A,B	***10/0A***	***74,5/4A,B***

2.9.9 Serial Data Registers RDR and TDR (Continued)

Table 2-1. IUSC Registers, in Address Order (Continued)

Register Name	Acronym	S//D	D//C	Reg Addr	Direct Address: 16-Bit Data	Direct Address: 8-Bit Data	DCAR7-0 or CCAR6-0: 16-Bit Data	DCAR7-0 or CCAR6-0: 8-Bit Data
Test Mode Data	TMDR	H (1)	L (0)	00110	268/10C	332,3/14C,D	***12/0C***	***76,7/4C,D***
Test Mode Control	TMCR	H (1)	L (0)	00111	270/10E	334,5/14E,F	***14/0E***	***78,9/4E,F***
Clock Mode Control	CMCR	H (1)	L (0)	01000	272/110	336,7/150,1	***16/10***	***80,1/50,1***
Hardware Configuration	HCR	H (1)	L (0)	01001	274/112	338,9/152,3	***18/12***	***82,3/52,3***
Interrupt Vector	IVR	H (1)	L (0)	01010	276/114	340,1/154,5	***20/14***	***84,5/54,5***
Input/Output Control	IOCR	H (1)	L (0)	01011	278/116	342,3/156,7	***22/16***	***86,7/56,7***
Interrupt Control	ICR	H (1)	L (0)	01100	280/118	344,5/158,9	***24/18***	***88,9/58,9***
Daisy-Chain Control	DCCR	H (1)	L (0)	01101	282/11A	346,7/15A,B	***26/1A***	***90,1/5A,B***
Misc. Interrupt Status	MISR	H (1)	L (0)	01110	284/11C	348,9/15C,D	***28/1C***	***92,3/5C,D***
Status Interrupt Control	SICR	H (1)	L (0)	01111	286/11E	350,1/15E,F	***30/1E***	***94,5/5E,F***
Receive Data	RDR	H (1)	L (0)	1x000	288/120	352,3/160,1	***32/20***	***96/60***
(Read only; TDR for Write)			or H (1)	xxxxx	384-511	384-511	***xxx***	***xxx***
Receive Mode	RMR	H (1)	L (0)	10001	290/122	354,5/162,3	***34/22***	***98,9/62,3***
Receive Command/Status	RCSR	H (1)	L (0)	10010	292/124	356,7/164,5	***36/24***	***100,1/64,5***
Receive Interrupt Control	RICR	H (1)	L (0)	10011	294/126	358,9/166,7	***38/26***	***102,3/66,7***
Receive Sync	RSR	H (1)	L (0)	10100	296/128	360,1/168,9	***40/28***	***104,5/68,9***
Receive Count Limit	RCLR	H (1)	L (0)	10101	298/12A	362,3/16A,B	***42/2A***	***106,7/6A,B***
Receive Character Count	RCCR	H (1)	L (0)	10110	300/12C	364,5/16C,D	***44/2C***	***108,9/6C,D***
Time Constant 0	TC0R	H (1)	L (0)	10111	302/12E	366,7/16E,F	***46/2E***	***110,1/6E,F***
Transmit Data	TDR	H (1)	L (0)	1x000	304/130	368,9/170,1	***48/30***	***112/70***
(Write only; RDR for Read)			or H (1)	xxxxx	384-511	384-511	***xxx***	***xxx***
Transmit Mode	TMR	H (1)	L (0)	11001	306/132	370,1/172,3	***50/32***	***114,5/72,3***
Transmit Command/Status	TCSR	H (1)	L (0)	11010	308/134	372,3/174,5	***52/34***	***116,7/74,5***
Transmit Interrupt Control	TICR	H (1)	L (0)	11011	310/136	374,5/176,7	***54/36***	***118,9/76,7***
Transmit Sync	TSR	H (1)	L (0)	11100	312/138	376,7/178,9	***56/38***	***120,1/78,9***
Transmit Count Limit	TCLR	H (1)	L (0)	11101	314/13A	378,9/17A,B	***58/3A***	***122,3/7A,B***
Transmit Character Count	TCCR	H (1)	L (0)	11110	316/13C	380,1/17C,D	***60/3C***	***124,5/7C,D***
Time Constant1	TC1R	H (1)	L (0)	11111	318/13E	382,3/17E,F	***62/3E***	***126,7/7E,F***

Table 2-2. IUSC Registers, In Alphabetical Order

Register Name	Acronym	S//D	D//C	Reg Addr	Direct Address: 16-Bit Data	Direct Address: 8-Bit Data	DCAR7-0 or CCAR6-0: 16-Bit Data	DCAR7-0 or CCAR6-0: 8-Bit Data
Burst/Dwell Control	BDCR	L (0)	x	01001	18/12	82,3/52,3	18/12	82,3/52,3
Channel Command/Address	CCAR	H (1)	L (0)	00000	256/100	320,1/140,1	***0/0***	***64,65/40,1***
Channel Command/Status	CCSR	H (1)	L (0)	00010	260/104	324,5/144,5	***4/4***	***68,9/44,5***
Channel Control	CCR	H (1)	L (0)	00011	262/106	326,7/146,7	***6/6***	***70,1/46,7***
Channel Mode	CMR	H (1)	L (0)	00001	258/102	322,3/142,3	***2/2***	***66,7/42,3***
Clear DMA Interrupt	CDIR	L (0)	x	01101	26/1A	90,1/5A,B	26/1A	90,1/5A,B
Clock Mode Control	CMCR	H (1)	L (0)	01000	272/110	336,7/150,1	***16/10***	***80,1/50,1***
Daisy-Chain Control	DCCR	H (1)	L (0)	01101	282/11A	346,7/15A,B	***26/1A***	***90,1/5A,B***
DMA Array Count	DACR	L (0)	x	00100	8/8	72,3/48,9	8/8	72,3/48,9
DMA Command/Address	DCAR	L (0)	x	00000	0/0	64,5/40,1	0/0	64,5/40,1
DMA Control	DCR	L (0)	x	00011	6/6	70,1/46,7	6/6	70,1/46,7
DMA Interrupt Control	DICR	L (0)	x	01100	24/18	88,9/58,9	24/18	88,9/58,9
DMA Interrupt Vector	DIVR	L (0)	x	01010	20/14	84,5/54,5	20/14	84,5/54,5
Hardware Configuration	HCR	H (1)	L (0)	01001	274/112	338,9/152,3	***18/12***	***82,3/52,3***
Input/Output Control	IOCR	H (1)	L (0)	01011	278/116	342,3/156,7	***22/16***	***86,7/56,7***
Interrupt Control	ICR	H (1)	L (0)	01100	280/118	344,5/158,9	***24/18***	***88,9/58,9***
Interrupt Vector	IVR	H (1)	L (0)	01010	276/114	340,1/154,5	***20/14***	***84,5/54,5***
Misc. Interrupt Status	MISR	H (1)	L (0)	01110	284/11C	348,9/15C,D	***28/1C***	***92,3/5C,D***
Next Receive Address (L)	NRARL	L (0)	H (1)	11110	188/BC	252,3/FC,D	188/BC	252,3/FC,D
Next Receive Address (U)	NRARU	L (0)	H (1)	11111	190/BE	254,4/FE,F	190/BE	254,5/FE,F
Next Receive Byte Count	NRBCR	L (0)	H (1)	11101	186/BA	250,1/FA,B	186/BA	250,1/FA,B
Next Transmit Address (L)	NTARL	L (0)	L (0)	11110	60/3C	124,5/7C,D	60/3C	124,5/7C,D
Next Transmit Address (U)	NTARU	L (0)	L (0)	11111	62/3E	126,7/7E,F	62/3E	126,7/7E,F
Next Transmit Byte Count	NTBCR	L (0)	L (0)	11101	58/3A	122,3/7A,B	58/3A	122,3/7A,B
Port Control	PCR	H (1)	L (0)	00101	266/10A	330,1/14A,B	***10/0A***	***74,5/4A,B***
Port Status	PSR	H (1)	L (0)	00100	264/108	328,9/148,9	***8/8***	***72,3/48,9***
Receive Address (Lower)	RARL	L (0)	H (1)	10110	172/AC	236,7/EC,D	172/AC	236,7/EC,D
Receive Address (Upper)	RARU	L (0)	H (1)	10111	174/AE	238,9/EE,F	174/AE	238,9/EE,F
Receive Byte Count	RBCR	L (0)	H (1)	10101	170/AA	234,5/EA,B	170/AA	234,5/EA,B
Receive Character Count	RCCR	H (1)	L (0)	10110	300/12C	364,5/16C,D	***44/2C***	***108,9/6C,D***

2.9.9 Serial Data Registers RDR and TDR (Continued)

Table 2-2. IUSC Registers, in Alphabetical Order (Continued)

Register Name	Acronym	S//D	D//C	Reg Addr	Direct Address: 16-Bit Data	Direct Address: 8-Bit Data	DCAR7-0 or CCAR6-0: 16-Bit Data	DCAR7-0 or CCAR6-0: 8-Bit Data
Receive Command/Status	RCSR	H (1)	L (0)	10010	292/124	356,7/164,5	***36/24***	***100,1/64,5***
Receive Count Limit	RCLR	H (1)	L (0)	10101	298/12A	362,3/16A,B	***42/2A***	***106,7/6A,B***
Receive Data	RDR	H (1)	L (0)	1x000	288/120	352,3/260,1	***32/20***	***96/60***
(Read only; TDR for Write)			or H (1)	xxxxx	384-511	384-511	***xxx***	***xxx***
Receive DMA Interrupt Arm	RDIAR	L (0)	H (1)	01111	158/9E	222,3/DE,F	158/9E	222,3/DE,F
Receive DMA Mode	RDMR	L (0)	H (1)	00001	130/82	194,5/C2,3	130/82	194,5/C2,3
Receive Interrupt Control	RICR	H (1)	L (0)	10011	294/126	358,9/166,7	***38/26***	***102,3/66,7***
Receive Mode	RMR	H (1)	L (0)	10001	290/122	354,5/162,3	***34/22***	***98,9/62,3***
Receive Sync	RSR	H (1)	L (0)	10100	296/128	360,1/168,9	***40/28***	***104,5/68,9***
Set DMA Interrupt	SDIR	L (0)	x	01110	28/1C	92,3/5C,D	28/1C	92,3/5C,D
Status Interrupt Control	SICR	H (1)	L (0)	01111	286/11E	350,1/15E,F	***30/1E***	***94,5/5E,F***
Test Mode Control	TMCR	H (1)	L (0)	00111	270/10E	334,5/14E,F	***14/0E***	***78,9/4E,F***
Test Mode Data	TMDR	H (1)	L (0)	00110	268/10C	332,3/14C,D	***12/0C***	***76,7/4C,D***
Time Constant 0	TC0R	H (1)	L (0)	10111	302/12E	366,7/16E,F	***46/2E***	***110,1/6E,F***
Time Constant1	TC1R	H (1)	L (0)	11111	318/13E	382,3/17E,F	***62/3E***	***126,7/7E,F***
Transmit Address (Lower)	TARL	L (0)	L (0)	10110	44/2C	108,9/6C,D	44/2C	108,9/6C,D
Transmit Address (Upper)	TARU	L (0)	L (0)	10111	46/2E	110,1/6E,F	46/2E	110,1/6E,F
Transmit Byte Count	TBCR	L (0)	L (0)	10101	42/2A	106,7/6A,B	42/2A	106,7/6A,B
Transmit Character Count	TCCR	H (1)	L (0)	11110	316/13C	380,1/17C,D	***60/3C***	***124,5/7C,D***
Transmit Command/Status	TCSR	H (1)	L (0)	11010	308/134	372,3/174,5	***52/34***	***116,7/74,5***
Transmit Count Limit	TCLR	H (1)	L (0)	11101	314/13A	378,9/17A,B	***58/3A***	***122,3/7A,B***
Transmit Data	TDR	H (1)	L (0)	1x000	304/130	368,9/170,1	***48/30***	***112/70***
(Write only; RDR for Read)			or H (1)	xxxxx	384-511	384-511	***xxx***	***xxx***
Transmit DMA Interrupt Arm	TDIAR	L (0)	L (0)	01111	30/1E	94,5/5E,F	30/1E	94,5/5E,F
Transmit DMA Mode	TDMR	L (0)	L (0)	00001	2/2	66,7/42,3	2/2	66,7/42,3
Transmit Interrupt Control	TICR	H (1)	L (0)	11011	310/136	374,5/176,7	***54/36***	***118,9/76,7***
Transmit Mode	TMR	H (1)	L (0)	11001	306/132	370,1/172,3	***50/32***	***114,5/72,3***
Transmit Sync	TSR	H (1)	L (0)	11100	312/138	376,7/178,9	***56/38***	***120,1/78,9***

2.10 BYTE ORDERING

Various microprocessors differ on the correspondence between addresses and how bytes are arranged within a 16-bit or 32-bit value. The Zilog Z80 family and most Intel processors use what is 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 Z16C0x and Motorola 680x0 processors are "Big-Endian": they store and fetch the MS byte in the lowest-addressed byte, and the LS byte in the highest address.

The Z16C32 includes two separate control facilities that allow it to be used with either kind of processor. The "Select D15-D8 First" and "Select D7-D0 First" commands in the RTCmd field of the Channel Command/Address Register (CCAR15-11) control the byte ordering within a 16-bit transfer of serial data, and apply to DMA and processor accesses to RDR and TDR. These commands also control which data lines the Transmit DMA channel takes byte data from on a 16-bit bus. The ALBVO bit in the DMA Control Register (DCR12) controls how the DMA channels fetch buffer addresses and lengths from memory when operating in "Array" or "Linked List" mode. The following table summarizes how these bits should be programmed for various system configurations:

Table 2-3. System Configuration Programming

Bus Size	Processor Type	Programming
8 Bits	Big-Endian	16-bit (BCR2) := 0 ALBVO (DCR12) := 1
8 Bits	Little-Endian	16-bit (BCR2) := 0 ALBVO (DCR12) := 0
16 Bits	Big-Endian	16-bit (BCR2) := 1 ALBVO (DCR12) := 1 RTCmd (CCAR15-11) := "Select D15-D8 First"
16 Bits	Little-Endian	16-bit (BCR2) := 1 ALBVO (DCR12) := 0 RTCmd (CCAR15-11) := "Select D7-D0 First"

2.11 REGISTER READ AND WRITE CYCLES

Figures 2-13 through 2-16 show the waveforms of the signals involved when the host processor reads or writes an IUSC register. Separate drawings are included for the signaling on a bus with multiplexed addresses and data, and for a bus with separate address and data lines. 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 data strobe, have been combined by labeling 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 readers that its state does not matter with /RD and /WR.

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

3. The difference between a sampled (address-like) /INTACK signal, and one that is a strobe, is handled by showing it "maybe or maybe not" going Low after the address-sampling time, again with appropriate labeling.

Chapter 6 covers details of DMA cycles initiated by the IUSC as the bus master, while Chapter 7 covers interrupt acknowledge cycles.

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

2.11 REGISTER READ AND WRITE CYCLES (Continued)

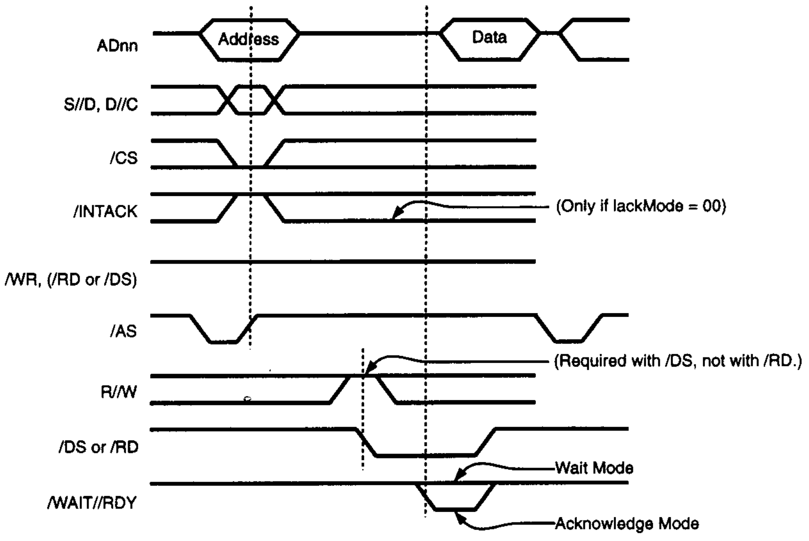


Figure 2-13. Register Read Cycle with Multiplexed Addresses and Data

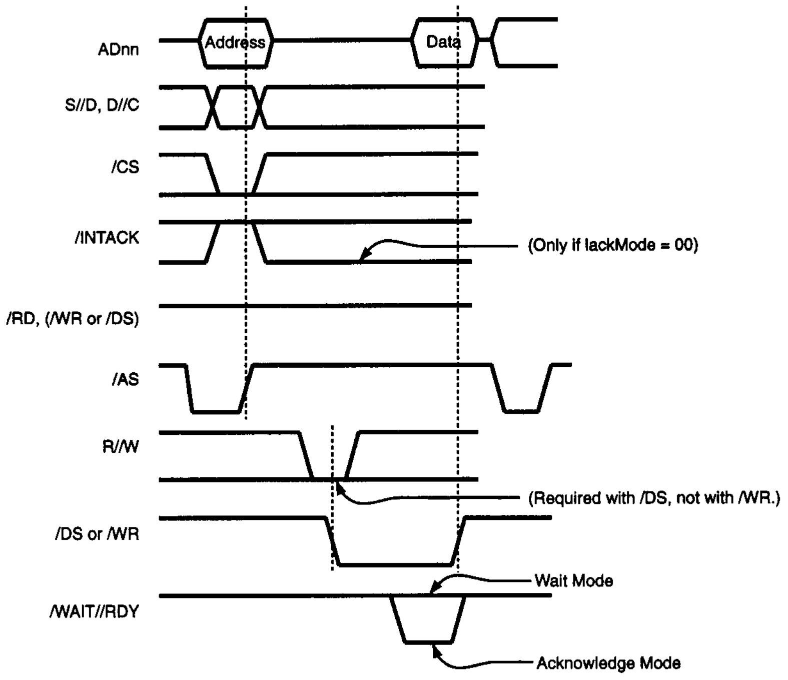


Figure 2-14. Register Write Cycle with Multiplexed Addresses and Data

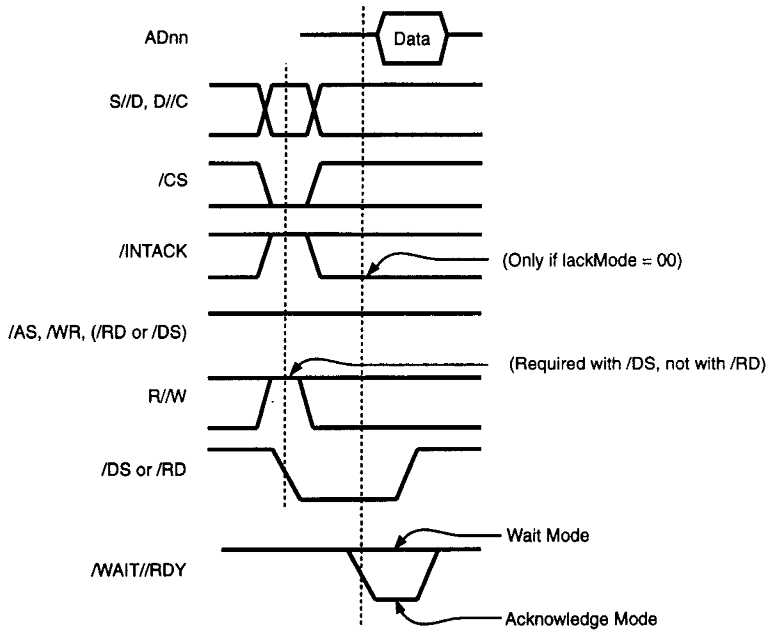


Figure 2-15. Register Read Cycle with Non-Multiplexed Data Lines

2.11 REGISTER READ AND WRITE CYCLES (Continued)

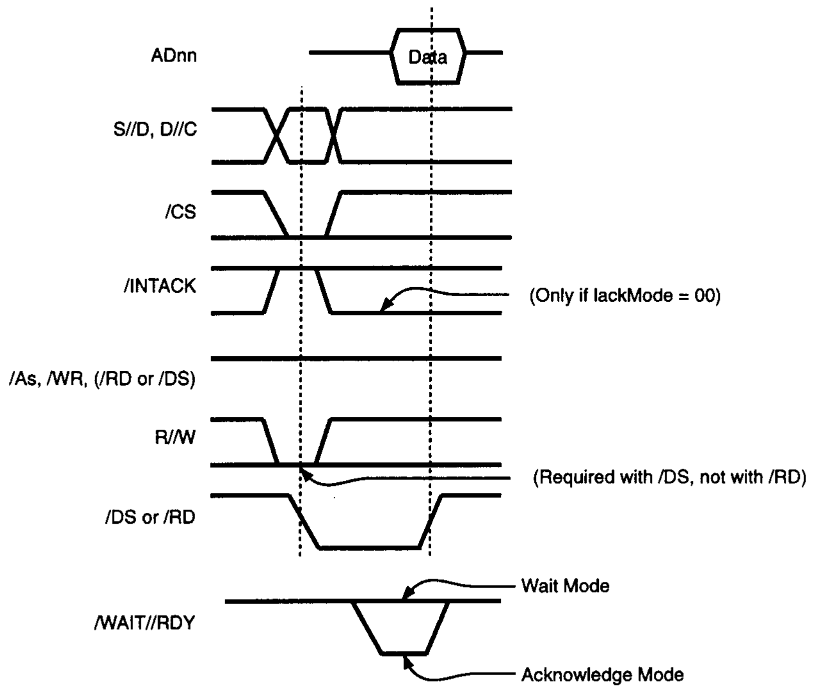


Figure 2-16. Register Write Cycle with Non-Multiplexed Data Lines

2.12 DMA CYCLE OPTIONS

Three bits in the DMA Control Register (DCR) affect how the IUSC operates as a bus master—that is, how it acts when it has control of the bus. This information is presented both here and in Chapter 6.

2.12.1 S//D, D//C Status Output

The **DCSDOut** bit (DCR4) controls whether the IUSC drives the S//D and D//C pins when it is the bus master. If DCSDOut is 1, the IUSC drives S//D Low for Tx channel operations and High for Rx channel cycles, and drives D//C High during transfers of serial data and Low during array or linked-list fetching. When this bit is 1, on a multiplexed bus S//D and D//C cannot be connected directly to any of the address/data lines with the AD pins, and external drive on the S//D and D//C pins must be tri-stated (released) while the IUSC is the bus master.

If neither external logic nor monitoring equipment (like a logic state analyzer) has any use for the information described above, software can program DCSDOut as 0. In this case the IUSC never drives S//D and D//C, and these pins can be connected directly to AD lines on a multiplexed bus, or can be driven full-time by external drivers on a non-multiplexed bus.

2.12.2 Wait Insertion

If the **1Wait** bit (DCR3) is 1, the IUSC extends the data portion of each master bus cycle by one CLK period. This allows use of slower memories for a given CLK frequency, or use of a faster CLK frequency with a particular memory type. Signaling on /WAIT//RDY can be used to extend master bus cycles, regardless of the state of this bit. When 1Wait is 1 the IUSC starts actively sampling /WAIT//RDY one CLK period later than when it is 0.

2.12.3 /UAS Frequency

Since the DMA channels maintain 32-bit addresses but have only a 16-bit external bus, they present each address in two parts. They signal the availability of the more significant half of an address by driving /UAS Low, and signal that the LS half of an address is on the AD lines by driving /AS Low. The **UASAll** bit (DCR2) controls how often the channels present the more-significant half of the address. If UASAll is 1, every master bus cycle includes presentation of the more-significant half of the address on the AD15-AD0 pins, with a Low-going pulse on /UAS. This means that every bus cycle takes at least four cycles of CLK.

If UASAll is 0, the IUSC includes a /UAS sequence only in cycles that meet one or more of the following criteria:

1. in the first cycle after taking control of the bus from another master,
2. in the first cycle after switching from one channel to the other,
3. in Pipelined mode, in the first cycle after switching from one buffer to the next,
4. for a channel in Array or Linked List mode, in the first cycle after switching from data buffer accesses to array/list accesses, or vice-versa,
5. in the first cycle after incrementing a memory address results in a carry from A15 to A16, even if the AddrSeg field (DCR1-DCR0) is 10 so that the carry is blocked.

When the IUSC includes a /UAS sequence in a bus cycle, the minimum length of the bus cycle is 4 CLK periods, while if it does not, the bus cycle can be as short as 3 CLKs.

UASAll should be programmed as 1 only if required by unusual external hardware. For example, if the IUSC and another bus master share an upper-address latch and the other bus master can insert cycles between IUSC cycles within the same bus grant, UASAll would want to be 1.

[illegible] 1

Bus Interfacing 2

Sample Application 3

Serial Interfacing 4

Serial Modes and Protocols 5

Direct Memory Access (DMA) Channels 6

Interrupts 7

CHAPTER 3
Z16C32 IUSC™ SAMPLE APPLICATION

3.1 INTRODUCTION

Figure 3-1 shows a sample application of two IUSCs interfaced to an ISA (AT) personal computer bus. The design includes two 128K x 8 static RAMs that can be read or written by either the host (80x86) processor or by the IUSCs. This example is a simplified version of a commercial-quality design that is available for licensing from Zilog.

3.2 I/O AND MEMORY SPACE ADDRESSING

The design presents a block of four registers in I/O space, two of which are implemented in this example. The J1 jumper header allows this 4-byte block to be located anywhere in the upper 512 bytes of the 1 Kbyte I/O space of the original PC/XT architecture. Figure 3-2 shows the (simplified) structure of this 4-byte block.

If software writes to the highest of these four I/O addresses repeatedly, the write pulses will pump down an RC circuit (R1, C1) that also provides a power-on reset.

Writing to the second of the four I/O addresses allows software to set the location of the RAM and IUSCs in memory space, as well as selecting which ISA bus interrupt level the board should use. On the ISA bus, the design presents a 16 Kbyte "window" into the 256 Kbytes of RAM on the card. Writing to the first of the four I/O addresses allows software to set the base of this window within the RAM on the card.

When the base of the window within the RAM is set to zero, the first 512 bytes of the 16K window refer not to RAM but to the registers in the IUSCs, with U1 ("IUSC 0") occupying addresses 0-255 and U2 ("IUSC 1") occupying 256-511.

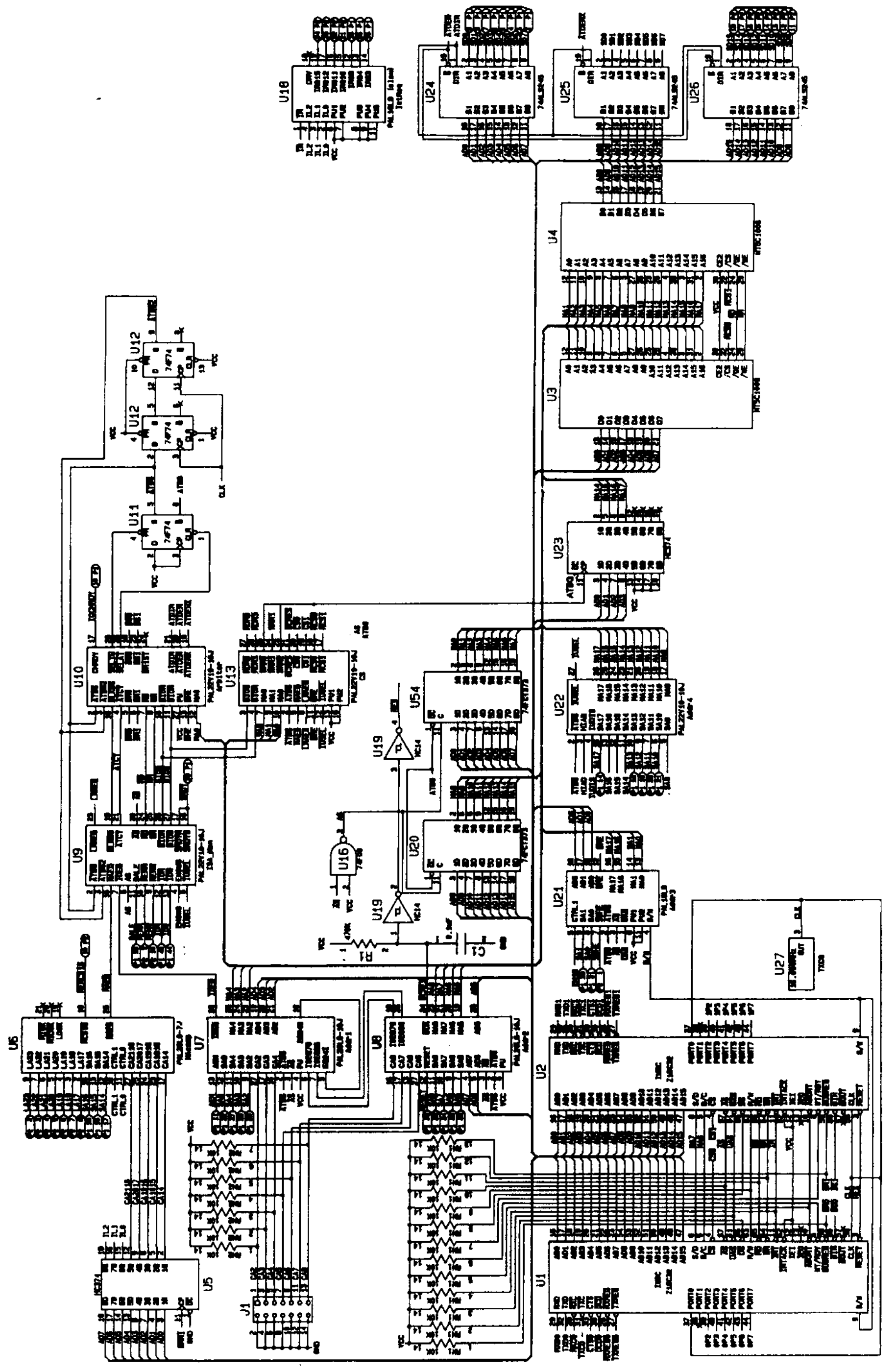


Figure 3-1. Sample IUSC Application Schematic

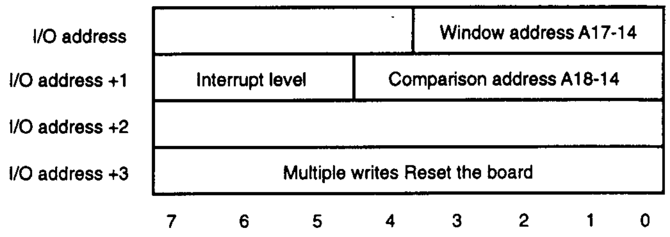


Figure 3-2. Register Map in I/O Space

3.3 HOST ADDRESS HANDLING

Proceeding from the upper-left corner of Figure 3-1, the HC374 "U5" implements the second of the four I/O space registers, including the Comparison Address signals CA18-CA14 that determine where the board resides within the second 512 Kbytes of the first 1 Mbyte of the 16 Mbyte memory space of the ISA (AT) bus. The three MSBits of this register hold the Interrupt Level code IL2-IL0, that controls on which of seven ISA bus interrupt levels the U18 Interrupt Requester device requests interrupts.

The actual memory address comparison is performed by the U6 Memory Address Comparator, which drives its combinatorial output "/MAEQ" Low when the address on the "LA" and high-order "SA" lines of the ISA bus matches the CA18-CA14 value. In addition, the "MAComp" device drives the /MEMCS16 line of the ISA bus low whenever the address matches, to signify that it's a 16-bit device. The equations for U6 are shown in Table 3-1.

Table 3-1. Logic Equations for U6 "MAComp"

/HIOK	=	/LA23*/LA22*/LA21*/LA20*LA19*/LA18*/CA2118*/LA17*/CA2017	
	+	/LA23*/LA22*/LA21*/LA20* LA19*/LA18*/CA2118* LA17* CA2017	
	+	/LA23*/LA22*/LA21*/LA20* LA19* LA18* CA2118*/LA17*/CA2017	
	+	/LA23*/LA22*/LA21*/LA20* LA19* LA18* CA2118* LA17* CA2017	
/LONOK	=	/CTRL1*/SA16* CA1916	
	+	/CTRL1* SA16*/CA1916	
	+	/CTRL1*/SA15* CA1815	
	+	/CTRL1* SA15*/CA1815	
	+	/CTRL1*/SA14* CA14	
	+	/CTRL1* SA14*/CA14	
/MAEQ	=	/HIOK* LONOK	
/MCS16	=	1	; open-collector/open-drain
MCS16.TRST	=	/HIOK* LONOK	; drive when equal

Note:
In the logic equations in this chapter, "/" represents logical negation, "*" presents logical And, and "+" presents logical Inclusive OR.

The J1 jumper header plus its pull-up resistors source the Comparison Address signals CA8-CA2. U7 "Addr1" and U8 "Addr2" compare CA8-CA2 to SA8-SA2 on the ISA bus, driving the /IOEQ line low whenever these lines match and SA9 is High. U7-U8 also act as buffers for the low-order SA lines, driving the address on SA8-SA2 onto the RAM address lines MA8-MA2 whenever the /ATBG signal is Low, signifying that the host on the ISA (AT) bus is in control of the RAMs.

A third function of U7-U8 is to multiplex the address from SA5-SA2 onto the AD5-AD2 lines of the IUSCs, whenever /ATBG is low and the Address Strobe line (/AS) is low. The equations for U7 and U8 are shown in Tables 3-2 and 3-3 respectively.

3.3 HOST ADDRESS HANDLING (Continued)

Table 3-2. Logic Equations for U7 "Addr1"

```
GROUP MAS MA4 MA3 MA2
GROUP ADS AD4 AD3 AD2
; drive addresses from AT bus onto MA lines
            /MA2  =  /SA2
            /MA3  =  /SA3
            /MA4  =  /SA4
        MAS.TRST  =  /ATBG
; multiplex the addresses from the AT bus onto the AD lines
;  for register addressing in the IUSCs
            /AD2  =  /SA2
            /AD3  =  /SA3
            /AD4  =  /SA4
        ADS.TRST  =  /ATBG*/AS                ; drive SA to AD during Address Strobe
; match the I/O address on the SA lines
        /IOEQ24O  =  SA2*/CA2 + /SA2*CA2
                  +  SA3*/CA3 + /SA3*CA3
                  +  SA4*/CA4 + /SA4*CA4
; IOEQ is low-active
           /IOEQ  =  IOEQ24I*IOEQ56*IOEQ78*SA9*/AEN
```

Note:
In the logic equations in this chapter, "/" represents logical negation, "*" presents logical And, and "+" presents logical Inclusive OR.

Table 3-3. Logic Equations for U8 "Addr2"

```
GROUP MAS MA8 MA7 MA6 MA5
; drive addresses from AT bus onto MA lines
            /MA5  =  /SA5
            /MA6  =  /SA6
            /MA7  =  /SA7
            /MA8  =  /SA8
        MAS.TRST  =  /ATBG
; drive address bit 5 to the IUSCs during address strobe
            /AD5  =  /SA5
        AD5.TRST  =  /ATBG*/AS
; match the I/O address on the SA lines
         /IOEQ56  =  SA5*/CA5 + /SA5*CA5
                  +  SA6*/CA6 + /SA6*CA6
         /IOEQ78  =  SA7*/CA7 + /SA7*CA7
                  +  SA8*/CA8 + /SA8*CA8
; open-collector driver for Reset
          /OCRES  =  1
      OCRES.TRST  =  RESET
```

Note:
In the logic equations in this chapter, "/" represents logical negation, "*" presents logical And, and "+" presents logical Inclusive OR.

U21 "Addr3" handles the low order addresses similarly to U7-U8: when /ATBG is low it drives SA1-SA0 onto MA1-MA0, and when /ATBG and /AS are both low it drives SA1-0 onto AD1-AD0. In the latter case it also synthesizes a value for AD6 from the SA0 and /SBHE lines. The IUSC uses AD6 to determine whether a register access is to a full 16-bit register or only to one byte.

U21 also multiplexes /SBHE or B//W from the IUSCs, onto the /BHE signal depending on the state of /ATBG. The equations for U21 are shown in Table 3-4.

Table 3-4. Logic Equations for U21 "Addr3"

```
GROUP ADS AD0 AD1 AD6
GROUP LOMAS MA0 MA1
GROUP HIMAS MA16 MA17
; drive addresses from the AT bus onto the MA lines
;  NO LONGER latch A1-A0 from the IUSCs !!!!
            /MA0  =  /SA0
            /MA1  =  /SA1
     LOMAS.TRST  =  /ATBG
; latch MS bits of IUSC addresses and drive onto MA lines
           /MA16  =  /AD0 */UAS*PU2        ; capture MS address from IUSC
                  +  /AD0 */MA16           ; deglitch term
                  +  /MA16* UAS*PU2        ; hold high address from IUSC
           /MA17  =  /AD1 */UAS*PU2
                  +  /AD1 */MA17           ; deglitch term
                  +  /MA17* UAS*PU2
     HIMAS.TRST  =  ATBG                   ; drive high address when IUSC controlling bus
; drive LS bits of AT bus addresses back to IUSCs for register addressing
            /AD0  =  /SA0
            /AD1  =  /SA1
; drive AD6 of register addresses with B/W indication
            /AD6  =  /SA0*/SBHE            ; else high for byte operation
       ADS.TRST  =  /ATBG*/AS              ; drive onto AD lines during AS (from ISA_Mon)
; convert from B/W style of IUSCs to BHE style
            /BHE  =  /ATBG*/SBHE           ; from AT bus
                  +  ATBG* MA0             ; odd byte op by IUSC
                  +   ATBG*B/W             ; word op by IUSC
```

Note:
In the logic equations in this chapter, "/" represents logical negation, "*" presents logical And, and "+" presents logical Inclusive OR.

3.3 HOST ADDRESS HANDLING (Continued)

The last of the four address-handling PLDs is U22 "Addr4". It drives SA17-SA9 onto MA17-MA9 when /ATBG is low, as well as decoding when these lines are all zero, indicating an access to an IUSC rather than RAM, and driving /IUSEL accordingly. The equations for U22 are Table 3-5.

The HC374 U23 implements the lowest addressed register of the four I/O space locations, capturing the LS 4 bits that software writes to this location, and presenting them on MA17-MA14 thereafter whenever /ATBG is Low.

Table 3-5. Logic Equations for U22 "Addr4"

```
GROUP LOMAS MA13 MA12 MA11 MA10 MA9
; drive addresses from the AT bus onto the MA lines
            /MA9  =  /SA9
            /MA10 =  /SA10
            /MA11 =  /SA11
            /MA12 =  /SA12
            /MA13 =  /SA13
      LOMAS.TRST  =  /ATBG                        ; drive lo ads whenever AT controlling bus
; decode whether the cycle is targeted for the IUSCs
          /IUSEL  =  /SA9*/SA10*/SA11*/SA12*/SA13*/MA14*/MA15*/MA16*/MA17
```

Note:
In the logic equations in this chapter, "/" represents logical negation, "*" presents logical And, and "+" presents logical Inclusive OR.

3.4 IUSC ADDRESS HANDLING

The final function of U21 is to capture the state of AD1-AD0 whenever an IUSC drives /UAS low, and to present it thereafter on MA17-MA16.

The 74FCT373 latches U20 and U54 similarly capture the LS 16 bits of the address when an IUSC drives /AS low, and present this value on MA15-MA0 thereafter.

3.5 BUS MONITORING

U9 "ISA_Mon" monitors the outputs of the host address handlers described above, namely /MAEQ, /IOEQ, and /IUSEL, plus the ISA (AT) bus control signals BALE, /MEMR, /MEMW, /IOR, and /IOW, and the arbiter outputs /ATBG and /ATBG2. From these signals it derives the following outputs:

/LMAEQ. A latched version of /MAEQ (which follows the LA lines and thus is not well-synchronized to data transfer cycles). /LMAEQ is low when the host address falls within the board's memory address range.

/ATCY. Low whenever the host is trying to access a board-level register, RAM, or an IUSC on this board.

/AS. Pulsed low when a host access to RAM or an IUSC is beginning, to strobe the register address and chip selection into the IUSCs.

/RD. Low-active output enable for the RAMs and IUSCs.

/WR. Low-active write strobe for the RAMs and IUSCs.

/BIOR. Low-active register read strobe.

/BIOW. Low-active register write strobe.

/BLKBG. A signal that blocks bus granting to the IUSCs by the Arbiter logic, if/while a cycle from the host looks like it may be targeted for this board.

The equations for these outputs are shown in Table 3-6.

Table 3-6. Logic Equations for U9 "ISA_Mon"

```
GROUP RW RD WR
; latch MAEQ
          /LMAEQ  =  /MAEQ * BALE
                  +  /MAEQ */LMAEQ
                  +  /LMAEQ*/BALE
; block bus grants to IUSCs by Arbiter, if cycle may be starting
          /BLKBG  =  /MAEQ*BALE                        ; start if memory address equal
                  +  /IOEQ*BALE                        ; start if I/O address equal
; mod 1/4/93, use LMAEQ because Compaq 486/33M doesn't "wait" LA23-17
; further modified 1/4/94, to improve IUSC access to RAM and prevent
; host locking up the RAM during a block move or compare (on some platforms)
                  +  /BLKBG*/BALE*/LMAEQ*ATCY* IOR*IOW
                  +  /BLKBG*/BALE*/IOEQ*ATCY*MEMR*MEMW
; request from the bus
           /ATCY  =  /LMAEQ*/MEMR                      ; memory read
                  +  /LMAEQ*/MEMW                      ; memory write
                  +  /IOEQ */IOR                       ; I/O read
                  +  /IOEQ */IOW                       ; I/O write
; make address strobe for cycle to IUSC
; mod 1/4/93, use LMAEQ because Compaq 486/33M doesn't "wait" LA23-17
             /AS  =  ATBG2*/LMAEQ                      ; for first cycle of AT grant
                  +  BALE */MAEQ                       ; for subsequent one
         AS.TRST  =  /ATBG                             ; drive /AS when AT in control
; drive RD and WR for memory space cycles
             /RD  =  /LMAEQ */MEMR*/ATBG*/ATBG2*/PAS   ; memory read for us
             /WR  =  /LMAEQ */MEMW*/ATBG*/ATBG2*/PAS   ; memory write for us
         RW.TRST  =  /ATBG                             ; drive when AT bus grant
; fully decoded I/O read and write strobes
           /BIOR  =  /IOEQ*/IOR*/ATBG*/ATBG2*/PAS      ; I/O read for us
           /BIOW  =  /IOEQ*/IOW*/ATBG*/ATBG2*/PAS      ; I/O write for us
```

Note:
In the logic equations in this chapter, "/" represents logical negation, "*" presents logical And, and "+" presents logical Inclusive OR.

3.6 ARBITER LOGIC

This consists of the U10 "Arbiter" plus the three high-speed flip-flops to its right in Figure 3-1 (U11,12). The main function of these parts is to control when the host processor can access the RAM, board registers, and IUSCs, as opposed to when the IUSCs can access RAM.

U10 drives its /SELIU output low whenever one or both IUSCs is (are) requesting access to RAM via their /BUSREQ outputs, and the host is not requesting access to the board. Conversely, it drives /SELAT low when the host is requesting access, but neither IUSC is. These two outputs control the U11 flip-flop, which acts as an SR latch. Its outputs are in turn synchronized to the 16 MHz IUSC clock by the first stage of U12, and a one-clock-delayed version is provided by the second stage of U12.

Because the host and the IUSCs operate from independent clocks, pulses on /SELIU and /SELAT are subject to arbitrarily short "runt pulses", and thus the U11 flip-flop is subject to metastable states. Since U11 can switch based on the host's timing, the first stage of U12 is also subject to metastability, though less frequently than U11 is. The output of the second stage of U12 is not subject to metastability. U11 and U12 are 74F74 devices because these are fast and resolve metastability fairly quickly.

The first stage of U12 (ATBG and /ATBG) controls whether the address bus MA17-MA0, plus the various control signals for the RAM and IUSC, are controlled by the host bus or by the IUSCs. The second stage (/ATBG2) must match /ATBG before the control signals /RD, /WR, etc., are activated to begin a data transfer by the host.

A related function of U10 is to drive the ISA (AT) bus signal IOCHRDY low to "wait" the host processor, whenever it is trying to access RAM, an IUSC, or a board-level register, and this access can't be performed immediately because an IUSC is accessing RAM.

Similarly, U10 drives one of the /BIN inputs of the IUSCs whenever /ATBG is high and that IUSC is requesting bus access. If both IUSCs request access, U10 remembers which requested first via its (externally unused) output BR1ST.

Finally, U10 drives the ATDIR and /ATDEN signals that control the data transceivers U24 and U26, between the host data lines SD15-SD0 and the on-board address/data lines AD15-AD0. ATDIR and /ATDEN are generated in U10 only as a matter of convenience, and this logic is not related to U10's primary function of arbitration.

The equations for U10 are shown in Table 3-7.

Table 3-7. Logic Equations for U10 "Arbiter"

```
; switch ATBG FF toward IUSC's when IUSC request, no AT bus cycle for us
          /SELIU  =  /BR0*ATCY*BLKBG
                  +  /BR1*ATCY*BLKBG
; wait the host until bus is granted to it
        /IOCHRDY  =  ATBG                ; wait if bus is not granted to host
                  +  ATBG2               ; and wait for 1-2 clocks thereafter
    IOCHRDY.TRST  =  /ATCY               ; drive this signal whenever cycle is for us
; switch ATBG FF toward AT bus when AT bus cycle for us, no IUSC request
          /SELAT  =  BR0*BR1*/ATCY
; track IUSC BR's for first-come, first-served
          /BR1ST  =  /BR1  * BR0         ; remember BR1 without BR0
                  +  /BR1ST*/BR1         ; once asserted, keep it until IUSC drops RQ
; the above is subject to glitches and metastability, and is acceptable
;  only because the IUSCs sample their BG (/BIN) lines twice, two clocks (125 ns) apart
; grants to the IUSC's
            /BG0  =  /BR0* BR1ST*ATBG    ; request from 0, none from 1
            /BG1  =  /BR1*/BR1ST*ATBG    ; request from 1 was first
; control signals for thé data transceivers
MINIMIZE_OFF
          /ATDIR  =  /RD
                  +  /BIOR
                  +  /ATDIR*/ATDEN       ; anti-backlash
MINIMIZE_ON
          /ATDEN  =  /ATBG*/RD*/MA0
                  +  /ATBG*/RD*/BHE
                  +  /ATBG*/WR*/MA0
                  +  /ATBG*/WR*/BHE
                  +  /BIOR
                  +  /BIOW
```

3.7 CHIP SELECT DECODING

U13 "CS" decodes the outputs of the various devices described in earlier sections, including some of the MA lines, to provide write strobes for the on-board registers and chip selects for the RAMs and IUSCs. It also pulses the /OCRES line low in an open-drain fashion when software writes the highest-addressed of the four register locations the board presents on the ISA (AT) bus, which allows a software reset via repeated writes to this location. The equations for U13 are shown in Table 3-8.

Table 3-8. Logic Equations for U13 "CS"

; chip selects for IUSCs			
/CS0	=	/ATBG*/LMAEQ*/IUSEL*/MA8	; A23-18(or 14) match, A17-9 zero,
			; not IUDIS, A8=0
/CS1	=	/ATBG*/LMAEQ*/IUSEL* MA8	; A23-18(or 14) match, A17-9 zero,
			; not IUDIS, A8=1
; chip selects for RAM			
/RAMCS0	=	ATBG*/MA0	; IUSC in control, even address
	+	/ATBG*/LMAEQ*IUSEL*/MA0	; AT bus control, not IUSC, even ad
/RAMCS1	=	ATBG*/BHE	; IUSC in control, word or odd byte
	+	/ATBG*/LMAEQ*IUSEL*/BHE	; AT ctrl, not IUSC, word or odd byte
; write strobes for the control registers			
/WRR0	=	/BIOW*/MA0*/MA1	; reg 0
/WRR1	=	/BIOW* MA0*/MA1	
; write to reg 3 pulses the RC-Reset circuit			
; more than one such write is needed to make a Reset			
/OCRES	=	1	; open-drain
OCRES.TRST	=	/BIOW* MA0* MA1	; drive it on write reg 3

3.8 IUSC HOOKUP

The IUSCs U1 and U2 are wired in parallel for most of their "bus side" lines, but the exceptions are significant.

The "serial side" pins are shown on the "outside" edges of the IUSCs, simply as signal names which can be utilized for the serial interfaces, on a second page of this schematic that is quite application-specific and so is not provided here. The one serial-side signal that is connected is PORT0/CLK0, which is connected to the same 16 MHz clock that's used for the DMA clock, so that it can be used for baud-rate generation.

(B/W is a bus-side output signal that ended up on the serial side of the IUSC symbol for obscure historical reasons.)

MA7,6 are connected to the S/D and D/C inputs. This would have had to be MA8,7 except for the synthesis of AD6 as a byte/word indication by U21 (Addr3) as described above. As it is, the DMA channel registers for U1 are addressed as 0-127, the serial controller part of U1 is addressed as 128-255, the U2 DMA channel registers are addressed as 256-383, and the U2 serial controller is at 384-511. These ranges are half the size of those described in Chapter 2.

The /AS, /RD, and /WR pins of the two IUSCs are wired in parallel, and a pull-up resistor is attached to each signal to keep it at a valid logic high when the arbitration logic is maintaining /ATBG high because an IUSC accessed RAM more recently than the host accessed RAM, an IUSC, or a boardlevel register, but said IUSC is no longer using the bus nor driving these lines.

The /UAS inputs are similarly wired in parallel, and are pulled up to keep them at a valid logic high when neither IUSC is actively using the bus.

The /DS inputs are pulled up separately. If they were connected together, when one IUSC was operating it would drive both /DS and either /RD or /WR low, which would make the other IUSC go into a "pre-reset" state in which it would not operate at all, until it was Reset. The pull-up resistors ensure a valid logic high when the IUSC is not using the bus.

The /INT pins are connected together and to a pull-up resistor, and to the Interrupt Requestor U18. More on this in the next section.

The R/W pins are connected together and to a pull-up. The host interface logic uses /RD and /WR to communicate with the IUSCs, not /DS and R/W. The pull-up is included because floating inputs on a CMOS device tend to increase noise inside the device.

The /INTACK and /IEI pins are wired directly to V_{CC}. Interrupt acknowledge cycles are not visible to add-in cards on the ISA (AT) bus.

The Abort pins are pulled up. The Hardware Abort facility of the IUSCs is not used in this design.

The /WT/RDY pins of the IUSC are pulled up separately. Each IUSC drives its /WT/RDY pin full-time (totem-pole) except when it's actively using the bus, at which time the pull-up assures a valid logic high. The RAMs are fast enough to operate without wait states to the IUSCs.

The /BUSREQ outputs and /BIN inputs of both IUSCs are separately connected to the Arbiter logic as described in an earlier section. The /BOUT outputs are not used.

The /RESET inputs of both IUSCs are driven from the RC circuit described earlier, and are driven low on power up, when the ISA (AT) bus RESET line is activated, or when a series of writes to the highest-addressed board register location are performed by the software.

Both B/W outputs are wired together and connected to the U21 Addr3 device as described earlier. No pull-up is used because U21 masks this signal except when one of the IUSCs is actively using the bus.

3.9 INTERRUPTS

As noted above, the /INT outputs of the two IUSC are wired together and pulled up. The "open drain /INT" option should be programmed in the Bus Configuration Registers (BCRs) of both IUSCs during initialization. This "wire-ORed" request is then used by the U21 device IntReq, together with the level-select code that software programs into the U5 register, to request interrupt on one of seven of the ISA (AT) bus' IRQ lines.

The equations for U21 are shown in Table 3-9.

Table 3-9. Logic Equations for U21 "IntReq"

; interrupt the host on any of seven pins			
/IRQ3	=	IR	; invert of IR line
IRQ3.TRST	=	/IL2*/IL1* IL0	; only drive the selected pin
/IRQ4	=	IR	
IRQ4.TRST	=	/IL2* IL1*/IL0	
/IRQ9	=	IR	
IRQ9.TRST	=	/IL2* IL1* IL0	
/IRQ10	=	IR	
IRQ10.TRST	=	IL2*/IL1*/IL0	
/IRQ11	=	IR	
IRQ11.TRST	=	IL2*/IL1* IL0	
/IRQ12	=	IR	
IRQ12.TRST	=	IL2* IL1*/IL0	
/IRQ15	=	IR	
IRQ15.TRST	=	IL2* IL1* IL0	

CHAPTER 4
Z16C32 IUSC™ SERIAL INTERFACING

4.1 INTRODUCTION

The IUSC includes several serial interface options and features that promote its usefulness in many different kinds of applications. It allows a variety of clocking schemes, and will do serial encoding and decoding for Non-Return-to-Zero-Inverting (NRZI) and Biphase formats that carry clocking information with the serial data. The IUSC further supports such decoding with an on-chip Digital Phase Locked Loop circuit. It also provides specialized and general purpose 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 are not related to the serial link at all. Finally, for time-division-multiplexed links such as ISDN and Fractional T1 circuits, the IUSC includes separate Time Slot Assigner modules for the Receiver and Transmitter. Each "TSA" restricts active operation to a programmable time window within a cyclic time-multiplexed data stream.

4.2 SERIAL INTERFACE PIN DESCRIPTIONS

RxD. *Received Data* (input, positive logic). The serial input.

TxD. *Transmit Data* (output, positive logic). The serial output.

/RxC. *Receive Clock* (input or output). This signal can be used as a clock input for any of the functional blocks in the serial controller, or software can program the IUSC 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.

/TxC. *Transmit Clock* (input or output). This signal can be used as a clock input for any of the functional blocks in the serial controller, or software can program the IUSC 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.

/RxREQ. *Receive DMA Request* (input or output). In device testing or in applications not using the serial and DMA controller sections together in the usual way, this pin can carry a low-active DMA Request from the receive FIFO. On the IUSC this request is internally routed to the on-chip Receive DMA channel, and it is more typical to use this pin as a general-purpose input or output or as an interrupt input.

/TxREQ. *Transmit DMA Request* (input or output). In device testing or in applications not using the serial and DMA controller sections together in the usual way, this pin can carry a low-active DMA Request from the transmit FIFO. On the IUSC this request is internally routed to the on-chip Transmit DMA channel, and it is more typical to use this pin as a general-purpose input or output or as an interrupt input.

/DCD. *Data Carrier Detect* (input or output, active low). Software can program the IUSC so that this signal enables/disables the receiver. In addition or instead, software can program the device to request interrupts in response to transitions on this line. The pin can also be used as a simple input or output.

/CTS. *Clear to Send* (input or output, active low). Software can program the IUSC so that this signal enables/disables the transmitter. In addition or instead, software can program the device to request interrupts in response to transitions on this line. The pin can also be used as a simple input or output.

PORT7//TxComplete. *General-Purpose I/O or Transmit Complete* (input or output). Software can program the IUSC so that this pin is a general-purpose input or output, or so that it carries a Transmit Complete signal from the Transmitter, that can control an external driver on TxD. The IUSC captures transitions on this pin in internal latches, as described later in this chapter.

4.2 SERIAL INTERFACE PIN DESCRIPTIONS (Continued)

PORT6//FSYNC. *General-Purpose I/O or Frame Sync* (input or output). Software can program the IUSC so that this pin is a general-purpose input or output, or a Frame Sync input for the IUSC's Time Slot Assigner circuits. The IUSC captures transitions on this pin in internal latches, as described later in this chapter.

PORT5//RxSYNC. *General-Purpose I/O or Receive Sync* (input or output). Software can program the IUSC so that this pin is a general-purpose input or output, or so that it carries a Receive Sync output from the Receiver. The IUSC captures transitions on this pin in internal latches, as described later in this chapter.

PORT4//TxTSA. *General-Purpose I/O or Transmit Time Slot Assigner Gate* (input or output). Software can program the IUSC so that this pin is a general-purpose input or output, or so that it carries the Gate output of the Transmit Time Slot Assigner, that can enable an external TxD driver in time-slotted ISDN or Fractional T1 applications. The IUSC captures transitions on this pin in internal latches, as described later in this chapter.

PORT3//RxTSA. *General-Purpose I/O or Receive Time Slot Assigner Gate* (input or output). Software can program the IUSC so that this pin is a general-purpose input or output, or so that it carries the Gate output of the Receive Time Slot Assigner. The IUSC captures transitions on this pin in internal latches, as described later in this chapter.

PORT2//LTTxEnab. *General-Purpose I/O or LocalTalk Driver Enable* (input or output). Software can program the IUSC so that this pin is a general-purpose input or output, or so that it carries low-active enable for an external TxD driver in a LocalTalk (AppleTalk) application. The IUSC captures transitions on this pin in internal latches, as described later in this chapter.

PORT1-0/CLK1-0. *General-Purpose I/Os or Reference Clocks* (inputs or outputs). Software can program the IUSC so that either of these pins is a general-purpose input or output, or a clock for the Receiver and/or Transmitter. On the Z16C32, this clock can be used directly as a bit clock or divided down as a time base. When one of these pins is a general-purpose I/O, the IUSC captures transitions on it in internal latches, as described later in this chapter.

4.3 TRANSMIT AND RECEIVE CLOCKING

The IUSC's Receiver and Transmitter logic have separate internal clock signals, which we will call RxCLK and TxCLK. In most of the IUSC'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 4-1 shows how RxCLK and TxCLK can be derived in several different ways. This flexibility is an important part of the IUSC'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 IUSC to take RxCLK directly from the /RxC pin and TxCLK directly from the /TxC pin. When an IUSC 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 IUSC, rather than by means of offchip hardware. To provide for this, the IUSC includes various means by which high-speed clocking on one or more of the /RxC, /TxC. PORT1, or PORT0 pins can be divided down to almost any desired bit rate.

4.3.1 CTR0 and CTR1

Two separate 5-bit counters called CTR0 and CTR1 comprise the first stage of the IUSC's clock-generation logic. Figure 4-2 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, /TxC, PORT0, or PORT1 pin:

CTRnSRC	CTRn Clock Source
00	CTRn disabled
01	CTRn input = PORTn/CLKn pin
10	CTRn input = /RxC pin
11	CTRn input = /TxC pin

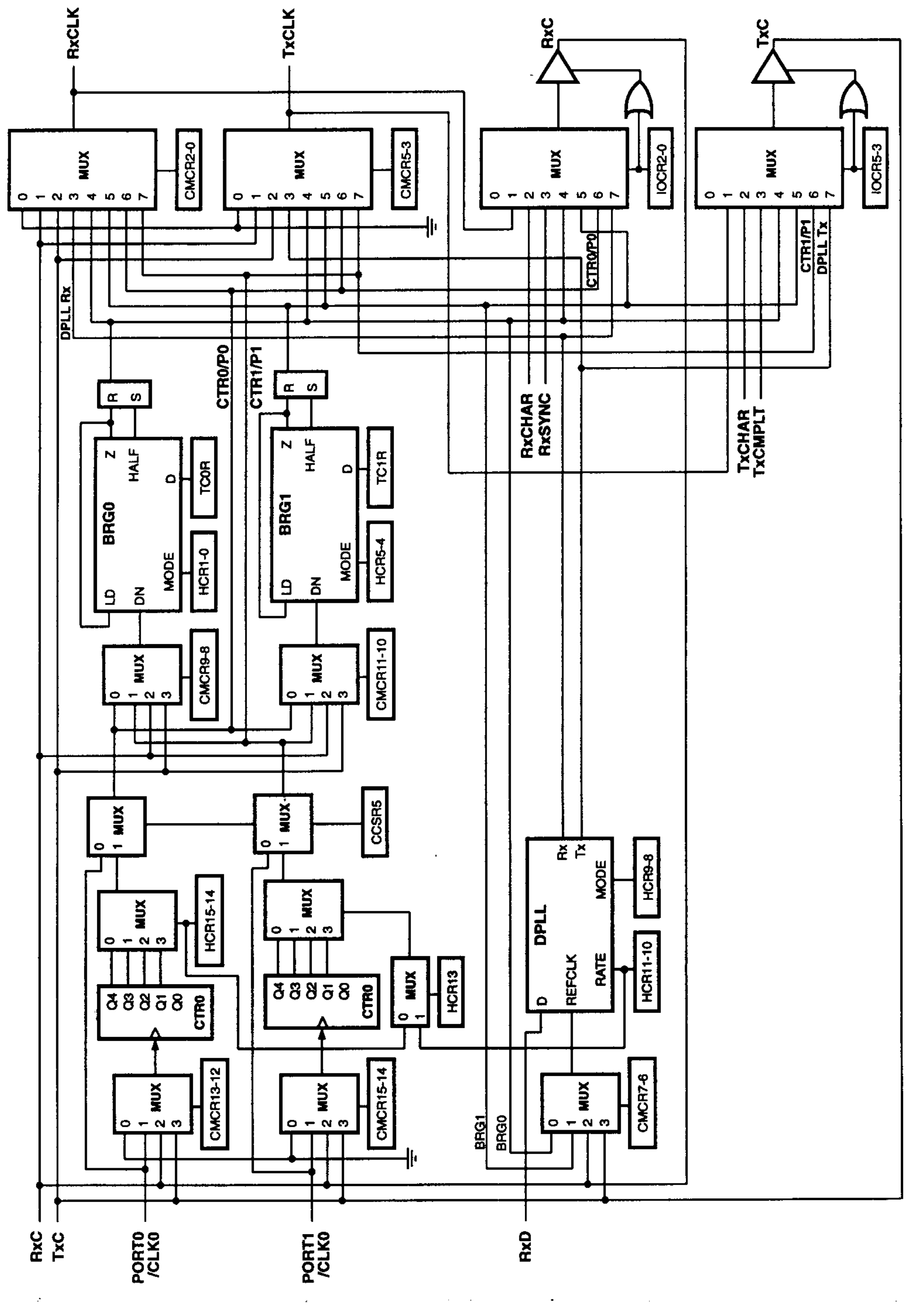


Figure 4-1. A Model of the Z16C32's Clocking Logic

4.3 TRANSMIT AND RECEIVE CLOCKING (Continued)

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

Figure 4-2. Clock Mode Control Register (CMCR)

CTR0Div		CTR1 DSel	CVOK	DDPLLDiv		DDPLLMode		Reserved		BRG1S	BRG1E	Reserved		BRG0S	BRG0E
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 4-3. Hardware Configuration Register (HCR)

Figure 4-3 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 IUSC interprets the selected 2-bit field as shown above for CTR0Div.

4.3.2 Using PORT1-PORT0 for Bit Clocking

With the Z16C32, a clock on the PORT0/CLK0 and/or PORT1/CLK1 pin(s) can be used directly as RxCLK and/or TxCLK, without being divided down by CTR0/ CTR1 respectively. This feature is controlled by the **CtrBypass** bit in the Channel Command/Status Register (CCSR5).

When this bit is 0, the outputs of CTR0 and CTR1 can be used directly as RxCLK and/or TxCLK, as inputs to the two Baud Rate Generators called BRG0 and BRG1, and can be routed to the /RxC or /TxC pin.

When CtrBypass is 1, both Counters are effectively by-passed. The signals from PORT0 and PORT1 can be used directly as RxCLK and/or TxCLK, as inputs to the Baud Rate Generators, and can be routed to the /RxC and /TxC pins. When using this option, always program CTR0Src and CTR1Src as 00 to save power, because there is no reason for the Counters to run.

4.3.3 Baud Rate Generators

Two 16-bit down counters called BRG0 and BRG1 form the second stage of the IUSC's clock-generation logic. The **BRG0Src** and **BRG1Src** fields in the Clock Mode Control Register (CMCR9-8 and CMCR11-10, respectively) control what the BRGs use as inputs:

BRGnSRC	BRGn clock source
00	CTR0 output or PORT0
01	CTR1 output or PORT1
10	/RxC pin
11	/TxC pin

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 an IUSC can derive many different baud rates 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 is in Single Cycle mode and has counted down to zero and stopped.

The **TC0RSel** bit in the Receive Interrupt Control Register (RICR0) and the **TC1RSel** bit in the Transmit Interrupt Control Register (TICR0) control what data the IUSC provides when software reads the TC0R and TC1R addresses. If a TCnRSel bit is 0, the IUSC returns the time constant value last written to TCn. At the time that a 1 is written to a TCnRSel bit, the IUSC 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 TCnR 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 is not used to make a serial clock, software can use it for other purposes such as protocol timeouts, and can program the IUSC to request an interrupt when it counts down to zero. Chapter 7 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).

4.3.4 Introduction to the DPLL

A Digital Phase Locked Loop (DPLL) circuit comprises the "third stage" of the IUSC'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 8, 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 is not used, software can program this value together with writing a 1 to 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 is 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.

4.3 TRANSMIT AND RECEIVE CLOCKING (Continued)

4.3.5 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 (transmitter disabled)
001	/RxC pin
010	/TxC pin
011	Tx output of DPLL
100	BRG0 output
101	BRG1 output
110	PORT0 or CTR0 output
111	PORT1 or 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	PORT0 or CTR0 output
111	PORT1 or CTR1 output

4.3.6 Clocking for Asynchronous Mode

For asynchronous reception, transitions on RxCLK do not 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 (8, 16, 32). 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 the 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 MS bits 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).

4.3.7 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 not send a separate clock over a telephone line. In such cases it is common practice to encode the data so that serial stream also includes clocking information. For such applications, the IUSC can encode transmitted data and decode received data in any of several popular formats.

In addition, the IUSC'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.

4.3.8 Stopping the Clocks

CMOS circuits like those in the IUSC draw less power than older technologies, but their power requirements can be reduced still further if their clock signals are stopped when the circuits do not 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 serial-side power savings is obtained by having external logic stop the input clock(s) on the /RxC, /TxC, and/or PORT/CLK1-0 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 three RxCLKs after the receiver samples the last bit of a closing Flag. For Async it means at least three 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).

4.4 DATA FORMATS AND ENCODING

The IUSC's Transmitter and Receiver can handle data in any of the eight formats shown in Figure 4-4. 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 IUSC 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

Non-Return-to-Zero (NRZ) mode does not 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.

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.

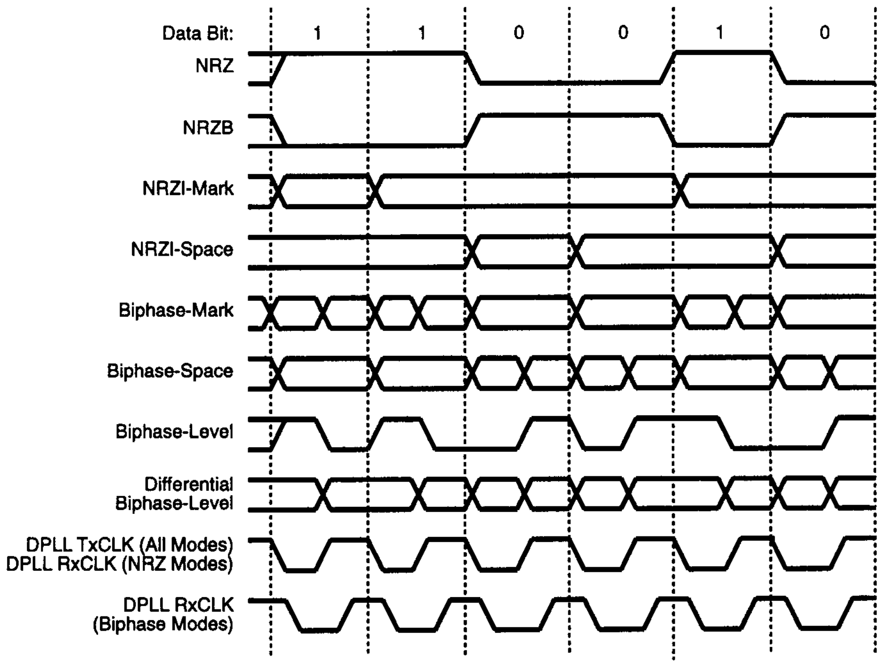
None of these NRZ-type modes, by itself, guarantees transitions in the data stream. However, if the serial protocol can guarantee transitions often enough, then the DPLL can use these transitions to recover a clock from the data stream. By some method the protocol must eliminate long bit sequences without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

4.4 DATA FORMATS AND ENCODING (Continued)

See the waveform for Biphase-Mark mode in Figure 4-4. 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 1 and Low for a 0. 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.



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 4-4. Data Formats/Encoding

4.5 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.

The DPLL produces clock transitions only when it is "in sync:" as described below.

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.

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 4-5 shows the IUSC'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 and 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.

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

Figure 4-5. Channel Command/Status Register (CCSR)

4.5 MORE ABOUT THE DPLL (Continued)

In Biphase-Level and 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 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 does not reapply 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 does not 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 is described in Chapter 5. Regardless of CVOK, code violations in two consecutive bit cells set the DPLL2Miss and DPLLDSync L/U bits and desynchronize 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 in the Channel Command/Status Register (CCSR12), or in a Biphase mode when it detects two consecutive missing clocks. In this mode, the next RxD transition (that is allowed by the DPLLEdge field) resynchronizes the DPLL counter and puts the DPLL "back in sync."

The DPLL watches the RxD line for transitions, and classifies them as either clock or data. Depending on the position of transitions within each bit cell, the logic adjusts the phase of the DPLL output clock to synchronize the clock with the bit cell boundaries of the incoming data. "Fast Sync" tells the DPLL that the NEXT edge it sees is the one to synchronize to; otherwise the DPLL has to see "n" correct edges before becoming "in sync." This "n" is about 3 for X8 mode, 6 for X16, and 12 for X32.

The time required to get in sync in the worst case is thus a function of the data encoding method as well as the data on the line. The key issue is the number of "edges" the DPLL sees on RxD.

DPLLSync (CCSR12) reads as 1 when 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 IUSC sets the **DPLLDSync L/U** bit when it loses sync in a Biphase mode. This bit is similar to DPLL2Miss in that once it is set, it stays that way until software writes a 1 to the bit to "unlatch" it. Chapter 7 explains how to program the IUSC so that it interrupts the host processor when it sets DPLLDSync L/U.

4.6 THE RXD AND TXD PINS

In some sense these are the most important pins on an IUSC. Typically they carry the serial input to the Receiver and the serial output of the Transmitter respectively. Figure 4-6 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 IUSC 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 IUSC 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 IUSC'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).

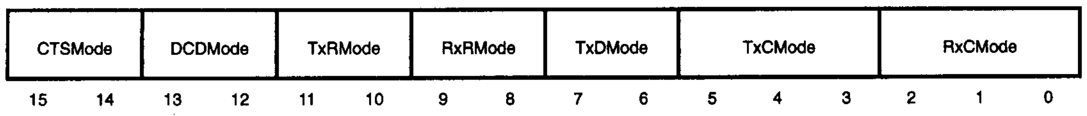


Figure 4-6. Input/Output Control Register (IOCR)

4.7 EDGE DETECTION AND INTERRUPTS

Software can program the IUSC 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 4-7 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 7 describes the IUSC's interrupt features in detail.) A 1 in one of these bits makes the IUSC 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 the IUSC detects an edge that is 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 IUSC's register map. Instead, as shown in Figure 4-8, 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 IUSC 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 7 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.

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 IUSC 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.

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 4-7. 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 4-8. Miscellaneous Interrupt Status Register (MISR)

4.8 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 4-9 shows how the required relationship between /DCD and RxD varies depends on the Receiver mode:

- 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 so that it sets up to the rising edge of RxCLK before the one at which the Receiver should capture the first data bit. For 802.3 /DCD should go low when carrier is detected—a figure in Chapter 5 shows that the timing relationship to RxD is not critical but /DCD should go low no later than six bits into the 64 alternating bits that precede the frame. 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.

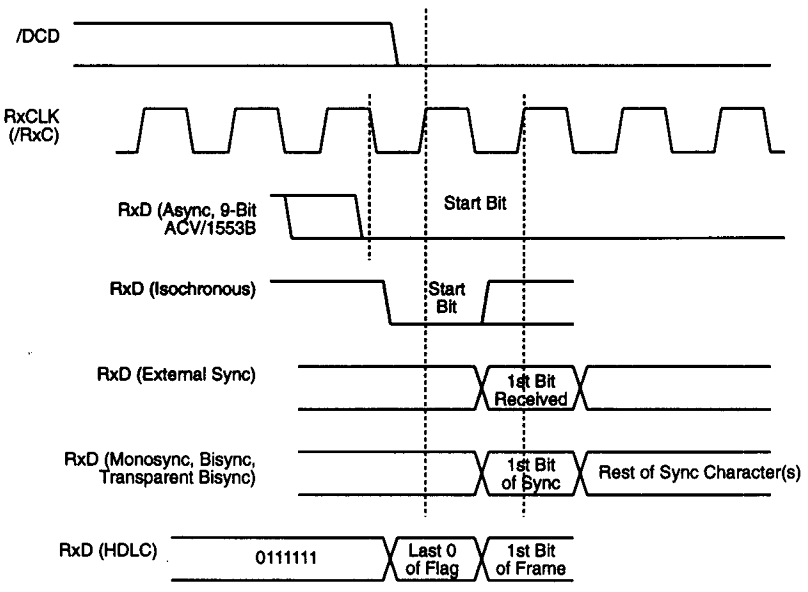


Figure 4-9. /DCD Auto-Enabling Timing

4.8 THE /DCD PIN (Continued)

Sync conditions generated internal to the IUSC are not output on this pin as on certain predecessor devices, but can be output on either the /RxC or PORT5 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 IUSC 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 pull-down resistor to the /DCD pin, because the IUSC will not drive the pin from the time /RESET goes low until the software programs DCDMode to 10 or 11.

Software can program the IUSC 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 the IUSC 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 IUSC 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.

4.9 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 Figure 4-10, 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.

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 IUSC 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 pull-down resistor to the /CTS pin, because the IUSC won't drive the pin from the time /RESET goes low until the software programs CTSMode to 10 or 11.

Software can program the IUSC 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 the IUSC 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 IUSC 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.

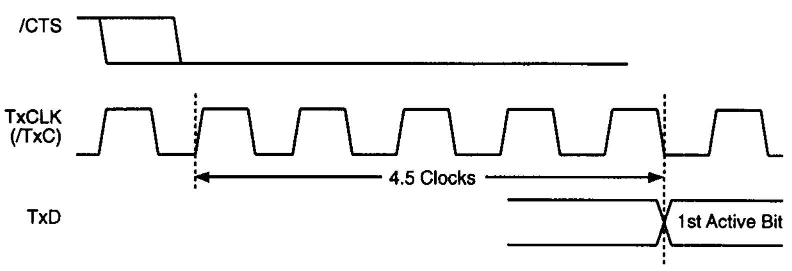


Figure 4-10. /CTS Auto-Enable Timing

4.10 THE /RXC AND /TXC PINS

Figure 4-1 (near the start of this chapter) shows the IUSC'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 PORT0 or CTR0 out
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 PORT1 or CTR1 out
111	/TxC carries the DPLL Tx output

Some of these possible outputs need further description. An IUSC drives the 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 the Transmit Character Clock high for one TxCLK period each time it transfers a character from the Transmit FIFO to the Transmit shift register. The /RxSYNC output goes low for one RxCLK cycle each time the 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 is not very useful to use a high-speed free-running clock as a source of interrupt events, for other uses of /RxC and /TxC software can program an IUSC 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 an IUSC 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 IUSC 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 a 1 indicates a low.

4.11 THE /RXREQ AND /TXREQ PINS

The predecessor USC device provides separate /RxREQ and /TxREQ outputs for signaling an off-chip DMA controller when the Transmit and Receive FIFO's are in a programmed degree of "readiness" for DMA data transfer. It also provides /RxACK and /TxACK inputs by which the external DMA controller could signal that a "flyby" DMA transfer was occurring.

The IUSC includes internal Request and Acknowledge connections between its serial controller and integrated DMA channels. Therefore there's little need for such pins, and in fact there are no ACK pins. The /RxREQ and /TxREQ pins survive for testing reasons, and can be used in applications as general I/O's under control of the **RxRMode** and **TxRMode** fields of the I/O Control Register (IOCR9-8 and IOCR11-10 respectively):

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

Note that software does not have to program these fields as 01 in order to use the IUSC's DMA channels.

Software can program an IUSC 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 the IUSC detect falling edges on /RxREQ or /TxREQ, and should write a 1 to **RxRUp IA** or **TxRUp IA** (SICR10 or SICR8) to make it detect rising edges.

As described in Edge Detection and Interrupts, the **RxR L/U** or **TxRL/U** bit (MISR11 or MISR9) is 1 if the IUSC 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 a 1 indicates a low.

The IUSC does not provide /RxACK and /TxACK pins, and so its Transmitter and Receiver cannot be used with an external "flyby" DMA controller. The fields associated with these pins in predecessor devices, HCR7-6 and HCR3-2, are not used in the IUSC.

4.12 THE PORT PINS

These eight pins can be individually programmed to be general-purpose inputs or outputs. Alternatively, seven of the eight can carry a specific, dedicated input or output signal. Regardless of the directions and roles of the various pins, transitions on all eight are latched by the IUSC. Host software can read this latched status from the Port Status Register (PSR). Unlike the pins described in earlier sections, transitions on PORT7-0 cannot make the IUSC interrupt the host processor.

Figure 4-11 shows the Port Control Register (PCR). It includes eight **PnMode** fields, each of which determines the use of one PORT pin:

PnMode	Function of PORTn pin
00	General-purpose input
01	Dedicated I/O
10	Low output
11	High output

The "dedicated I/O" function differs for each pin:

PORT7	Tx Complete output
PORT6	/FSYNC input
PORT5	/RxSYNC output
PORT4	Tx Time Slot Assigner Gate output
PORT3	Rx Time Slot Assigner Gate output
PORT2	LocalTalk Driver Enable
PORT1	Reference clock input to CTR1
PORT0	Reference clock input to CTR0

(Other sections of this chapter or Chapter 5 describe the utilization of each of these inputs and outputs.)

On the Z16C32, a hardware or software Reset makes all the PORT pins act as inputs. As noted earlier for /DCD and /CTS, for Port pins that are outputs, the system designer may want to connect a pull-up or pull-down resistor of about 10 Kohms to the pin(s), to assure their state from when /RESET goes low to the time that software programs the PCR.

Whether the various pins are inputs or outputs, the IUSC detects and latches transitions on all eight of them, and host software can read the latched status from the Port Status Register (PSR). Figure 4-12 shows how this register includes two bits for each pin, one called **PnL/U** (for Latched/Unlatch) that can be both written and read back. The other bit of each pair is called **/Pn** and can only be read, which is to say, the IUSC ignores data written to the /P7-0 bits.

After software writes a 1 to a particular PnL/U bit, the PnL/U bit reads back as a 0 and the associated /Pn bit reflects the state of the corresponding PORTn pin at the time of the write operation. After a Reset PnL/U is 0 and /Pn reflects the state of the pin when /RESET went high. A 0 in a /P7-0 bit corresponds to a high on the associated pin, and a 1 corresponds to a low.

The PnL/U bit remains 0, and /Pn does not change, until the IUSC detects a rising or falling transition on the associated PORTn pin. After such a transition, PnL/U reads back as 1 and /Pn reads as 0 for a rising edge and 1 for a falling edge. The two bits remain in this state, regardless of further transitions on the PORTn pin, until host software writes a 1 to PnL/U. This clears the PnL/U input bit to 0 and "unlatches" the transition detecting logic for the pin, although the IUSC will set the L/U bit again immediately if one or more transitions occurred while it was set. Writing a 0 to a PnL/U bit has no effect on the logic for that pin.

One mode in which software can use the Port logic is to read the PSR and immediately write back what it has read. Software can then look for 1's in any and all "interesting" PnL/U bits, and process/handle all such changes without rereading the PSR. To obtain the current state of a PORTn pin, software can write a 1 to its PnL/U bit and then immediately read the PSR.

P7Mode	P6Mode	P5Mode	P4Mode	P3Mode	P2Mode	P1Mode	P0Mode
15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0

Figure 4-11. Port Control Register (PCR)

P7L/U	/P7	P6L/U	/P6	P5L/U	/P5	P4L/U	/P4	P3L/U	/P3	P2L/U	/P2	P1L/U	/P1	P0L/U	/P0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 4-12. Port Status Register (PSR)

4.13 TIME SLOT ASSIGNERS

In applications such as ISDN and Fractional T1, a set of independent voice and data streams share a high speed link by means of time multiplexing. The IUSC can send and/or receive such a data stream with the aid of its Transmit and Receive Time Slot Assigner logic (TTSA and RTSA).

To use the IUSC in such an application, external logic must find the start point of (or at least a consistent point in) each cycle of the total data stream, and signal the IUSC when this point occurs, using a "Frame Sync" pulse on the PORT6//FSYNC pin that is low for one period of RxCLK and/or TxCLK. Both the Receive and Transmit Time Slot Assigners use this pulse. This means that if both the Receiver and Transmitter are operating simultaneously in a Time Slotted application, they must both be operating in (different parts of) the same overall data stream.

In order to use the Rx Time Slot Assigner, RxCLK must come directly from the /RxC pin, and in order to use the Tx Time Slot Assigner TxCLK must come directly from the /TxC pin. The clocking must be set up this way even though the two clocks are typically identical.

Figure 4-13 shows how the Time Slot Assigners determine when to start receiving and/or transmitting in each cycle. After sensing the /FSYNC pulse, the RTSA waits for a number of RxCLK cycles (bit times) that is determined by the **RTSASlot** and **RTSAOffset** fields in the Receive Interrupt Control Register (RICR). Specifically, it waits for this many RxCLK cycles (bits): eight times the value in RTSASlot, plus the value in RTSAOffset.

Unless both fields are zero, the RTSA blocks RxCLKs to the Receiver for this number of bits. Then it allows RxCLK to reach the Receiver for the number of consecutive bytes/octets/slots programmed into the **RTSACount** field in RICR. That is, it allows 8(RTSACount) RxCLKs to reach the Receiver. Figure 4-14 illustrates these points. (A zero in the RTSACount field disables the whole RTSA feature.) Then the RTSA again blocks RxCLKs to the Receiver until after the next pulse on /FSYNC.

The net result of this clock-gating is that the IUSC can receive up to 15 consecutive bytes/octets out of each cycle on the serial link. This data can start at any point within the first 128 octets of each cycle. The TSAs also allow for possible delays in sensing and signaling the frame sync.

In ISDN circles it seems to be common parlance to refer to the octets in each frame as numbered "slots" starting at 0. Given this definition of "slot number" if the frame sync detection logic is such that /FSYNC will be sampled low in the bit time before RxD should be sampled for the first bit of the first slot, then RTSAOffset should be programmed with zero and RTSASlot should be programmed with the slot number of the first octet that should be received.

Otherwise, call the "Frame Sync delay" one if /FSYNC will be sampled low in the same bit time that the first bit of the first slot is available on RxD, two if /FSYNC is low in the bit time after the first bit appears on RxD, and so on up through the maximum value of seven if /FSYNC is low six bit times after the first bit of the first slot appears on RxD. In these cases, the first slot cannot be received: program the RTSAOffset field with eight minus the "Frame Sync delay", and program RTSASlot with the slot number of the first octet that should be received, minus one.

Figure 4-13 applies equally to the transmit side: the TTSA similarly blocks TxCLKs to the Transmitter for the number of TxCLK cycles programmed in the **TTSASlot** and **TTSAOffset** fields in the Transmit Interrupt Control Register (TICR).

After blocking TxCLKs for 8(TTSASlot) + (TTSAOffset) bits, the TTSA allows TxCLK to reach the Transmitter for the number of consecutive bytes/octets/slots programmed into the **TTSACount** field in the Transmit Interrupt Control Register (TICR). That is, it allows 8(TTSACount) TxCLKs to reach the Transmitter, as shown in Figure 4-14. (As for the receive side, zero in the TTSACount field disables the whole TTSA feature.) Then the TTSA again blocks TxCLKs to the Transmitter until after the next pulse on /FSYNC.

Thus, symmetrically with the receive side, the IUSC can transmit up to 15 consecutive bytes/octets/slots in each cycle on the serial link. This data can start at any point within the (first) 128 octets of each cycle, and the TTSA allows for possible delays in sensing and signaling the frame sync.

Since the IUSC maintains output drive on TxD throughout each cycle on the serial link, this kind of time-multiplexed environment requires an external driver with an enable/disable input. The IUSC can provide the required "Transmit Gate" signal on the PORT4 pin. Figure 4-14 shows how this signal goes low while the TTSA is enabling the Transmitter in each frame. There is also a similar facility by which the RTSA's low-active Receive Gate signal can be output on the PORT3 pin, but the application of this signal is less obvious. As already noted in the section on the PORT pins, the P4Mode and/or P3Mode fields of the Port Control Register (PCR9-8 and/or PCR7-6 respectively) should be 01 to enable these options.

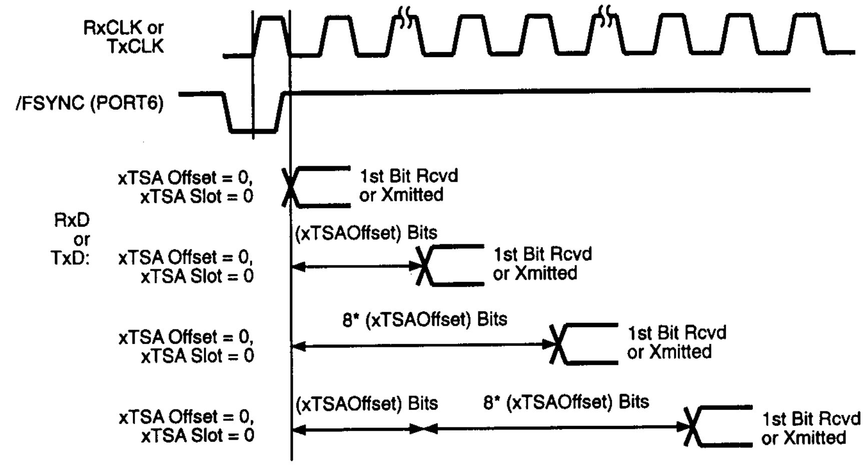


Figure 4-13. Start of Received or Transmitted Data in a TSA Application

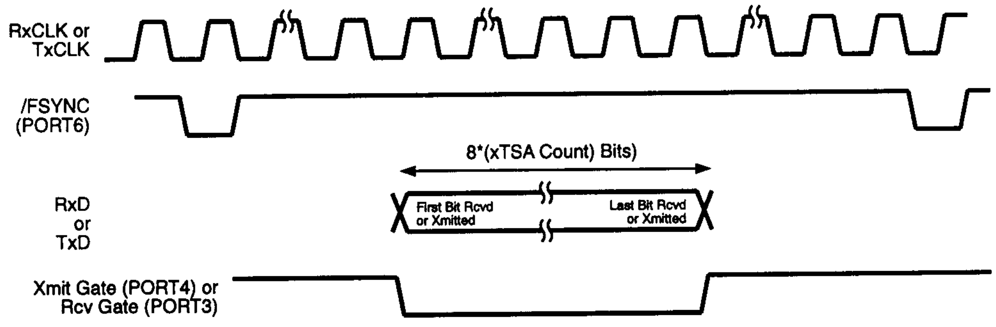


Figure 4-14. Length of Received or Transmitted Data in a TSA Application

4.13.1 Programming the Time Slot Assigners

There is an intentional vagueness in the preceding description of the Time Slot Assigner control fields as being "in" the Receive and Transmit Interrupt Control Registers (RICR and TICR). These two registers are somewhat more complex than other IUSC registers—this section describes how to access the TSA fields.

Figure 4-15 shows how the less-significant byte of both the RICR and TICR contains fixed data, but any of five different internal registers can be selected as the more-significant byte of each register. At the first level of data structure, four of the commands that can be written to the RCmd field of the Receive Command/Status Register (RCSR15-12) select the contents of RICR15-8. Similarly, four of the commands that can be written to the TCmd field of the Transmit Command/Status Register (TCSR15-12) select the contents of TICR15-8. The encoding of both sets of commands is the same:

xCmd	Contents of xICR15-xICR8
0100	xTSA data
0101	Current xFIFO Level
0110	xFIFO Level for Interrupt
0111	xFIFO Level for DMA Request

(where "x" stands for either "R" or "T"). The other options will be discussed in subsequent chapters. For our purposes it is sufficient to note that "TSA data" can be read and written as xICR15-xICR8 if the 0100 command has been written to xSCR15-xSCR12 more recently than 0101, 0110, or 0111. The IUSC resets to reading the Current FIFO level in both the RICR and TICR.

Figure 4-14 also shows how a second level of data structuring determines the meaning of "TSA data". For write operations, the bit written in the "bit 8" position selects the destination of the data:

xICR8 value	Destination of xICR15-9
0	xICR15-9 —> xTSASlot
1	xICR15-13 —> xTSAOffset
1	xICR12-9 —> xTSACount

Reading "TSA data" from RICR or TICR always yields the xTSASlot value, with the LS bit of the MS byte equal to zero.

In summary, to set up xTSA, first write the 0100 command to the xCmd field of the xSCR. Then write the xTSASlot value to the MS byte of xICR with the LS bit of the byte equal to 0. Finally, write the xTSAOffset and xTSACount values to the MS byte of xICR with the LS bit of the byte equal to 1.

It is good programming practice to follow the writing of TSA data with writing a "Select RICRHi=FIFO Status" command to the RCSR, and/or a "Select TICRHi=FIFO Status" command to the TCSR as applicable, to protect the TSA data from inadvertent modification when other parts of the software change the IA bits in the LS byte of the RICR or TICR.

Code that writes or reads "TSA data" must ensure that no interrupts will occur between the time it writes the "Select xICRHi=TSA Data" command to the TCSR or RCSR, and when it finishes writing or reading the TSA data in the TICR or RICR, if such interrupts can lead to other code writing a different Select command (for a FIFO Fill level or threshold) to the same Command/Status Register.

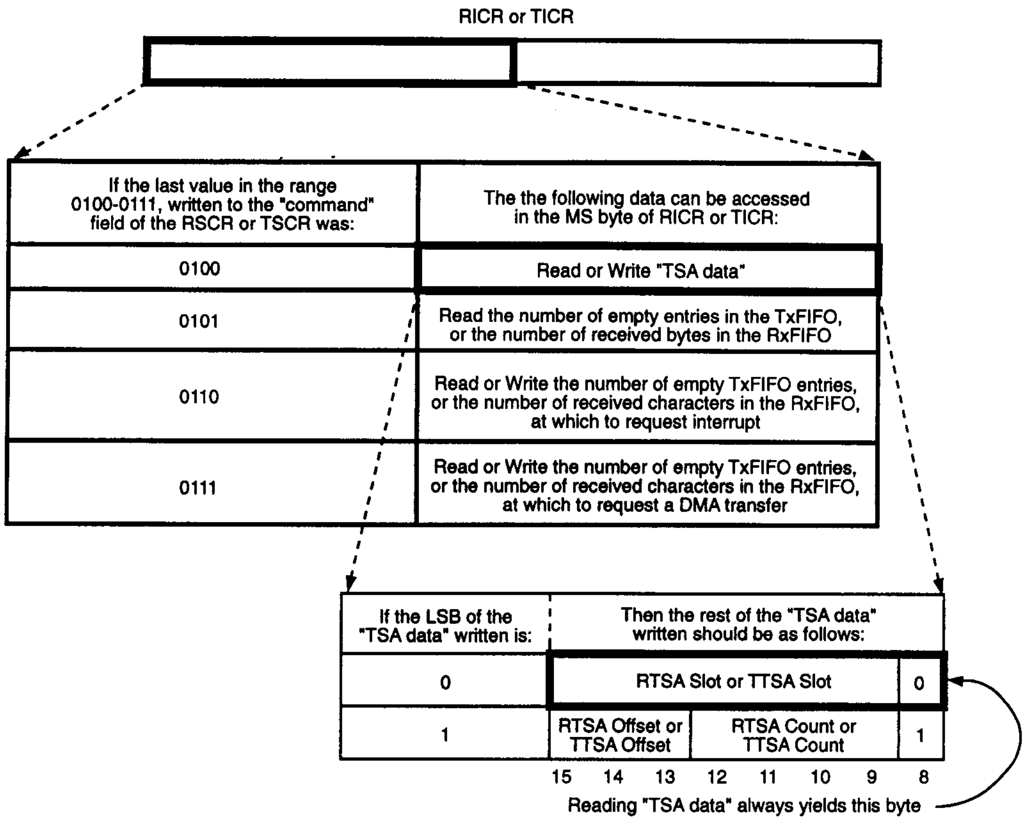

RICR or TICR
If the last value in the range 0100-0111, written to the "command" field of the RSCR or TSCR was:
The the following data can be accessed in the MS byte of RICR or TICR:
0100
Read or Write "TSA data"
0101
Read the number of empty entries in the TxFIFO, or the number of received bytes in the RxFIFO
0110
Read or Write the number of empty TxFIFO entries, or the number of received characters in the RxFIFO, at which to request interrupt
0111
Read or Write the number of empty TxFIFO entries, or the number of received characters in the RxFIFO, at which to request a DMA transfer
If the LSB of the "TSA data" written is:
Then the rest of the "TSA data" written should be as follows:
0
RTSA Slot or TTSA Slot
0
1
RTSA Offset or TTSA Offset
RTSA Count or TTSA Count
1
15 14 13 12 11 10 9 8
Reading "TSA data" always yields this byte

Figure 4-15. Structure of the RICR and TICR

4.14 THE LOCALTALK (APPLETALK) INTERFACE

With the IUSC, Zilog customers can implement a LocalTalk interface with far less time-critical software attention than is needed with other devices, including Zilog's own SCC family. LocalTalk (with its close relative Farallon PhoneNet) is the commonest and lowest-cost physical and link layer used in AppleTalk networks.

If software programs the P2Mode field in the Port Configuration Register (PCR5-PCR4) to 01, IUSCs will output a signal on the PORT2 //LTTxEnab pin that is suitable for enabling an external RS-422 driver in a LocalTalk network.

The PORT2 pin is connected to the active low output enable input of the RS-422 driver, and the TxD pin to its data input. Software should program other register fields as follows:

TxMode (CMR11-8) and RxMode (CMR3-0)	= 0110 : HDLC/SDLC
TxEncode (TMR15-13) and RxDecode (RMR15-13)	= 101: Biphase-Space (FM0)
DPLLMode (HCR9-8)	= 10 (Biphase-Space)
RxClkSrc (CMCR2-0)	= 011 (Rx clock from DPLL)
TxIdle (TCSR10-8)	= 011 (idle = continuous encoded ones)
TxPreL (CCR11-10)	= 10 (16-bit Preamble pattern)
TxPrePat (CCR9-8)	= 10 (Preamble ones or Flags)
FlagPreamble (CCR12)	= 1 (Preamble = Flags)

When the Transmitter wants to send a frame, as it sends the first of the two "preamble Flags" it enables the external driver for its first bit, then disables it for the next four bits, and then enables it throughout the rest of the first Flag, the two subsequent Flags, the frame itself, the closing Flag, and the following 16 bit times during which it sends "idle" ones. All this is as specified by Apple for LocalTalk, including the resultant encoding violations in the first Flag, and the Abort sequence after the closing Flag. Best of all, it is all done automatically by the IUSC, with no need for software intervention.

CHAPTER 5
Z16C32 IUSC™ SERIAL MODES AND PROTOCOLS

5.1 INTRODUCTION

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 IUSC 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.

5.2 ASYNCHRONOUS MODES

These protocols date back to when the first teletypewriters were succeeding Morse code, although there have been various changes since. Figure 5-1 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 is the same length as each following data bit. Each stop bit is a period of mark/one that is more than half a bit time long with a typical minimum duration of one bit time. (The IUSC 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 the IUSC 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.

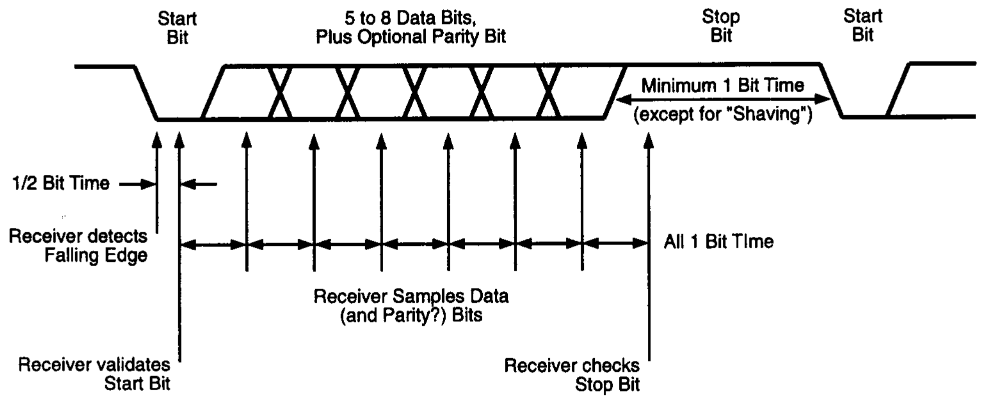

Start Bit
5 to 8 Data Bits, Plus Optional Parity Bit
Stop Bit
Start Bit
Minimum 1 Bit Time (except for "Shaving")
1/2 Bit Time
Receiver detects Falling Edge
All 1 Bit Time
Receiver Samples Data (and Parity?) Bits
Receiver validates Start Bit
Receiver checks Stop Bit

Figure 5-1. Asynchronous Data

5.2 ASYNCHRONOUS MODES (Continued)

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 actually check parity nowadays, although they may generate it 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.

The IUSC 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.

5.3 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 does not 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 5-2 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.

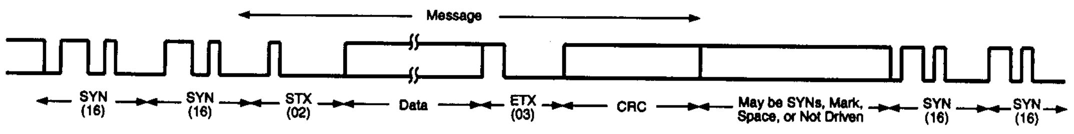


Figure 5-2. Character Oriented Synchronous Data

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 IUSC hardware can automatically generate and check various kinds of CRCs in synchronous modes.

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 the IUSC'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 IUSC 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. Biphase-level data encoding must be selected in the Transmit and Receiver Mode Registers (TMR and RMR), as described in Chapter 4. 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.

5.4 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 5-3 shows how in bit-oriented modes, a "frame" is a group of sequential characters, 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

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 do not 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., seven or more 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 integral number of characters.)

The IUSC 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.

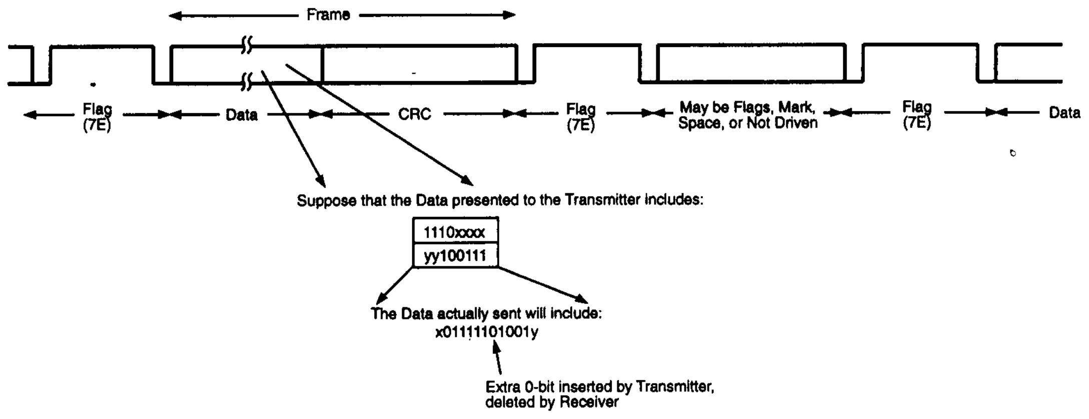

Frame
Flag (7E)
Data
CRC
Flag (7E)
May be Flags, Mark, Space, or Not Driven
Flag (7E)
Data
Suppose that the Data presented to the Transmitter includes:
1110xxxx
yy100111
The Data actually sent will include:
x01111101001y
Extra 0-bit inserted by Transmitter, deleted by Receiver

Figure 5-3. HDLC/SDLC Data

5.5 THE MODE REGISTERS (CMR, TMR AND RMR)

Three Mode registers control the basic operation and serial protocol of the IUSC's Transmitter and Receiver.

The Channel Mode Register (CMR) selects among the various communication protocols mentioned in the preceding sections. Figure 5-4 shows that the MS byte controls the mode of the Transmitter, while the LS byte controls that of the Receiver. Software can select the modes of the two modules independently by writing bytes to the CMR or, on a 16-bit bus, it can set both modes simultaneously using a 16-bit write.

Within each byte, the four LS bits 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 use in future USC family members; 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.)

The TxMode field should be changed only while the Transmitter is disabled in the TMR, as described in the next section. Similarly, the RxMode field should be changed only while the Receiver is disabled in the RMR. While it is possible to change the TxSubMode or RxSubMode fields while the Transmitter or Receiver is operating, the options provided by these fields are typically static in nature and the need to change them should seldom arise.

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 5-5 and 5-6 show the TMR and RMR respectively.

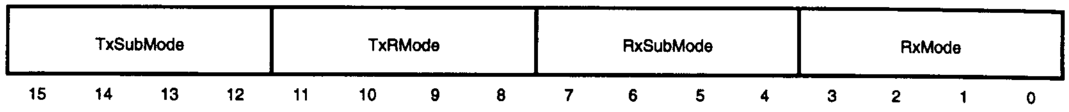


Figure 5-4. The Channel Mode Register (CMR)

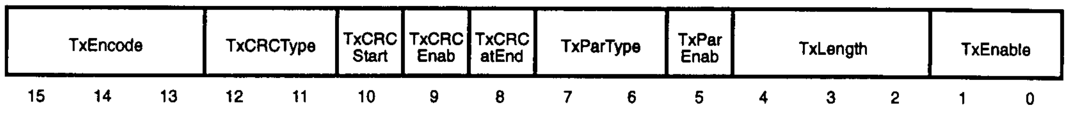


Figure 5-5. The Transmit Mode Register (TMR)

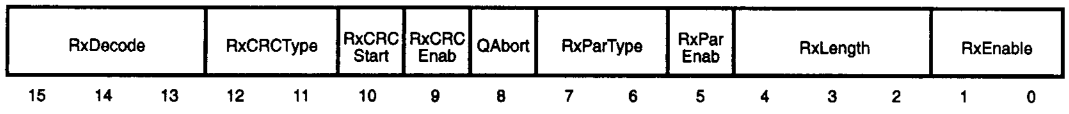


Figure 5-6. The Receive Mode Register (RMR)

5.5.1 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 does not 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 is a constant Mark/High if the MS bit 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 carries 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 does not 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.

5.5.2 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 IUSC 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 eight 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 eight 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 eight bits, the Receiver fills out the more-significant bits of each byte with the Stop bit, which is 1 except when there is a Framing Error.

When RxLength is less than eight 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 eight 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 eight bits long but subsequent characters are less than eight bits long, each subsequent character is left-justified in its byte.

3. In HDLC/SDLC mode, if the frame does not 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 are 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.

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.

5.5.3 Parity, CRC, Serial Encoding

A later section of this chapter, 'Parity Checking', discusses how bits 7-5 of the TMR and 8-5 of the RMR control parity checking. Similarly, the later section 'Cyclic Redundancy Checking' describes how bits 12-8 of the TMR and bits 12-9 of the 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 4 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

5.6 ASYNCHRONOUS MODE

Software can select classic asynchronous operation for both the Transmitter and the Receiver, by programming the TxMode and RxMode fields (CMR11-CMR8 and CMR3-CMR0 respectively) to 0000. The earlier Figure 5-1 shows how a "0" Start bit precedes each character and a "Stop bit" follows each, the latter being a "1" condition that is 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 LS bits 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 IUSC 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 4 for more information about clock selection.)

Zilog reserves the two MS bits 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 status was read from the RxFIFO, or the status of the oldest one or two characters that are still in the RxFIFO, as described later in 'Status Reporting'.

5.6 ASYNCHRONOUS MODE (Continued)

5.6.1 Break Conditions

A Break condition is a period of Space (zero) state on an Async line, that is 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 does not 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. If it is necessary to have an interrupt at the end of a Break, software can put the receiver in Monosync mode, looking for an all-ones sync character and Arm the Exited Hunt condition to flag the end of the Break. After the interrupt, software has to switch back to async mode and purge the FIFO. Alternatively, software can tell when the Break ends by polling the Break/Abort bit. The bit does not 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 4 describes the IUSC'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 is only slightly longer than a character, older conventions required a Break to be much longer: 200 milliseconds or more.

5.7 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 is 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 IUSC does not use the other three 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 an IUSC.

5.8 NINE-BIT MODE

This mode is compatible with various equipment including some Intel single-chip microcontrollers. In some contexts it is called "address wake-up 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 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 this bit as that character's address/data bit. By convention a 0 signifies "data" and a 1 signifies "address." As software or the Transmit DMA channel writes each character into the TxFIFO, the IUSC 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 set to the same value as CMR14.

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 MS bit of addresses.

RxSubMode bit 2 (CMR6) similiarly controls parity checking in the Receiver. The RxParEnab bit in the Receive Mode Register (RMR5) must be set to the same value as CMR6. As 0 disables parity checking on data bytes, allowing all 8 bits to be used for data. A 1 enables parity checking on data bytes. Address bytes are always parity checked.

As in Async mode, the two LS bits of the Tx and RxSubMode fields (CMR13-12 and CMR5-4) 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 equal 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, are not of much use in Nine-Bit mode. For example, you probably would not want to store a Receive Status Block for an address character.

The IUSC does not use the MSbit of the RxSubMode field (CMR7-6) in Nine Bit mode, but Zilog reserves this bit for future enhancements and software should program them as 00 in this mode.

5.9 EXTERNAL SYNC MODE

Software can select this mode only for the Receiver, by programming the RxMode field of the Channel Mode Register (CMR30) as 0001. This value is not defined for the TxMode field (CMR11-CMR8).

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 IUSC first samples /DCD low at the rising edge of RxCLK before the one at which it should capture the first data bit. (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 IUSC 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 is 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 IUSC does not 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.

5.10 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 an IUSC mode in which the Transmitter sends, and the Receiver searches or "hunts" for, a Sync pattern composed of two characters totaling 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 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 the last one of a 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 MS bit 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 does not send a CRC code; if there is 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 LS byte 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 is 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 does not 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 (CCR118). 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 LS bits 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" MS bits 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 eight bits long regardless of the length of data characters.

5.10 MONOSYNC AND BISYNC MODES (Continued)

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 do not 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 IUSC does not use the two MS bits 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.

5.11 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 the IUSC 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 LS bits 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 does not 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 IUSC 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 does not include it in the RxFIFO nor in the CRC calculation.

If the character after a DLE is a SYN, the Receiver excludes both the DLE and the SYN from the CRC calculation, but places both characters in the FIFO so that they will appear in the received data stream.

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 Received Data Interrupt Pending bit and thus force an interrupt request on the /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 does not 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 IUSC 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 MS bits 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 not 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 is typical of this mode, precludes the kind of automatic reception that the RSB feature allows in modes like HDLC/SDLC anyway.

The IUSC does not use the three MS bits 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.

5.12 SLAVED MONOSYNC MODE

This mode applies only to the Transmitter. Software can select 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 4.

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 not 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 IUSC 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 IUSC does not 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.

5.13 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 IUSC'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 IUSC'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 practice and IEEE 802.3. In addition, the two LS bits 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 or PORT7 to control the driver.

External logic must detect collisions that may occur while the IUSC 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.

As in other synchronous modes, the MS bit 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 5-7 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 select the mode of 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.

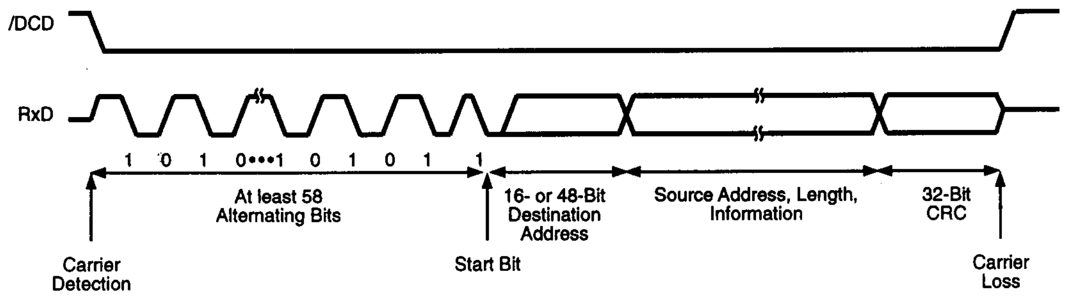

/DCD
RxD
1 0 1 0•••1 0 1 0 1 1
At least 58 Alternating Bits
16- or 48-Bit Destination Address
Source Address, Length, Information
32-Bit CRC
Carrier Detection
Start Bit
Carrier Loss

Figure 5-7. Carrier Detection for a Received Ethernet Frame

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 Received Data Interrupt Pending bit and thus force an interrupt request on the /INT pin, and/or it may force a DMA request on the /RxREQ pin.

The LS bit 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 LS bit first. The Length field that follows the Destination and Source Address fields is sent MS byte-first. IEEE 802.3 does not include any other byte ordering information.

The IUSC does not use the three LS bits of the TxSubMode field (CMR14-CMR12) in 802.3 mode, nor the three MS bits of RxSubMode (CMR7-CMR5), but Zilog reserves these bits for future enhancements. Software should always program them with zeros in this mode.

5.14 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-CMR8 and CMR3-CMR0).

In some sense this is the most important mode of the IUSC, 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 seven or more consecutive ones. 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 seven or 15 consecutive ones.

On the Transmit side, the two MS bits of the TxSubMode field (CMR15-CMR14) 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-CMR14	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.

5.14 HDLC/SDLC MODE (Continued)

After sending the sequence specified by this field, the Transmitter sends the next frame if software or the Transmit DMA channel 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-TCSR8), as described later in 'Between Messages, Frames, or Characters'. That 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. Furthermore, the same section describes the feature of a Z16C32 whereby software can ensure that a programmable minimum number of Flags is sent between frames.

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 CMR15-14 is 01 as described above, 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 Received Data Interrupt Pending bit and thus force an interrupt request on the /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.

5.14.1 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 does not matter.)

CMR7-4	Address/Control Processing
xx00	The Receiver does not handle an Address or Control field. It simply divides all the data in received frames into characters per RxLength and places them in the RxFIFO.
xx01	The Receiver checks the first eight bits of each frame as an address. If they are all ones or if they match the contents of the LS byte 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 RSR7-0, the Receiver places the first 24 bits of the frame in the RxFIFO as three 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 RSR7-0, the Receiver places the first 32 bits of the frame in the RxFIFO as four 8-bit bytes, before shifting to dividing characters according to RxLength.

CMR7-4	Address/Control Processing
0011	The Receiver processes an Extended Address at the start of each frame. First it checks the first eight bits of the frame as described above. If these bits are all onesor if they match RSR7-0, as the Receiver places each eight bits of the address into the RxFIFO, it checks the LS bit. If the LS bit is 0, it goes on to put the next eight bits into the RxFIFO as part of the address as well, through an address byte that has its LS bit 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 eight bits of the address are all ones or if they match RSR7-0, the Receiver places the 24 bits after the extended address into the RxFIFO as three 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 eight bits of the address are all ones or if they match RSR7-0, the Receiver places the next eight bits after the extended address in the RxFIFO without examination. Then, as it stores each subsequent eight bits in the RxFIFO, the Receiver checks the MS bit. If the MS bit is 1, it continues to receive more 8-bit bytes, through one that has its MS bit 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 MS bits 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 is 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 one or two characters that are still in the RxFIFO, as described in a later section, Status Reporting. Note, however, that this length checking does not report a problem if a frame ends within a CRC that follows an address and control field.

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

5.14.2 Frame Length Residuals

The Receiver detects and strips inserted zeroes, Flags, and Aborts before any other processing, and does not 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 MS bits of the byte, regardless of RxLength.) The Receiver saves the number of bits received in the 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, which reduces processing requirements still further.

Conversely, to send a frame that does not 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 5-9 shows the CCSR. The Transmit Control Block feature can be used to set the TxResidue value for each block under DMA control, without intervention by processor software. The active bits of a partial character must be right-justified, that is, they must be the LS bits 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-TMR2).

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 one to seven bits respectively.

5.14.3 Handling a Received Abort

The Z16C32 can report a received Abort sequence to software in two separate ways. The later section 'Status Handling' will note that the IUSC sets the **Break/Abort** bit in the Receive Command/Status Register (RCSR5) when it recognizes an Abort sequence. This notification is not tied to a specific point in the received data stream.

The same section will also note that, if the **QAbort** bit in the Receive Mode Register (RMR8) is 1, the Z16C32 queues Abort conditions through the RxFIFO. From there, they eventually appear as the **Abort/PE** bit (RCSR2) of the last character of the frame—the one that has RxBound (RCSR4) set to 1. (If QAbort is 0, the IUSC uses this bit in the RxFIFO and RCSR for Parity Error indication.)

With other devices, software typically handles Abort conditions by enabling an interrupt when one is detected, and at that point ignoring/purging all received data and forcing the receiver into Hunt mode for the next frame.

With the Z16C32, software can handle Aborts more efficiently/elegantly by setting QAbort to 1 and using the Receive Status Block feature to store the RCSR status in memory for each frame, as described in the later section 'Receive Status Blocks'. Software can then examine this status word for each "frame"; any one that has Abort/PE set is not a proper frame in that it ended with an Abort sequence rather than a Flag.

5.15 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-CMR8) as 1110 while programming the RxMode field (CMR3-CMR0) 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) with 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) with 000 to select Flags as the idle line state, and the Clock Mode Control Register (CMCR5-0) to select the same clock source for both TxCLK and RxCLK.

The two MS bits 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 does not recommend this option in IBM SDLC Loop applications because only the Primary station should issue a "Go Ahead" sequence (and a primary station should be in regular HDLC/SDLC mode).
01	Like 00 except that the Abort includes 15 ones.

CMR15-14	Response to Underrun
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 does not recommend this option either, because the frame format probably has not been met when there is 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 thereafter. 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 being 0, the part continues to output ones on TxD. When CMR13 is 1 after software first enables the Transmitter, the IUSC sends zeroes on TxD until the Receiver detects a "Go Ahead" sequence (01111111). At this point the IUSC 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 5-8).

The four-bit-time delay for repeating data includes one each for the RxD data decoder, the Receiver, the Transmitter, and the TxD data encoder.

OnLoop remains 1 unless the part is reset or software programs the TxMode field to a different value. Once OnLoop is 1 and the IUSC is repeating data from RxD to TxD, CMR13 controls what the Transmitter does when it receives a Go Ahead sequence.

If CMR13 is 0, the IUSC 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 is 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 IUSC has sent a closing Flag or an idle Flag, it clears the LoopSend (CCSR6) bit and returns to repeating data from RxD onto TxD. Because the Primary station sends ones between the time it sends the original GA (Abort) and when it receives a GA (Abort) back, and any and all intervening stations are repeating these ones, the repeated ones, combined with the last zero of the last closing or Idle Flag, constitute a Go Ahead (Abort) to the next station in the loop.

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

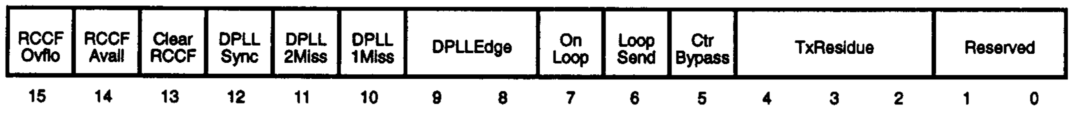


Figure 5-8. The Channel Command/Status Register (CCSR)

5.16 CYCLIC REDUNDANCY CHECKING (CRC)

The IUSC 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.

A 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 bits 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.

A 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.

A 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 RxCRCStart bits should be programmed as 1 to start the CRC generators 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 CRC** and **Clear 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 (TMR10 and RMR10) control whether the IUSC 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 is 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 does not 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 the Transmit DMA channel or software 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 one or two 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, that is, 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 is a two character delay between the time the Receiver places each received character in the RxFIFO and when it processes (or does not process) the character through the CRC generator. Therefore, software can 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. The Receiver does not 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 5-9 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 these 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 is almost always 1).

The CRCE bit that is 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 two or four 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 not 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 one or two 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, the IUSC can be used for CRC-pass-through applications. This is not true of all serial controllers.

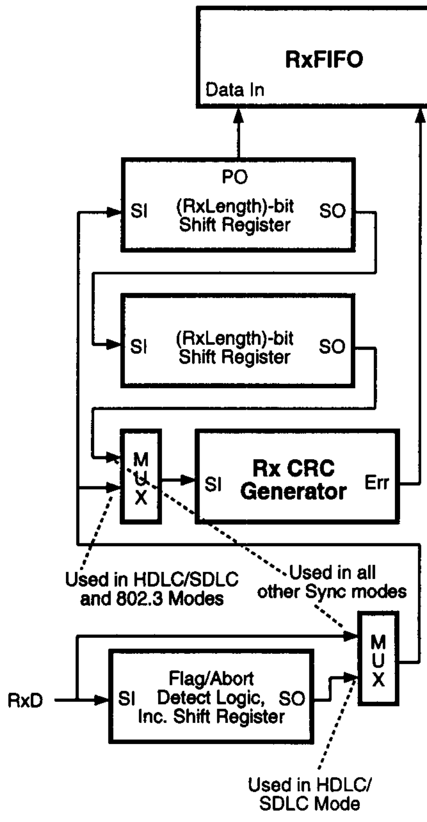

RxFIFO
Data In
PO
SI (RxLength)-bit Shift Register SO
SI (RxLength)-bit Shift Register SO
MUX
SI Rx CRC Generator Err
Used in HDLC/SDLC and 802.3 Modes
Used in all other Sync modes
MUX
RxD
SI Flag/Abort Detect Logic, Inc. Shift Register SO
Used in HDLC/SDLC Mode

Figure 5-9. A Model of the Receive Datapath

5.17 PARITY CHECKING

The IUSC can handle a Parity bit in each character in either asynchronous or synchronous modes, although many 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).

The IUSC 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 the IUSC does not carry these bits through 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.

In HDLC/SDLC protocols the Z16C32's Receiver can queue either a Parity Error or an Abort indication through the RxFIFO, but not both. Regardless of the protocol, in order to have the Receiver check parity, the QAbort bit in the Receive Mode Register (RMR8) must be 0.

If QAbort is 0, 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 Abort/PE bit in the Receive Command/Status Register (RCSR2). The Abort/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 one or two characters that are still in the RxFIFO, as described in the next section.

5.18 STATUS REPORTING

The most important status reported by the Transmitter and Receiver is available in the LS bytes of the Transmit and Receive Command/Status Registers (TCSR and RCSR). Figures 5-12 and 5-13 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 the IUSC's integrated DMA channels 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 channel to store a version of the RCSR in memory, either with the received data or with DMA control information. Such stored status differs slightly from the status in the RCSR.

Software can program the IUSC to assert its Interrupt Request output (/INT) based on certain bits in the TCSR and RCSR. Chapter 7 covers interrupts in detail; for now we'll just note that the IUSC 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 the IUSC will not 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 does not 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, CRCE/FE, Abort/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 (or the Rx DMA channel) reads data from the RxFIFO/RDR 16 bits at a time, and to 0 before it (or the Rx DMA channel) reads data eight bits at a time. Note that it is essential for software to keep WordStatus in the right state, when changing the IA bits in the LSbyte of the RICR, or when writing DMA or interrupt threshold values to the MSbyte.

The RxBound, Abort/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 in the Receive Interrupt Control Register (RICR) for one of these bits is 1, the IUSC 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 WordStatus=1, and once one of these bits is 1, it stays that way until software writes a 1 to it. (The IUSC does not actually set the Receive Status IP bit to request an interrupt for one of these bits, until software or the Receive DMA channel reads the associated character from RDR.)

For ShortF/CVType and CRCE/FE, and for RxBound, Abort/PE, and RxOver when the associated IA bit is 0, if the last time that software or the Receive DMA channel read the RxFIFO via the RDR, the IUSC 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 (MS byte of the) RCSR, or the Receive DMA channel stores it in the Receive Status Block, or until software or the Receive DMA channel reads the RDR again.

For ShortF/CVType and CRCE/FE, and for RxBound, Abort/PE, and RxOver when the associated IA bit is 0, if the last time that software or the Receive DMA channel read the RxFIFO via the RDR, the character returned (both of the characters returned) had RxBound=0, or if software has read the (MS byte of the) RCSR or the Receive DMA channel 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 the bit reflects the status of the oldest character in the FIFO.

Just in case that was not perfectly clear, the flowchart of Figure 5-11 presents the same information.

5.18 STATUS REPORTING (Continued)

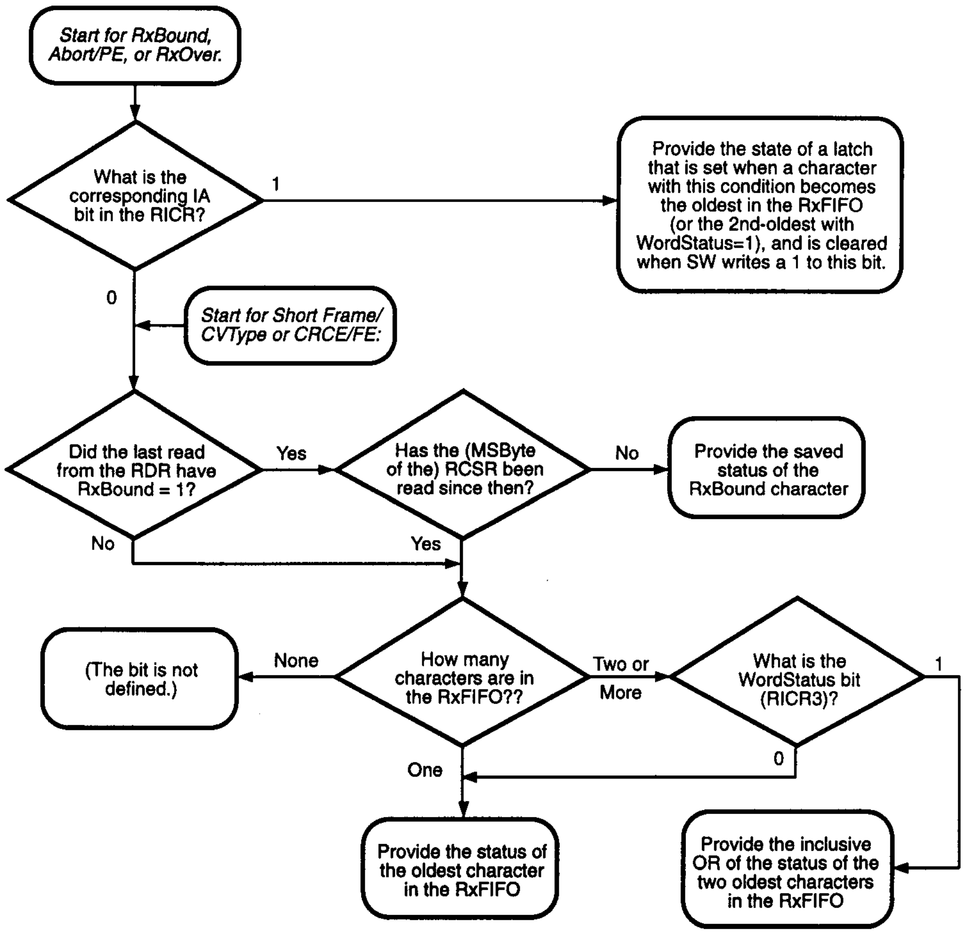

Start for RxBound, Abort/PE, or RxOver.
What is the corresponding IA bit in the RICR?
1
Provide the state of a latch that is set when a character with this condition becomes the oldest in the RxFIFO (or the 2nd-oldest with WordStatus=1), and is cleared when SW writes a 1 to this bit.
0
Start for Short Frame/ CVType or CRCE/FE:
Did the last read from the RDR have RxBound = 1?
Yes
Has the (MSByte of the) RCSR been read since then?
No
Provide the saved status of the RxBound character
No
Yes
(The bit is not defined.)
None
How many characters are in the RxFIFO??
Two or More
What is the WordStatus bit (RICR3)?
1
0
One
Provide the status of the oldest character in the RxFIFO
Provide the inclusive OR of the status of the two oldest characters in the RxFIFO

Figure 5-10. How the IUSC Provides the "Queued" Status Bits in the RCSR

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

Figure 5-11. The Transmit Command/Status Register (TCSR)

5.18.1 Detailed Status in the TCSR

PreSent: the Transmitter sets this 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). The IUSC 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.

IdleSent: the Transmitter sets this bit (TCSR6) in any mode, when it has finished sending "one unit" of the Idle line condition specified in the TxIdle field in the MS byte 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 IUSC 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 IdleSent to unlatch and clear it, and to allow further interrupts if TICR6 is 1; writing a 0 to IdleSent has no effect.

AbortSent: the Transmitter sets this bit (TCSR5) in HDLC/SDLC or HDLC/SDLC Loop mode, when it has finished sending an Abort sequence. The IUSC 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.

EOF/EOM Sent: the Transmitter sets this bit (TCSR4) in a synchronous mode, when it has finished sending a closing Flag or Sync sequence. The IUSC 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.

CRCSent: the Transmitter sets this bit (TCSR3) in a synchronous mode, when it has finished sending a Cyclic Redundancy Check sequence. The IUSC 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.

AllSent: this read-only bit (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.

TxUnder: the Transmitter sets this bit (TCSR1) in any mode, when it needs another character to send but the TxFIFO is empty. It does this even in asynchronous modes. The IUSC 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. The Transmitter sets TxUnder one or two clocks before the current character is completely sent on TxD. See 'Handling Overruns and Underruns' later in this chapter for further details on how to handle this condition.

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

5.18 STATUS REPORTING (Continued)

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

Figure 5-12. The Receive Command/Status Register (RCSR)

5.18.2 Detailed Status in the RCSR

2ndBE: the IUSC sets this read-only bit (RCSR15) to 1 when software or the Receive DMA channel 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, Abort/PE, or RxOver status. (The bit is name stands for Second Byte Exception.) The IUSC clears this bit to 0 when software or the Receive DMA channel reads data from the RxFIFO/RDR, there are two or more characters in the RxFIFO, and the Receiver did not mark the second-oldest one with any of these three conditions. If software or the Receive DMA channel reads data from the RDR when there is only one character in it, this bit is undefined until the next time one of them reads RDR.

1stBE: the IUSC sets this read-only bit (RCSR14) to 1 when software or the Receive DMA channel reads data from the RDR, and the Receiver marked the oldest character read with one or more of RxBound, Abort/PE, or RxOver status. (The bit's name stands for First Byte Exception.) The IUSC clears this bit to 0 when software or the Receive DMA channel reads data from the RDR, and the Receiver did not mark the oldest character with any of these three conditions.

ShortF/CVType: the Receiver queues this 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 one or two 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 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 has not come to the end of the specified field(s) by the end of the frame.

ExitedHunt: the Receiver sets this 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 a 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.

The IUSC 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.

IdleRcved: the Receiver sets this 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. The IUSC 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. The IUSC does not actually clear RCSR6 until software has written a 1 to unlatch it, and RxD has gone to 0 to end the idle condition. (IdleRcved is not useful in Async modes that use a 16X, 32X, or 64X clock. In these cases, keep RICR6 = 0 to avoid interrupts, and ignore RCSR6.)

Break/Abort: the Receiver sets this 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 ones, i.e., an Abort or Go Ahead sequence.

Break/Abort is not associated with a particular point in the received data stream, for either the Break or Abort condition. (But see the description of "Abort/PE" below for an Abort indication that is queued with received data.)

The IUSC 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, the IUSC does not actually clear RCSR5 until software has written a 1 to unlatch it, and RxD has gone to 1 to end the break condition.

RxBound: the Receiver queues this 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, but this indication is not visible to software until the character is the oldest one in the RxFIFO (or the second-oldest with Word Status = 1).

As described earlier in this *Status Reporting* section, RCSR4 may represent an interrupt bit, or the status associated with the oldest one or two 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. Note that the Receiver never sets this bit in other modes, including Monosync and Bisync modes.

The IUSC can request an interrupt when software or the Rx 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.

CRCE/FE: the Receiver queues this 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 one or two character(s) still in the RxFIFO, as described earlier in this 'Status Reporting' section. In a stored Receive Status Block it represents the status from 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 checking logic showed "correct" status when it stored the character in the RxFIFO, or 1 if the CRC generator was not 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.

Abort/PE: the Receiver queues this 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 one or two 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 one or two character(s).

If the **QAbort** bit in the Receive Mode Register (RMR8) is 0, the Receiver sets this bit to show a Parity Error for a character if RxParEnab (RMR5) is 1 and the character's parity bit does not match the condition specified by the RxParType field. See the earlier section 'Parity Checking' for more information.

In HDLC/SLDC mode with the QAbort bit 1, the Receiver sets this bit (along with RxBound) for a character that was followed by an Abort sequence.

The IUSC can request an interrupt when software or the Receive DMA channel reads a character from the RDR that has this bit set, if the Abort/PE IA bit in the Receive Interrupt Control Register (RICR2) is 1. In this case software must write a 1 to Abort/PE (RCSR2) to unlatch it and allow further interrupts; writing a 0 to RCSR2 has no effect.

RxOver: the Receiver queues this bit through the RxFIFO with each received character. It sets the bit to indicate a Receive FIFO overrun, but the overrun is not visible to software until the character that caused it is the oldest one in the RxFIFO (or the second-oldest with Word Status = 1).

5.18 STATUS REPORTING (Continued)

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 one or two 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 is held in a holding register between the shifter and the RxFIFO. Once this happens, the Receiver does not store any more received characters in the RxFIFO, until software responds as described in 'Handling Overruns and Underruns' later in this chapter.

The IUSC can request an interrupt when software or the Rx 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.

RxAvail: this read-only bit (RCSR0) is 1 if the RxFIFO contains 1 or more characters, or 0 if it is empty.

5.19 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 are 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 IUSC 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 the Tx DMA channel to fetch control information for each frame from memory, and the Receive Status Block feature that allows the Rx DMA channel to store status for each frame in memory. The following subsections describe these features.

5.19.1 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 cause an interrupt if a frame exceeds a certain length.

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.

Figures 5-14 and 5-15 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 length of future received messages and frames at which the Receiver will interrupt. Software can also use the Transmit Control Block feature to make the IUSC 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. The device 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 does not have any immediate effect on the TCC or RCC feature. Only when one of several events occurs does the IUSC 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 device disables the TCC or RCC feature, while if the value is non-zero it enables the feature.

The IUSC 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), or

2. software writes the Load TCC (or Load RCC and TCC) command to RTCmd in the CCAR, or

3. software writes the Purge TxFIFO (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 the Transmit DMA channel fetches (the second byte of) the second word containing the new character count. Which is to say, the IUSC fetches the count "through" the TCLR.

The IUSC 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-CCAR11), or

2. software writes the Load RCC (or Load RCC and TCC) command to RTCmd in the CCAR, or

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 the IUSC 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 one or two when software or the Transmit DMA channel loads transmit data into the TxFIFO. The Receiver decrements the RCC 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, nor a CRC that the Transmitter generates. A non-zero RCLR value can be either all ones, or the number of characters/bytes in a message or frame, above which the Receiver should interrupt, including any CRC but not including any Receive Status Block information. For frame or message-oriented applications in which there is 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 Receive 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 5-13 shows a decoder that detects when the counter contains 0001. When software or the Transmit DMA channel writes enough data into the TxFIFO so that the TCC counts down to 0, the IUSC marks the character that corresponds to decrementing from 1 to 0 as End of Frame/End of Message (EOF/EOM). 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 the Transmit DMA channel writes 16 bits to the TDR while the TCC contains 0001, the serial controller only puts the character on the IUSC's internal D7-D0 lines into the TxFIFO—it ignores the data on the D15-D8 lines. In a system in which even-addressed bytes fall on D7-D0 (e.g., a system based on a Zilog Z380™ or an Intel processor) this is not a problem. On the other hand, in systems in which even-addressed bytes reside on D15-D8 (e.g., a system based on a Zilog Z8000™ or Z16C0x or a Motorola 680x0) it can cause problems.

Chapter 6 describes a feature of the Z16C32's Tx DMA channel that helps alleviate this problem. If the Tx DMA channel is reading the data in a frame 16 bits at a time, and it decrements its Transmit Byte Count Register (TBCR) to 1, it next signals the memory for a byte read, and ensures that the data from the proper half of the data bus (according to "Select D15-D8 First" or "Select D7-D0 First" commands) is driven onto the internal D7-D0 lines for the serial controller.

Assuming that:

1. the Tx DMA channel is used,

2. the end of a transmitted frame always corresponds to the end of a memory buffer, and

3. the TBCR is programmed to reflect the number of transmit characters in the buffer, rather than relying on the Early Termination feature to terminate the buffer,

then this feature eliminates an unfortunate requirement that previous USC family members imposed on host software in Big-Endian systems. This requirement still applies when these assumptions are not met: if the last character of a frame falls at an even address in a Big-Endian system, software must copy the last character into the subsequent odd address as well, before presenting the frame to the Tx DMA channel.

5.19 DMA SUPPORT FEATURES (Continued)

The Transmitter suppresses its DMA request from the time the Transmit DMA channel 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 channel does not load the control information for the next frame or message, while the Transmitter still needs the values for the current one.

On the Receive side, software cannot 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 IUSC stores the captured RCC value as the second word of the RSB. The IUSC taks the RCC value in a 32-bit RSB from the RCC FIFO.

Note: IUSCs that do not bear the 16C32SL1660 topmarking used a hidden register called RCHR to hold the RCC value for a 32-bit RSB.

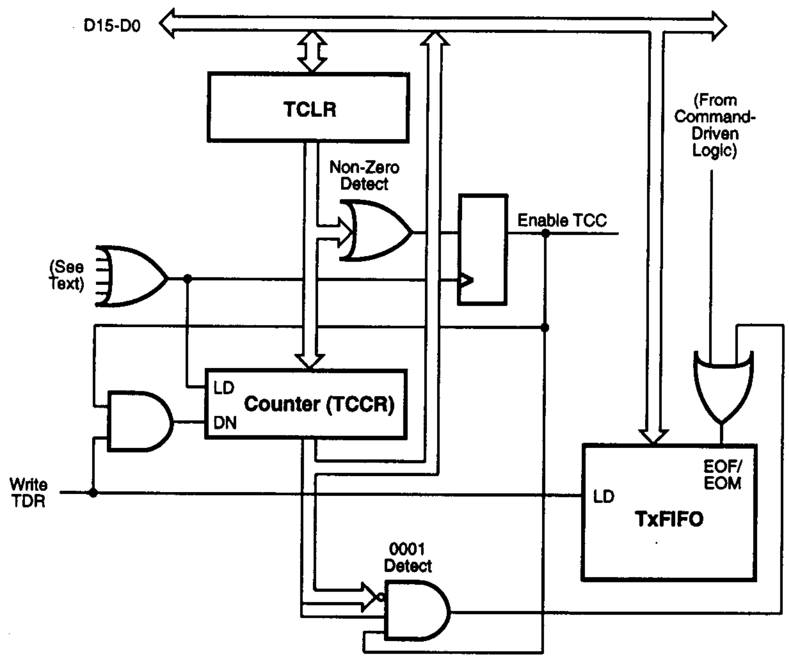


Figure 5-13. A Model of the Transmit Character Counter Feature

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 entire process.

If the IUSC decrements the RCC to 0 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 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 7.

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

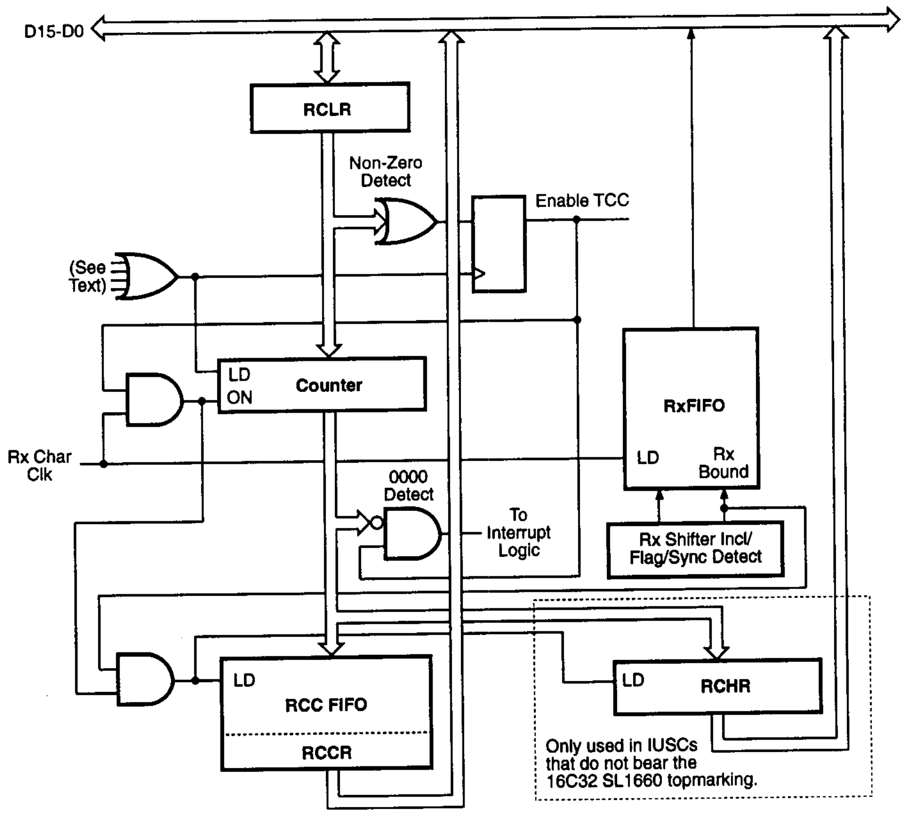


Figure 5-14. A Model of the Receive Character Counter Feature

5.19.2 The RCC FIFO

Figure 5-14 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 Bisync, 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 5-15 shows the Channel Command/Status Register (CCSR), the 3 MS bits 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 software selects 32-bit Receive Status Blocks as described in a later section, The IUSC automatically removes an entry from the RCC FIFO as they store the second word of an RSB. In other applications, software can monitor RCCFAvail to know when to read the RCC FIFO: when RCCFAvail is 1 software can read the oldest entry in the RCC FIFO from the Receive Character Count Register (RCCR). Whether the RCC residual is obtained from an RSB or by reading RCCR, software 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 Z16C32 does not set RCCFAvail, nor certain other status related to an End of Frame condition, until one bit time after it places an RxBound character in the RxFIFO. The Z16C32 delays forcing an Rx Data interrupt and/or an Rx DMA request until the same RxCLK rising edge at which it sets RCCFAvail, so that an Rx Data service routine can rely on the RCC FIFO and its status flags being current.

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 IUSC 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 in the RCC FIFO, the IUSC 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 serial controller) is to write a 1 to the **ClearRCCF** bit (CCSR13), or by writing a Purge Rx command to the RTCmd field (CCAR15-11). Either of these actions also empties the RCC FIFO and clears the RCCFAvail bit.

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

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

Figure 5-15. The Channel Command/Status Register (CCSR)

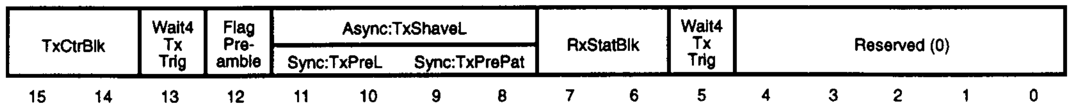


Figure 5-16. The Channel Control Register (CCR)

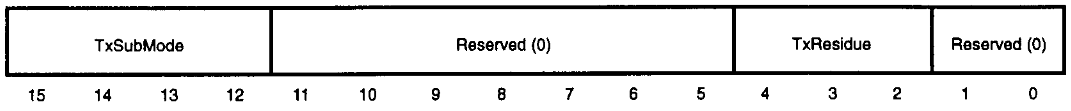


Figure 5-17. The First (or Only) 16 bits of a Transmit Control Block

5.19.3 Transmit Control Blocks

Figure 5-16 shows the Channel Control Register. Its **TxCtrlBlk** field (CCR15-14) controls what the Transmitter does with the initial data that the Transmit DMA channel or software writes to the TDR at the start of a frame or message. (While software can use Transmit Control Blocks when it fills the TxFIFO, there is no obvious reason to do so, compared to just writing the various control registers directly.) The Transmitter interprets TxCtrlBlk as follows:

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

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

1. after software writes a Trigger Tx DMA (or Trigger Tx and Rx DMA) command to the RTCmd field of the Channel Command/Address Register (CCAR15-11), or

2. after software writes a Load TCC (or Load RCC and TCC) command to RTCmd, or

3. after software writes a Purge TxFIFO (or Purge Tx and Rx FIFO) command to RTCmd, or

4. after the Transmit DMA channel (or software) writes data into the TxFIFO that decrements the TCC to zero. As noted in an earlier subsection, the Transmitter drops its DMA request from the time the DMA channel fetches the last character of a frame, until after it transfers the character to its serial shift register. It does this so that the DMA channel does not fetch the Transmit Control Block for the next frame or message, while the Transmitter still needs the control information for the current frame.

Note that this list ***does not*** include hardware or software Reset. This means that after either kind of Reset, the Transmitter is not expecting a TCB. Software must issue one of the commands listed above to condition it to receive the TCB for the first transmit frame after a Reset.

Chapter 6 describes how the Z16C32's Transmit DMA channel can fetch a Transmit Control Block from either of two locations in memory. The first method is USC-compatible: the channel fetches the TCB from the memory data buffer, before fetching the first characters of the frame or message. The other method is new with the Z16C32, and applies only when the Tx DMA channel is in "Array mode" or "Linked List mode." With this method, the channel fetches a TCB from the Array or Linked List entry for a buffer, if its start aligns with the start of a frame. For the transmit side the choice between these methods should be based on which is a better fit with the software I/O architecture.

5.19 DMA SUPPORT FEATURES (Continued)

When the Tx DMA channel reaches the end of an array or linked list that contains TCBs, it sends the "TCB words" from the terminating entry to the Transmitter before it recognizes the zero byte count in the entry and stops. This takes the Transmitter out of the state of expecting a TCB for the next frame. To overcome this potential problem, software must write one of the commands described in the preceding list (typically "Load TCC") to the CCAR, before restarting a Tx DMA channel in Array or Linked List mode with TCBs in the array/list entries.

IUSCs manufactured before August of 1992 did not fetch a TCB from the first entry of an array or linked list. For applications that might encounter such early devices, software can tell whether a given device fetches the first TCB as described in the 'Determining the Device Revision Level' section of Chapter 8.

Figure 5-17 shows the format of the first word of a 32-bit TCB or the only word of a 16-bit TCB. Its most significant four bits define a new TxSubMode value for the following transmit data. When the Transmit DMA channel or software writes this word to the TDR, the IUSC copies these four 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 IUSC similarly copies these bits into the TxResidue field of the Channel Command/Status Register (CCSR4-2) without affecting the rest of the CCSR. The device ignores bits 11-5 and 1-0 of the first or only word of a TCB, but Zilog reserves these bits for future enhancements and software should ensure that they are all zero.

For most protocols, the second word of a 32-bit TCB should contain the number of characters/bytes in this frame or message. The IUSC writes this word through the Transmit Count Limit Register (TCLR) and into the Transmit Character Counter (TCCR). As noted in the earlier section on the Transmit Character Counter, the TCC is loaded from the TCLR only when software writes one of three commands to the device, or when the second word of the 32-bit TCB is fetched. This means that if software wants the hardware to handle multiple frame transmissions without software intervention, 16-bit TCBs are not useful.

Chapter 6 describes and shows the various cases of TCB placement in memory in DMA applications.

5.19.4 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, and HDLC/SDLC. In these modes the Receiver can provide summary/status information after the frame or message. The **RxStatBlk** field of the Channel Control Register (CCR7-6) controls whether it does this. The IUSC interprets it as follows:

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 frame status as the first word of a 32-bit Receive Status Block, or the only word of a 16-bit RSB. Figure 5-18 shows this word, which is similar to but not identical with the contents of the Receive Command/Status Register (RCSR). The differences include:

1. The IUSC forces the bits that correspond to ExitedHunt and IdleRcved 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. Bit 5 of the RSB status is a copy of the RCCF Ovflo bit that is otherwise accessible as CCSR15. (Older IUSCs always store this bit as 0.) (**Note:** IUSCs that do not bear the 16C32 SL1660 topmarking always store this bit as 0.) Because RCCF Ovflo does not become 1 until an overwritten RCC residual value has come to the top of the RCC FIFO, a 1 in this bit indicates that the associated RCC residual value is not valid.

   When RCCF Ovflo is 1, if software has an alternative means of determining the length of the current frame, such as an embedded length field or fixed-length frames, it should use this alternative means to process the frame. Otherwise it must discard the frame as being unprocessable.

3. The LS bit of the first word of an RSB is a copy of the LS bit of the RCC at the end of the frame, rather than the RxAvail bit that is 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.

A 10 in RxStatBlk makes the IUSC also store the ending value of the Receive Character Counter in a second 16-bit word after the frame status word. This value indicates the length of the frame.

The IUSC takes the RCC value in a 32-bit RSB from the four-entry RCC FIFO, which with older IUSCs was only used by software. This allows up to four (short) frames to reside in the RxFIFO, without loss of information.

Note: Figure 5-14 illustrates that IUSCs that do not bear the 16C32 SL1660 topmarking save this RCC value in a 16-bit latch called RCHR, that is not directly accessible to software. Unfortunately this latch does not provide much buffering capacity when successive short frames are received. To write software that is compatible with both kinds of devices, either enable 32-bit RSBs or read the RCC FIFO, ***but not both!***

5.19.5 Storing the RSB

Chapter 6 describes how the Receive DMA channel can store a Receive Status Block in memory in two different ways. With the USC-compatible method, the DMA channel does not handle the RSB in any special way, it simply stores it in the memory buffer after the RxBound character, and decrements its Receive Byte Count Register (RBCR) as for serial data.

The other method is new with the Z16C32, and assumes the following circumstances:

1. the Receive DMA channel is in Array or Linked List mode, and either

2a. the channel's Early Termination feature is enabled, or

2b. the line protocol uses a fixed frame length and memory buffers are of this length as well.

With this method, when an Rx frame ends the Receive DMA channel stores the RSB in the Array or List entry for the terminated buffer, before going on to the next one. The channel does not decrement its byte count as it transfers this data.

The problem with the USC-compatible method is that software has to know how long each received frame is, in order to find its RSB. To obtain these lengths it has to read the RCC FIFO in a sufficiently timely manner to prevent overflows. For four or more successive frames each composed of, say, 4-6 characters, the four-entry depth of the RCC FIFO may impose interrupt-response requirements that can not be met in the worst case. Since the IUSC obtains the second word of a 32-bit RSB from the RCC FIFO, applications that use the IUSC's Rx DMA channel can only use 16-bit RSBs with this method.

By contrast, storing the RSB's in Array or Linked List entries allows software to ignore the receive process for longer periods, these being limited only by the extent of the Array or List structures it sets up, and/or by response timeouts imposed by the serial protocol.

2ndBE	1stBE	00		RxResidue			ShortF/ CVType	000		RCCF Ovflo	Always 1	CRCE /FE	Abort /PE	Rx Over	RCC0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 5-18. The First (or Only) 16 Bits of a Receive Status Block

5.19.6 Finding the End of a Received Frame

When software or the Receive DMA channel reads 16 bits from the RDR, and the Receiver has marked the oldest character in the RxFIFO with RxBound status, the IUSC only takes that one character out of the RxFIFO. When the Receive DMA channel is doing 16-bit transfers, software has to figure out whether the 16 bits preceding the RSB contain one or two characters/bytes, as follows:

1. 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.)

2. If the frame or message occupies more than one buffer, subtract from the frame length, the length of all of the buffers except the last one.

3. To the result from 1 or 2, add the starting address of the last buffer.

If the result is odd there is one character in the 16-bit word that precedes the RSB, while if it is even there are two characters in the word.

Another method applies only when bits 2-1 of the first word of the RSB, namely Abort/PE and RxOver, are both 0. The usual handling for a receive overrun condition in synchronous modes includes 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 Abort/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 in the word.

Chapter 6 describes the various ways in which the Receive DMA channel can store an RSB in memory.

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 5-19. The Channel Command/Address Register (CCAR)

5.20 COMMANDS

Commands are encoded values that software writes to a register field to change the state of the IUSC or make it perform some action. Typically commands do not take any software-perceptible time to perform. IUSC 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 the IUSC includes three separate encoded command fields in its serial section and one in its DMA section! Figure 5-20 shows the Channel Command/Address Register. Software can write various commands that affect the Transmitter and/or the Receiver to its **RTCmd** field (CCAR15-11). In addition, software can write commands that affect the Transmitter to the **TCmd** field in the Transmit Command/Status Register (TCSR15-TCSR12), and can write commands that affect the Receiver to the **RCmd** field in the Receive Command/Status Register (RCSR15-12). Chapter 6 describes the commands for the IUSC's DMA channels that software can write to the DMA Command/Address Register.

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 TxFIFO
01011	Purge Rx and TxFIFO
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 LS Bit First
10101	Select Serial MS Bit First
10110	Select D15-D8 First
10111	Select D7-D0 First
11001	Purge Rx

TCmd Value	Function
0010	Clear Tx CRC Generator
0100	Select TICRHi=TTSA Data
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
0100	Select RICRHi=RTSA Data
0101	Select RICRHi=FIFO Status
0110	Select RICRHi=/INT Level
0111	Select RICRHi=/RxREQ Level

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

Clear EOF/EOM (TCmd:=1110): this command conditions the IUSC so that it does not mark the next character, that software or the Transmit DMA channel writes to the Transmit Data Register, as End of Frame/End of Message. Since the IUSC 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 (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 CRC generators at the start of each frame.

5.20 COMMANDS (Continued)

Disable DLE Insertion (TCmd:=1101): this command applies only to Transparent Bisync mode. It conditions the IUSC so that it does not check subsequent characters written to the Transmit Data Register (TDR) for DLE characters, and so that it does not 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 an 'Enable DLE Insertion' command or until a hardware or software Reset. The IUSC queues the state that is 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 the IUSC 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 a Disable DLE Insertion command. The IUSC queues the state that is affected by this and the preceding command through its TxFIFO with each character, so that software can change it 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. In any mode, the Receiver discards any partial character that was in progress when software issued the command.

Load RCC and/or TCC (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). If 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, loading a BRG via one of these commands also enables it to count. See Chapter 4 for more information about the BRG's.

Purge Rx (RTCmd:=11001): on IUSCs manufactured after February 1994, this command purges (clears, empties) both the main RxFIFO and the RCC FIFO described in an earlier section. It combines the functions of the ClearRCCF bit in the Channel Command/Status Register (CCSR13) and the Purge RxFIFO command described below, including the latter command's function of reloading the RCC. This command is intended to be used after an Enter Hunt mode command in handling an Rx Overrun condition, and ensures that the two FIFOs are synchronized with respect to End-Of-Frame conditions.

Software can use the device-identification features described in 'Determining the Device Revision Level' in Chapter 8, to determine whether it can issue this command, or whether it has to issue the two separate commands noted above.

Purge Rx and/or TxFIFO (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 TxFIFO command also conditions the Transmitter to treat the next data written to the Transmit Data Register as a TCB. If software is using the Transmit DMA channel, a Purge TxFIFO command may cause the /TxREQ pin to be asserted immediately, while if it is using Transmit Data interrupts, the command may cause the /INT pin to be asserted immediately. (The previous two sentences also apply to a Purge Rx and TxFIFO command.)

On IUSCs that do not bear the 16C32 SL1660 topmarking the Purge Rx FIFO and Purge Rx and Tx FIFO commands did not clear a hidden "holding register" that stands between the Rx shift register and the RxFIFO. This caused problems if a very fast processor responded to an HDLC Abort interrupt and performed a Purge Rx FIFO command before the final character before the Abort was transferred from the holding register to the RxFIFO. In this case, this character would then go into the RxFIFO and finally emerge as an extraneous 1-character "frame". IUSCs with the SL1660 topmarking deals with this problem in two separate ways: 1) they do not set the Break/Abort flag until after the character before the Abort is in the RxFIFO, and 2) they clear the holding register as a result of either this command or a Purge Rx command. Thus they will never produce such a 1-character frame, regardless of how fast the processor may be.

Reset Highest Serial IUS (RTCmd:=00010): Chapter 7 describes how this command clears the highest-priority Interrupt Under Service latch in the serial controller section that is currently set (if any).

Select D15-D8 or D7-D0 First (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 IUSC arranges the oldest and second-oldest characters in the RxFIFO when software or the Receive DMA channel reads 16 bits from the Receive Data Register. "D15-D8 First" is the default value after either a hardware or programmed reset, and is compatible with the Zilog Z8000™ Zilog Z16C0x and Motorola 680x0 processors. "D7-D0 First" should be programmed for the Zilog Z380™ and most Intel processors. The IUSC applies this option only during a 16-bit transfer, between the TxFIFO or RxFIFO and the AD15-AD0 pins. However, if the Transmit Character Counter contains 0001 and the Transmit DMA channel writes 16 bits to the TxFIFO, the IUSC only puts the character from AD7-AD0 in the TxFIFO, regardless of these commands. In a "D7-D0 First" system this is not a problem. But if the last character of a frame or message falls at an even address when using the Transmit DMA channel in a "D15-D8 First" system, software must copy the last character into the subsequent odd address as well.

Select RICRHi=/INT Level (RCmd:=0110): this command conditions the IUSC so that subsequent accesses to the MSbyte of the Receive Interrupt Control Register (RICR15-8) read or write the number of received characters at which the IUSC starts requesting a Receive Data interrupt, as described in Chapter 7. If software uses the Receive DMA channel to store data in memory, it should disable Receive Data interrupts.

Select RICRHi=/RxREQ Level (RCmd:=0111): this command conditions the IUSC so that subsequent accesses to the MSbyte of the Receive Interrupt Control Register (RICR15-8) read or write the number of received characters at which the Receiver asserts /RxREQ to the Receive DMA channel, as described in Chapter 6.

Select RICRHi=FIFO Status (RCmd:=0101): this command conditions the IUSC so that reading the MSbyte of the 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 RICRHi=RTSA Data (RCmd:=0100): this command conditions the IUSC so that subsequent accesses to the MSbyte of the Receive Interrupt Control Register (RICR15-8) read or write Receive Time Slot Assigner data. This is described more fully in *Programming the Time Slot Assigners* in Chapter 4.

Select Serial Data LSB or MSB First (RTCmd:= 10100-10101): these commands control whether the IUSC 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. The IUSC applies this option as it transfers data between the AD pins and the FIFOs. Because of this, these commands do not 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 the IUSC 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 7. If software uses the Transmit DMA channel to fetch data from memory, it should disable Transmit Data interrupts.

Select TICRHi=/TxREQ Level (TCmd:=0111). This command conditions the IUSC so that subsequent accesses to the MSbyte of the Transmit Interrupt Control Register (RICR15-8) read or write the number of empty TxFIFO entries at which the Transmitter asserts /TxREQ to the Transmit DMA channel, as described in Chapter 6.

Select TICRHi=FIFO Status (TCmd:=0101): this command conditions the IUSC so that reading the MSbyte of the 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.

Select TICRHi=TTSA Data (TCmd:=0100): this command conditions the IUSC so that subsequent accesses to the MSbyte of the Transmit Interrupt Control Register (TICR15-8) read or write Transmit Time Slot Assigner data. This is described more fully in *Programming the Time Slot Assigners* in Chapter 4.

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 two MS bits of the TxSubMode field of the Channel Mode Register (CMR15-14) are 01, the Abort consists of a 0 followed by 15 consecutive ones. Otherwise it consists of a 0 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 is data in the TxFIFO, it starts a new frame, otherwise it sends the Idle condition defined by the TxIdle field (TCSR10-8).

5.20 COMMANDS (Continued)

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 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.

This command also releases the interlock that occurs if software sets the UnderWait bit (TCSR11) to 1, and a Tx Underrun condition occurs. See the section 'Handling Overruns and Underruns' later in this chapter.

In any case, this command releases an interlock that is established after frame transmission, and is never needed before the first frame after a Reset.

Set EOF/EOM (TCmd:=1111): this command conditions the IUSC so that it marks the next character, that software or the Transmit DMA channel 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 IUSC automatically clears the state set by this command when software (or the Transmit DMA channel) writes to the TDR. Therefore this command applies to at most one character.

Trigger Channel Load DMA (RTCmd:=00100): Chapter 8 will describe how this command puts the serial controller section of the IUSC in a special mode, in which the Transmit DMA channel can initialize all the registers in the serial controller. Software must program and set up the Transmit DMA channel as for transmitting data, before it issues this command. This operation can not initialize any of the registers in the IUSC's DMA section.

Trigger Rx and/or Tx DMA (RTCmd:=00101-00111): if one of the Wait4xxTrig bits in the Channel Control Register (CCR13 for Tx, CCR5 for Rx) is 1, the serial controller section of the IUSC 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 one or both DMA channel(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.

The two commands above release interlocks that occur at the end of a frame, and are never needed before the first frame after a Reset.

5.21 RESETTING THE SERIAL CONTROLLER

Figure 5-20 shows the **RTReset** bit in the Channel Command/Address Register (CCAR10). Software can use this bit to reset the serial controller section of the IUSC to a known and inactive state like that produced by driving the /RESET pin low. (The most significant difference is that the IUSC requires software to write the Bus Configuration Register (BCR) after a hardware reset, but not after this kind of "software Reset.")

To software-reset the serial controller on a 16-bit data bus:
1. Write CCAR (or its MS byte) with RTReset=1.
2. Write a 16-bit zero to CCAR.

To software-reset the serial controller on an 8-bit bus:
1. Write the MS byte of CCAR with RTReset=1.
2. Write the LS byte of CCAR with an 8-bit zero.
3. Write the MS byte of CCAR with an 8-bit zero.

The way this "software reset" works is that the 1 state of RTReset conditions the serial controller's register address decoding logic so that the subsequent write operation actually writes data into all the registers in the serial controller. Between the time that software writes RTReset as 1, and when it writes it back to 0, the IUSC does not drive I/O pins, it either tri-states output pins or holds them in their inactive state, but register bits that do not 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.

5.22 THE DATA REGISTERS AND THE FIFOS

When the RxFIFO contains received characters, software can read the "oldest" one or two characters in it from the Receive Data Register (RDR). When software uses the Receive DMA channel, the channel takes care of taking data out of the RxFIFO, in a "flyby" fashion using an internal "RxACK" signal. 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 eight bits long.

Similarly, when the TxFIFO is not full software can write one or two characters to the Transmit Data Register (TDR), or the Transmit DMA channel can write the TxFIFO in a flyby fashion using an internal "TxACK" signal.

5.22.1 Accessing the TDR and RDR

Chapter 2 describes how software can access the TDR and RDR using a register address that may be 1) multiplexed on the AD6-AD0 pins, 2) full-time on AD13-AD8 if only AD7-AD0 carry data, or 3) written into the Channel Command/Address Register (CCAR6-0).

Two other features of the IUSC make it easier for software to access these registers when the AD lines do not carry multiplexed addresses and the data bus is 16 bits wide. Host processor write cycles to the IUSC, with the S//D and D//C pin both high, always write the TDR. Similarly, host processor read cycles from the IUSC, with S//D and D//C both high, always read the RDR. A system designer can connect S//D and D//C to processor address lines, such as A2 and A1 for a non-multiplexed 16-bit bus, or A8 and A7 for a multiplexed bus. (If an application wants to have the IUSC drive cycle-type status onto S//D and D//C when it is the bus master as described in Chapters 2 and 6, external tri-state gates are needed to drive address bits onto these pins only when the IUSC is not in control of the bus.)

Chapter 2 also describes how to write the Bus Configuration Register to configure the IUSC for a 16-bit data bus. With a 16-bit data bus, software can write two characters at once to the TDR, or the Transmit DMA channel can read two characters out of memory at once. Similarly, software can read two characters at a time from the RDR, or the Receive DMA channel can write two characters into memory in each bus cycle. The earlier section *Commands* describes how the "Select D15-D8 First" and "Select D7-D0 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 IUSC 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 Z16C0x or Motorola 680x0 based system this means that software should write bytes to the TDR and read bytes from the RDR at odd addresses. On a Zilog Z380™ or Intel 80x86 processor, software should write bytes to the TDR and read bytes from the RDR at even addresses.

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

5.22.2 TxFIFO and RxFIFO Operation

The TxFIFO and RxFIFO have a maximum capacity of 32 characters (bytes) each. The IUSC empties them of all data when external hardware drives the /RESET pin low, when software resets the serial controller via the RTReset bit (CCAR10), and when software writes a "Purge Rx" or "Purge Rx and/or TxFIFO" 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 reads data from it via the RDR or the Rx DMA channel writes data into memory. The TxFIFO becomes one or two bytes more full each time software writes data to the TDR or the Tx DMA channel reads data from memory, and one byte less full each time the Transmitter moves a character into its output shift register.

One further point about RxFIFO operation applies only in HDLC/SDLC, HDLC/SDLC Loop, 802.3, and Transparent Bisync. In one of these modes, if software or the Rx DMA channel reads 16 bits from the RDR when the oldest character in the RxFIFO is the last one of a frame (i.e., it is marked with RxBound status), the IUSC removes only that one character from the RxFIFO.

5.22.3 Fill Levels

The IUSC maintains a "Fill Level" 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 has not 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 on the Receiver side, software can skip the step of writing the Select command if it has not written any of the other "Select TICRHi" commands to the TCSR since the last time it issued this command.

Code that reads a FIFO Fill level must ensure that no interrupts will occur between the time it writes the "Select xICRHi=FIFO Status" command to the TCSR or RCSR, and when it reads the value from the TICR or RICR, if such interrupts can lead to other code writing a different Select command (for a Time Slot Assigner or threshold) to the same Command/Status Register.

Large values of the FIFO Fill Levels indicate exceptional conditions. 33_{10} (21_{16}) in the Rx Fill Level indicates that data has been lost because of a Receive Overrun condition. Rx Fill Level values above that, particularly 63_{10} ($3F_{16}$), indicate that software read more data from the RxFIFO than was received. Tx Fill Levels between 33_{10} (21_{16}) and 63_{10} ($3F_{16}$) inclusive indicate that software wrote more data to the TxFIFO than there was room for. All of these situations should be handled by issuing a Purge FIFO command, although receive software may want to handle an Overrun by reading out the FIFO first, to salvage data received before the problem occurred.

5.22.4 DMA and Interrupt Request Levels

The IUSC continually compares the contents of the Fill Level counters against two "threshold" levels for each. Chapter 6 describes how the "Tx DMA Request Level" determines how empty the TxFIFO must get before the Transmitter starts requesting that the Transmit DMA channel should read more data from memory. Once the Transmitter has started to request DMA transfer, it typically keeps doing so until the DMA channel has filled the TxFIFO or until the Transmit Character Counter has counted down to zero.

Chapter 6 also describes how the "Receive DMA Request Level" controls how full the RxFIFO should get before the Receiver starts requesting that the Receive DMA channel should move data to memory. Once the Receiver has started to request DMA transfer, it typically keeps doing so until the DMA channel has emptied the RxFIFO, or until it has stored the last character of a frame or message.

Chapter 7 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 does not use these kinds of interrupts in most IUSC applications, because the Transmit and Receive DMA channels 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.)

5.22.5 Fill Level Correctness and Reliability

With the Z16C31 and other older members of the USC family, certain worst-case interarrivals of serial clocking and bus timing could result in transient states in which the RxFIFO and TxFIFO counts were incorrect. When software read these counts and transferred data to the TDR or from the RDR, it could work around such problems by the classic data-acquisition technique of reading a count until two successive readings agreed. The Z16C32 includes logical interlocks so that these counts will always be correct and need only be read once.

These interlocks have also eliminated a related problem of earlier USC family members, wherein a received character was completed just as the Receiver was deciding to withdraw its Receive DMA request because the Rx DMA Channel had emptied the RxFIFO. Under worst-case interarrivals, the logic would maintain the request on a 16-bit bus even though the RxFIFO contained only the single newly-received character. The DMA channel would then do a 16-bit transfer, so that the observable symptom of the problem was that occasionally, "extra characters" would appear in the received frame in memory. Such phenomena will not occur with the Z16C32.

5.23 HANDLING OVERRUNS AND UNDERRUNS

In general, both the Tx Underrun condition in the TCSR and the Rx Overrun condition in the RCSR should be enabled and armed for interrupt. While the IUSC can handle most things that can arise in normal operation in a fairly automatic fashion, these two conditions represent a breakdown in the relationship between the IUSC and its environment, namely insufficient granting of access to the bus and memory. Software should respond to them quickly to minimize further loss of received data and to prevent erroneous transmission.

5.23.1 Tx Underruns

All revisions of the IUSC will deal with a Transmit Underrun condition in synchronous modes by setting the TxUnder bit in the TCSR, and by concluding the current frame or message as specified in the TxSubMode field of the Channel Mode Register (CMR), which may have been set from a Transmit Control Block.

But if the Tx DMA channel then belatedly responds to the Transmitter's DMA Request and puts more data in the TxFIFO, before software can respond to the Underrun condition, the Transmitter can begin sending a new frame, typically starting with data that was meant to be in the middle of a frame.

If an application is subject to Tx Underruns and has response latency to the Underrun condition that allows such a subsequent frame to be started out onto the serial link, the only practical way to avoid this behavior is to set the Wait2Send bit (TICR2) and have software issue a Send Frame/Message command to allow each Tx frame out onto the link. This procedure may degrade transmit performance.

Software can avoid both the problem and the performance degradation associated with the workaround described above, by setting the **UnderWait** bit in the Transmit Command/Status Register (TCSR11), which was Reserved in previous revisions. When UnderWait is set, the IUSC's Transmitter will wait after dealing with a Transmit Underrun condition, sending the Idle condition specified in the TCSR, until software recognizes the Underrun condition and deals with it.

The recommended software response is to:

1. write the "Pause Tx DMA" command to the DCAR,
2. write the "Purge Tx FIFO" command to the CCAR,
3. reprogram the Tx DMA channel to the start of the frame in which the underrun occurred,
4. write the "Start Tx DMA" or "Start/Init Tx DMA" command to the DCAR,
5. write the "Send Frame/Message" command (plus the UnderWait bit) to the TCSR. This command releases the interlock caused by the underrun condition with UnderWait=1.
6. If the Underrun condition is armed for interrupt, write a 1 to TCSR1 to clear the status bit.
7. If Underrun and other conditions are armed to cause Transmit Status interrupts, clear all the IA bits in the TICR and then restore the ones you want.

When 32-bit Transmit Control Blocks are used, setting UnderWait to 1 has a further effect that helps minimize the occurrence of Tx Underrun conditions. When the TxCtrlBlk field (CCSR15-14) is 10 and UnderWait (TCSR11) is 1, the Transmitter will delay starting to send a frame until either the TxFIFO is full or an entire Tx frame has been placed in the TxFIFO.

Using UnderWait with 32-bit TCBs helps minimize Tx underruns in situations such as when the Rx DMA channel has pre-emptive priority over the Tx DMA channel, and it seizes control of the bus just after the channel has placed the first one or two characters of a new Tx frame in the TxFIFO.

5.23.2 Rx Overruns

If the external processor or arbiter doesn't grant the bus to an IUSC sufficiently soon/often, the 32-character RxFIFO may fill up. If another character arrives while the RxFIFO is full, the Receiver saves this character in a holding register between the Rx shift register and the RxFIFO. When the Rx DMA gets around to reading from the RxFIFO again, the Receiver places this "overrun character" in the RxFIFO with a status bit that accompanies it through the FIFO. When the Rx DMA channel stores the overrun character in memory, the IUSC sets the RxOver bit in the RCSR and requests an interrupt if the RxOver IA bit in the RICR and the RSIE and MIE bits in the ICR are all 1.

Once an overflow has occurred, the Receiver doesn't put any more received data in the RxFIFO (even if the external processor/arbiter grants the bus and the Rx DMA channel stores some or all of the data from the FIFO into memory) until software responds. The proper software response is to:

1. Write a "Pause Rx DMA" command to the DCAR
2. Write an "Enter Hunt Mode" command to the RCSR
3. Write a "Purge Rx" command to the CCAR. IUSCs that do not bear 16C32 SL1660 topmarking always store this bit as 0) write a "Purge Rx FIFO" command to the CCAR and a 1 to the "Clear RCCF" bit in the CCSR.
4. Reprogram the Rx DMA channel to point to the first buffer for the frame in which the overrun occurred.
5. Write a "Start Rx DMA" or "Start/Init Rx DMA" command to the DCAR
6. If the Overrun condition is armed for interrupt, write a 1 to RCSR1 to clear the status bit.
7. If Overrun and other conditions are armed to cause Receive Status interrupts, clear all the IA bits in the RICR and then restore the ones you want.

5.23.3 Rx Overrun Scribbling

IUSCs that do not bear the 16C32 SL1660 topmarking have a special problem with Rx Overruns. When the end of a frame or message arrives, the IUSC sets an internal state that forces the Rx DMA request to store the end of the frame. Normally, this state is cleared when the Rx DMA channel stores the last character of the frame in memory. However, if the frame ends while the Receiver is overrun, the logic sets the internal state as usual, but there's nowhere to store the EOF character that will clear this state. The result is that the Rx DMA channel keeps storing the entire contents of the RxFIFO again and again in memory, until it runs out of buffers or until software detects the overrun condition and stops the scribbling by the steps described above.

However, the scribbling activity itself handicaps the processor from executing the interrupt service routine efficiently until the Rx DMA channel runs out of buffers. If it's important to stop the scribbling ASAP:

1. Use the Burst/Dwell Control Register (BDCR) to limit the IUSC's activity in each period of bus control.
2. Set the MinOff39 bit (DCR5) to keep the IUSC from re-requesting the bus quickly.
3. Give interrupts from the IUSC the highest possible priority.

The first two steps should allow the processor enough bandwidth to slowly execute the ISR and terminate the scribbling.

5.24 BETWEEN FRAMES, MESSAGES, OR CHARACTERS

5.24.1 Synchronous Transmission

When software issues a "Set EOF/EOM" command and then writes data to the TDR, or when the TCC is enabled and software or the Transmit DMA channel fetches enough data so that it counts down to zero, the IUSC flags the last character of the message or frame in the TxFIFO. After this last character passes through 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 is empty (an "underrun" condition). The IUSC's exact actions at these points depend on the serial mode/protocol and possibly 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.

After sending a CRC for either reason, or right after the last character from the TxFIFO if it does not 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 right 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:

1a. The UnderWait bit (CCSR11) is 0 and/or the TxCtrlBlk field (CCR15-14) is 0x, and there is at least 1 character in the TxFIFO, or

1b. UnderWait is 1 and TxCtrlBlk is 10, and the TxFIFO is full or a complete frame has been placed 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 are not 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 does not 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 4 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, or, with the Z16C32, to guarantee a minimum number of Flags between HDLC/SDLC frames. 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	eight bits
01	16 bits
10	32 bits
11	64 bits

TxPrePat	Preamble Pattern Sent
00	All zeroes
01	All ones, or Flags
10	101010
11	010101

For HDLC/SDLC mode, if TxPrePat is 01 and the **FlagPreamble** bit in the Channel Control Register (CCR12, see Figure 5-17) is 1, the Z16C32 sends 1, 2, 4, or 8 Flags as the Preamble. Including the opening and closing Flags, this guarantees a minimum of 3, 4, 6, or 10 Flags between frames respectively. This is useful when sending to certain kinds of equipment that can not handle less Flags, or as a means of slowing down the gross frame rate slightly.

FlagPreamble should be 0 in all other modes. For 802.3 (Ethernet) mode, program 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 4 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 software has not selected sending a Preamble (CMR13=0); this does not apply in Loop mode), and the conditions for starting a frame (described earlier in this section) are met 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 7 describes how software can enable any or all of these conditions to cause an interrupt.

5.24.2 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.

5.24.3 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 covered 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 4 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 7 describes how software can enable the RxBound, IdleRcved, and/or Exited Hunt conditions to cause an interrupt.

The IdleRcved logic is not 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).

5.25 SYNCHRONIZING FRAMES/MESSAGES WITH SOFTWARE RESPONSE

In some applications, software can simply set up DMA buffers for multiple frames or messages, and set the IUSC's Transmitter and/or Receiver and DMA channel(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 IUSC 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. Note that all three of these interlocks apply only after the end of a frame, not before the first frame sent or received.

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 the Transmit DMA channel (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 Transmit DMA channel will not 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 channel, although the DMA section provides three different modes that do this more efficiently.

On the Receive side, if the **Wait4RxTrig** bit in the Channel Control Register (CCR5) is 1, then after the Receive DMA channel 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 does not assert /RxREQ to the Receive DMA channel 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 channel between frames.

CHAPTER 6
DIRECT MEMORY ACCESS (DMA) CHANNELS

6.1 INTRODUCTION

The main advantage of the IUSC, compared to predecessor devices like the Z16C3x MUSC, is the inclusion of Transmit and Receive DMA channels. These allow the IUSC to fetch its own transmit data from memory and store its received data in memory. This chapter describes the various operating modes of these DMA channels and how to program them.

The IUSC's Receiver and Transmitter can be handled via DMA or programmed transfers. Software can even mix DMA and programmed transfers for the Receiver or the 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. Software can then determine the kind of frame from examining the first characters, optionally program the Rx DMA controller accordingly, and then write the "Trigger Rx DMA" command to the RTCmd field of the Channel Command/Address Register (CCAR15-CCAR11). The DMA controller can then transfer the rest of the frame into memory without further software intervention.

6.2 DMA FUNDAMENTALS

Each channel can operate in any of four main operating modes. Figure 6-1 shows the format of the Transmit and Receive DMA Mode Registers (TDMR and RDMR). The **DMAMode** fields of these registers control the main mode of each channel, and are encoded as follows:

DMAMode	Basic DMA Mode
00	Single Buffer
01	Pipelined
10	Array
11	Linked List

Later sections will describe each of these modes in detail, but first it is worthwhile to present some characteristics that are common to all the modes.

6.2.1 Addresses and Byte Counts

Before the Transmit DMA channel can transfer data from a memory buffer to the TxFIFO, and before the Receive DMA channel can transfer data from the RxFIFO to a memory buffer, software and/or hardware (depending on the mode) has to load the data buffer's starting address into the **Transmit or Receive Address Register (TAR or RAR).** The same software/hardware mechanism has to load the number of bytes to be read out of the buffer into the **Transmit Byte Count Register (TBCR)**, or load the (maximum) number of bytes to be written into the buffer into the **Receive Byte Count Register (RBCR)**. The TAR and RAR are 32-bit registers, allowing the IUSC to address up to a 4-Gbyte linear address space, while the TBCR and RBCR are 16-bit registers, allowing a channel to transfer up to 65,535 bytes to or from each buffer. (In Single-Buffer and Pipelined modes, a zero byte count makes a channel do nothing, while in Array and Linked List modes, a zero byte count indicates that the last buffer of the array or list has been completed.) In any mode, a block of data longer than 65,535 bytes can be easily transferred, by treating it as two or more consecutively-addressed buffers.

The 32-bit TAR and RAR are each divided into two 16-bit registers, with the less significant half being called Lower (TARL, RARL) and the more significant half being called Upper (TARU, RARU), In Single-Buffer and Pipelined modes, software must program address registers directly; they are arranged with the Lower register at the lower register address, which sounds right but is in fact the natural order only for Little-Endian systems (Z80 family or 8086 family processors). On Big-Endian machines, including Z8000 and 680x0 processors, software should not program an address register using an instruction that moves 32-bit data, but rather by means of two separate instructions each transferring 16 bits.

6.2 DMA FUNDAMENTALS (Continued)

Aside from certain "overhead" operations in Array and Linked List modes, each DMA channel actually transfers data only when the serial Receiver or Transmitter requests that it do so, using an internal request signal. *Programming the DMA Request Levels*, later in this chapter, describes how software can program the number of received characters in the RxFIFO at which the Receiver requests DMA transfer, and the number of empty slots in the TxFIFO at which the Transmitter does so.

DMAMode	TCB /RSB InA/L	Clear Count	AddrMode	TermE	8/16	CONT	GLink	BUSY	INITG	EOA/ EOL	EOB	HAbort	SAbort
15 14	13	12	11 10	9	8	7	6	5	4	3	2	1	0

Figure 6-1. The DMA Mode Registers (TDMR and RDMR)

6.2.2 Data Width and Byte Ordering

If "16-bit" in the Bus Configuration Register (BCR2) is 0, indicating an 8-bit external data bus, and/or if the **8/16** bit in a channel's DMA Mode Register (TDMR8 or RDMR8) is 1, the channel does only 8-bit transfers with memory, including data buffer transfers and "array" and "list" accesses in Array and Linked List modes. The channel decrements its Byte Count Register (TBCR or RBCR) by 1 for each transfer to or from a data buffer. Typically it also increments its address register by 1 for each byte transfer, although software can program a channel to keep a data buffer address constant, or decrement it. If 16-bit is 0, the IUSC transfers all bytes on the AD7-AD0 lines. If 16-bit and 8/16 are both 1, and address incrementing or decrementing is enabled, the Transmit DMA channel provides each byte on both halves of the data bus, while the Receive DMA channel alternates between taking a byte from AD15-AD8 and from AD7-AD0, as determined by bit 0 of its address register. Chapter 5 describes the "Select D15-D8 First" or "Select D7-D0 First" commands that software can write to the CCAR; these affect how the channel relates address bit 0 to AD15-AD8 and AD7-AD0.

If 16-bit is 1 and 8/16 is 0, a Z16C32 DMA channel will do 16-bit transfers whenever it can. This includes all array and list transfers in Array and Linked List modes, and all transfers to and from data buffers when the address in TAR or RAR is even and TBCR or RBCR contains 0002 or more. If the address in TAR or RAR is odd, and/or if the byte count in TBCR or RBCR is 0001, the Z16C32 will do a byte transfer with memory. (This can happen only for the first byte and last byte of a buffer.) In such transfers the Receive channel will take the byte from AD15-AD8 or AD7-AD0 according to bit 0 of its address register, interpreted according to any "Select D15-D8 First" or "Select D7-D0 First" command that software has written to the CCAR.

When an IUSC does a 16-bit transfer to or from a memory buffer it decrements TBCR or RBCR by 2, and typically increments its address register by 2. For serial data, the IUSC arranges the oldest and second-oldest characters from the RxFIFO on the AD15-AD0 lines, or routes the two characters on these lines into the TxFIFO, according to any "Select D15-D8 First" or "Select D7-D0 First" command that software has written to the CCAR.

There is one other feature of the Z16C32's byte/word switching mechanism that software needs to know about. When a channel is programmed for 16-bit transfers and for Early Termination as described in the next section, and the last character of a frame falls at an even memory address, the serial controller signals the DMA channel that the current transfer includes the last character of a frame, but it does not indicate whether this character is the first or second of the two characters in the transfer. That is, it does not tell the DMA channel whether or not to force a byte transfer. On the receive side this is not a big problem, because if such an end-of-frame character is the oldest one in the RxFIFO, the IUSC provides it on the "even-addressed" half of the data bus. On the transmit side this is not a problem in a Little-Endian system, because when the TCC contains 0001 the TxFIFO logic always takes the last byte from AD7-AD0, which is the even-addressed location in such systems. On the Transmit side in a Big-Endian system, software can avoid this situation by not programming the Transmit DMA channel for Early Termination, but rather setting the byte count for the last buffer of the frame to match the frame length used in the TCC.

6.2.3 Buffer Termination

A DMA channel transfers data from memory to the TxFIFO as the Transmitter requests it, or from the RxFIFO to memory as the Receiver requests such transfer, until one of the following occurs:

1. the channel decrements the count in TBCR or RBCR to zero, or
2. if the TermE bit in the DMA Mode Register (TDMR9 or RDMR9) is 1, and the serial controller signals for a "buffer termination," or
3. external hardware asserts the /ABORT input during a DMA transfer, or
4. host software writes one of the following commands to the DMA Command/Status Register (DCAR):

   "Reset This Channel",
   "Pause This Channel",
   "Abort This Channel",
   "Reset All Channels",
   "Pause All Channels", or
   "Abort All Channels".

When a channel stops because of item 3 or 4 above, it does so summarily, without any further actions. But when a channel terminates a buffer for reason 1 or 2, it attempts to go on to another buffer, except in Single Buffer Mode. In Pipelined mode, if software has provided the address and byte count of the next buffer, the channel continues on to transfer that buffer; otherwise it stops. In Array or Linked List, the channel tries to fetch the address and byte count of the next buffer from the array or list in memory; if it finds them it continues on to transfer that buffer, otherwise it stops.

Point 2 above notes that if the **TermE** bit in a DMA Mode Register (TDMR9 or RDMR9) is 1, the channel will terminate a memory buffer before it decrements its byte count (in TBCR or RBCR) to zero, if/when the serial controller asserts a termination signal. (If TermE is 0, the channel ignores the signal.)

The serial controller asserts the internal termination signal to the Transmit DMA channel only in synchronous modes. It does so as the DMA channel writes 1 or 2 characters into the TxFIFO, so that the Transmit Character Counter (TCC) is decremented to 0.

On the receive side, the Receiver forces the Request signal True to the Receive DMA channel, as/after it places an RxBound character in the RxFIFO in HDLC/SDLC, Ethernet/802.3 or Transparent Bisync mode. It does this to force the DMA channel to store the end of the frame or message, even though the number of received characters in the FIFO is less than the programmed threshold for DMA Requesting. The serial controller then maintains the request until the DMA channel stores the RxBound character (and the Receive Status Block if it is enabled) in memory. The serial controller asserts buffer termination as the DMA channel stores these last bytes. (Early termination signaling on the receive side is actually more complex than we need to know about at this point. The full story is told later in *Storing Receive Status Blocks.*)

The Receive Character Counter (RCC) feature can neither cause early buffer termination nor forcing of the internal DMA Request.

6.3 SINGLE BUFFER MODE

Figure 6-2 illustrates Single Buffer mode, which is the most basic of the IUSC DMA channels' major modes. Software loads the starting address of each memory buffer containing data to be transmitted into the Transmit Address Register (TAR). Similarly, it loads the starting address of each memory area, into which received data should be stored, into the Receive Address Register (RAR). The software also loads the number of characters to be transmitted from each memory area into the Transmit Byte Count Register (TBCR). Similarly, it loads the **maximum** number of received characters to be stored in each memory area into the Receive Byte Count Register (RBCR).

Then the host processor software enables the DMA channel for operation by writing a "Start This Channel" command to the DMA Command/Address Register (DCAR). Thereafter the DMA channel moves the data from memory to the TxFIFO or from the RxFIFO into memory, as described in *DMA Fundamentals* above.

Software can program the IUSC to request several kinds of interrupts at the end of the buffer. A DMA channel interrupt, an interrupt request from the serial controller, or both can be used to trigger host software response at appropriate points in the serial data stream. Alternatively, host software can periodically poll the DMA channel status in the DMA Mode Register (TDMR or RDMR) and/or the serial channel status to determine when the DMA transfer is over.

Note that for transmitting, the DMA channel completes its operation before the serial Transmitter has finished sending all the data in the block. For reception the serial Receiver may know about an end-of-block situation before the Receive DMA channel has finished transferring the data into memory.

When an interrupt or polled status has informed the host software that a DMA block transfer is over, the software can read back the ending contents of the TAR or TBCR to figure out whether all of the bytes to be sent actually were sent. Similarly, software can read back the ending contents of the RAR or RBCR to determine how many bytes the channel stored in memory. Note that software can read similar information from the RCC FIFO in the serial controller, or can have the IUSC store it in memory in a Receive Status Block.

Particularly for receiving, host software will typically want to reprogram the RAR and RBCR, or TAR and TBCR, and restart the channel for the next buffer of data, as soon as possible after the DMA channel finishes with each buffer.

In many applications, data from two or more memory areas must be sent without interruption on the serial link (e.g., in the same frame). The corresponding characteristic on the receive side is almost always required, namely that received data not be lost while the host software responds to a buffer-complete condition, reprograms the channel for the next buffer, and restarts the channel.

While the IUSC's deep FIFOs provide some assurance of continuous transmission and protection against loss of receive data, above a certain bit rate these characteristics can only be assured by using Pipelined, Array, or Linked List mode. The actual rate at which Single-Buffer mode is no longer sufficient is a fairly complex matter involving processor speed and system architecture.

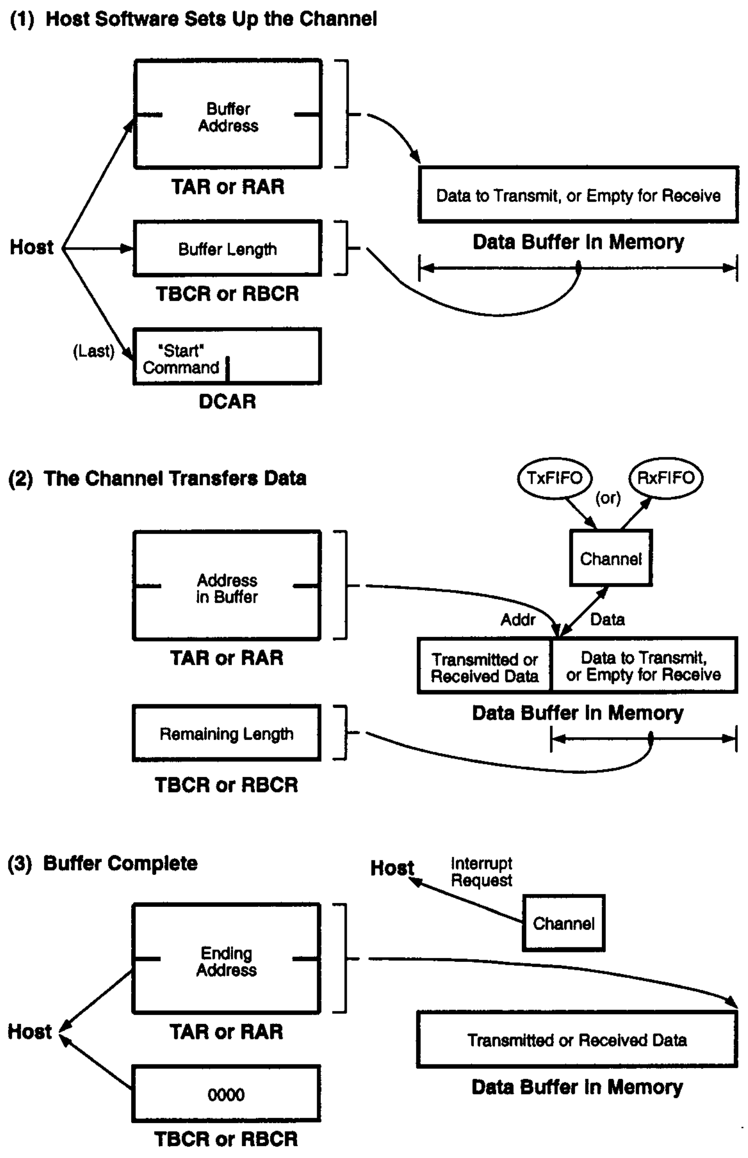


Figure 6-2. Single Buffer Mode DMA Operation

6.4 PIPELINED MODE

In this mode the IUSC employs two additional registers for each channel, called the Next Transmit Address Register (NTAR), the Next Transmit Byte Count Register (NTBCR), the Next Receive Address Register (NRAR), and the Next Receive Byte Count Register (NRBCR). Figure 6-3 illustrates Pipelined mode, in which software can write the starting address and byte count for the next data buffer into these registers, while the DMA channel is using the TAR and TBCR, or RAR and RBCR, to transfer the preceding buffer.

After programming a Channel Mode Register for Pipelined mode, the host software can start the channel in one of two ways. It can program the address and length of the first buffer into the TAR and TBCR, or RAR and RBCR, and then write the "Start This Channel" command to the DCAR. Alternatively, software can also write the address and length of the second buffer into the NTAR and NTBCR, or NRAR and NRBCR, and then write the "Start/Continue This Channel" command to the DCAR. The latter command differs from the former in that, in addition to setting the BUSY bit in the channel's DMA Mode Register (TDMR5 or RDMR5), it also sets the CONT bit (TDMR7 or RDMR7).

Whichever way software starts the channel, it then transfers from the data buffer indicated by TAR and TBCR, or into the buffer indicated by RAR and RBCR, as described earlier in *DMA Fundamentals.*

If a new transmit buffer is available in Pipelined mode and the CONT bit is zero, while the Transmit DMA channel is still transferring an earlier buffer, software should write the address and byte count of the new buffer into the NTAR and the NTBCR, and then write a "Start/Continue This Channel" command for the Transmit DMA channel into the DCAR. If it accomplishes these things before the DMA channel finishes transferring the preceding buffer from memory to the TxFIFO, then when the channel finishes with the preceding buffer, it automatically transfers the contents of the NTAR and NTBCR to the TAR and TBCR respectively, and continues sending the data in the new buffer.

Similarly, if an empty receive buffer is available and the CONT bit is zero, while the Receive DMA channel is still transferring an earlier buffer, software should write the address and byte count for the buffer into the NRAR and the NRBCR, and then write the "Start/Continue This Channel" command for the Receive DMA channel into the DCAR. If it accomplishes these steps before the channel finishes transferring the preceding buffer from the RxFIFO to memory, then when the DMA channel finishes with the preceding buffer, it automatically transfers the contents of the NRAR and NRBCR to the RAR and RBCR respectively, and goes on to receive data into the new buffer.

In Pipelined mode, a DMA channel tries to advance to the next buffer when it has decremented the TBCR or RBCR to 0, and/or if software enables the early buffer termination feature and the serial controller signals for termination. In either case the channel does so only if the CONT bit in its DMA Mode Register (TDMR7 or RDMR7) is 1. The channel sets the CONT bit when software writes a "Start/Continue" command to the DCAR, and clears the bit each time it advances to a new buffer.

A DMA channel will not advance to the next buffer in response to assertion of the /ABORT signal during a transfer. Nor will it advance to the next buffer in response to any software commands.

As in Single Buffer mode, software can program the IUSC to request a DMA channel interrupt and/or a serial controller interrupt as these modules finish with each buffer. Alternatively, host software can periodically poll the DMA channel status and/or the serial channel status to track the progress of DMA transfer.

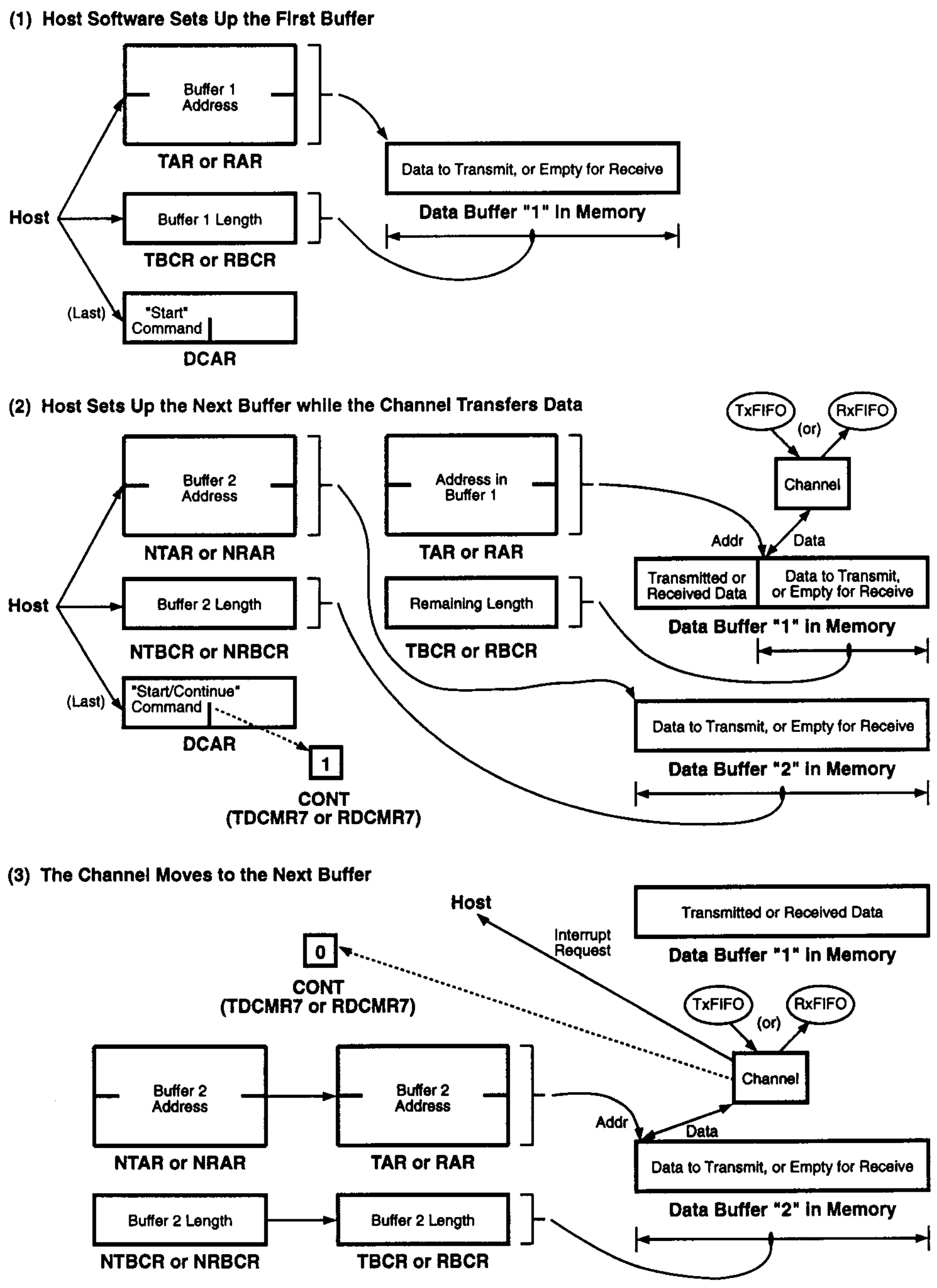

(1) Host Software Sets Up the First Buffer
Buffer 1 Address
TAR or RAR
Data to Transmit, or Empty for Receive
Data Buffer "1" In Memory
Host
Buffer 1 Length
TBCR or RBCR
(Last)
"Start" Command
DCAR
(2) Host Sets Up the Next Buffer while the Channel Transfers Data
TxFIFO
(or)
RxFIFO
Buffer 2 Address
NTAR or NRAR
Address in Buffer 1
TAR or RAR
Channel
Addr
Data
Transmitted or Received Data
Data to Transmit, or Empty for Receive
Data Buffer "1" In Memory
Host
Buffer 2 Length
NTBCR or NRBCR
Remaining Length
TBCR or RBCR
(Last)
"Start/Continue" Command
DCAR
Data to Transmit, or Empty for Receive
Data Buffer "2" In Memory
1
CONT (TDCMR7 or RDCMR7)
(3) The Channel Moves to the Next Buffer
Host
Interrupt Request
Transmitted or Received Data
Data Buffer "1" In Memory
0
CONT (TDCMR7 or RDCMR7)
TxFIFO
(or)
RxFIFO
Channel
Buffer 2 Address
NTAR or NRAR
Buffer 2 Address
TAR or RAR
Addr
Data
Data to Transmit, or Empty for Receive
Data Buffer "2" In Memory
Buffer 2 Length
NTBCR or NRBCR
Buffer 2 Length
TBCR or RBCR

Figure 6-3. Pipelined Mode DMA Operation

6.4.1 Avoiding Problems with the CONT Flag

Software must take care not to write a "Start/Continue" command to an operating channel while the channel is testing the CONT bit after completing a buffer. This is because, if the command occurs just after the channel has tested CONT as 0 and therefore cleared BUSY, the command restarts the channel to reuse the buffer described by TAR and TBCR, or RAR and RBCR, a second time.

The performance and interrupt-response characteristics of the processor and total system, considered in the context of the line protocol, may guarantee that software will always write the Start/Continue command for a buffer before the channel finishes with the previous one. But if this is not so, software should approach "notifying" the channel of a new buffer as shown in Figure 6-4. First, clear the Master Bus Request Enable bit (MBRE; DCAR8) and then test the BUSY bit (xDMR5). If BUSY is 1, write the register address and length to NxAR and NxBCR and then issue the "Start/Continue This Channel" command. If it is 0, write the address and length to xAR and xBCR and then issue the "Start This Channel" command. Be sure to set MBRE when writing either command, so that the channel(s) can operate again.

One drawback of Pipelined mode (as well as Array and Linked List modes), compared to Single-Buffer mode, is that host software can not read back the ending address and byte count to figure out the exact completion status of the buffer. For receiving, similar information can be obtained by using the Receive Status Block feature of the serial controller.

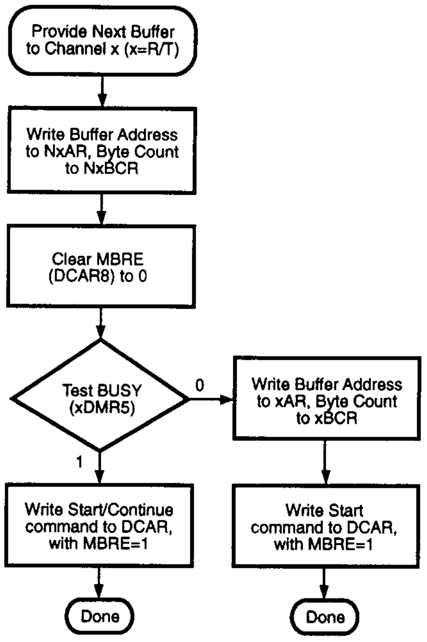

Provide Next Buffer to Channel x (x=R/T)
Write Buffer Address to NxAR, Byte Count to NxBCR
Clear MBRE (DCAR8) to 0
Test BUSY (xDMR5)
0
1
Write Buffer Address to xAR, Byte Count to xBCR
Write Start/Continue command to DCAR, with MBRE=1
Write Start command to DCAR, with MBRE=1
Done
Done

Figure 6-4. Posting a New Buffer (Pipelined Mode)

6.5 ARRAY MODE

In Array mode, host processor software sets up an arbitrarily long array or table of buffer addresses and byte counts in memory. Then it sets the DMA channel into operation to send all the data in all the buffers, or to receive data into all of them in turn.

The Array and Linked List modes differ from Pipelined mode in that software does not write the address of a data buffer into the Next Transmit Address Register (NTAR) or Next Receive Address Register (NRAR). Instead, in Array mode, it writes NTAR or NRAR with the address of the start of an array in memory, that contains the addresses and lengths of each of a whole set of data buffers.

Figure 6-5 illustrates Array mode operation. Each entry in the array may be six or 12 bytes long, as described in the later sections *Fetching Transmit Status Blocks* and *Storing Receive Status Blocks*; the Figure shows 6-byte entries to keep it as simple as possible. With either entry format, the first four bytes of each entry are the 32-bit buffer address and the next two bytes are the 16-bit byte count for the buffer.

Software can program the order in which the channel fetches the two halves of the address to match the characteristics of the host processor, as described later in *Format of Binary Values in Arrays and Lists.* If 16-bit (BCR2) is 0 and/or the 8/16 bit (TDMR8 or RDMR8) is 1, this parameter defines the order in which the channel fetches the four bytes of the address and the two bytes of the count.

After programming a Channel Mode Register for Array mode, software should start the channel by programming NTAR or NRAR to point to the array at the address of the first buffer to be used, and then writing a "Start/Init This Channel" command to the DCAR. This command differs from a "Start This Channel" command in that it set the INITG bit in the channel's DMA Mode Register (TDMR4 or RDMR4) as well as the BUSY bit (TDMR5 or RDMR5), so that the channel fetches the first array entry before starting DMA data transfer.

In array mode, the DMA channels treat two values of the byte count in each array entry as having special significance: zero indicates the end of the array, while one indicates a special "link entry" which does not describe a buffer, but rather provides a "link address" to another part of the array. This link address directly follows the byte count if Transmit Control Blocks or Receive Status Blocks are not included in array entries, else it follows the "unused" word that follows the TCB or RSB. In other words, in an array entry that includes a byte count of 0001, the link address is located in the same place as it is in a Linked List entry, and the IUSC ignores all other information in the entry.

This "array chaining" facility provides many of the benefits of the Linked List mode described in the next section, without the overhead of having to fetch a link address at the end of each entry. On the other hand, the overhead of fetching an entire entry just to get a link address may be worse than that of fetching a link address from each entry, in terms of worst-case timing.

For an array entry in which the byte count is 2 or more, if the **ClearCount** bit in the channel's DMA mode register (TDMR12 or RDMR12) is 1, the channel clears the byte count field of the entry, by writing zero to it. (Software can use this feature to track the DMA channel's progress through the array, but the main purpose of the feature is in Linked List mode.)

On the Transmit side, if the channel's TCBinA/L bit (TDMR13) is 1, the channel next reads the last six bytes of the entry. For the first entry in an array, and if a subsequent entry in the array aligns with the start of a frame, the IUSC interprets the first four of these six bytes as a Transmit Control Block. It always discards/ignores the last two bytes.

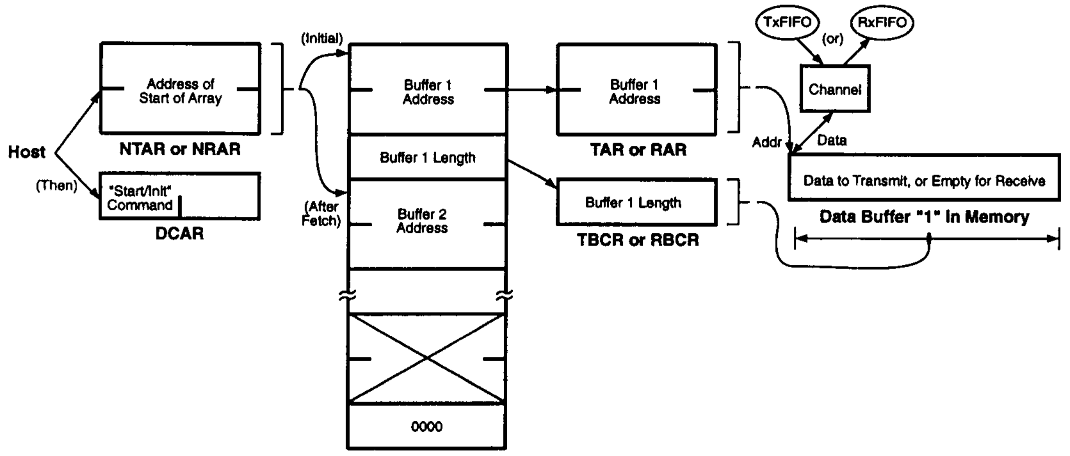
IUSCs with a datecode of 9239 or earlier did not fetch a TCB from the first entry of an array. See *Fetching Transmit Control Blocks* later in this chapter for further details.

After fetching an array entry, if and when the internal Request signal from the Transmitter or Receiver is true, the DMA channel begins transferring data to or from the first buffer in memory, as described earlier in *DMA Fundamentals.* (On the Transmit side, if TCBinA/L is 0 but Transmit Control Blocks are enabled, the Transmitter will interpret the first four bytes from the buffer as a TCB.)

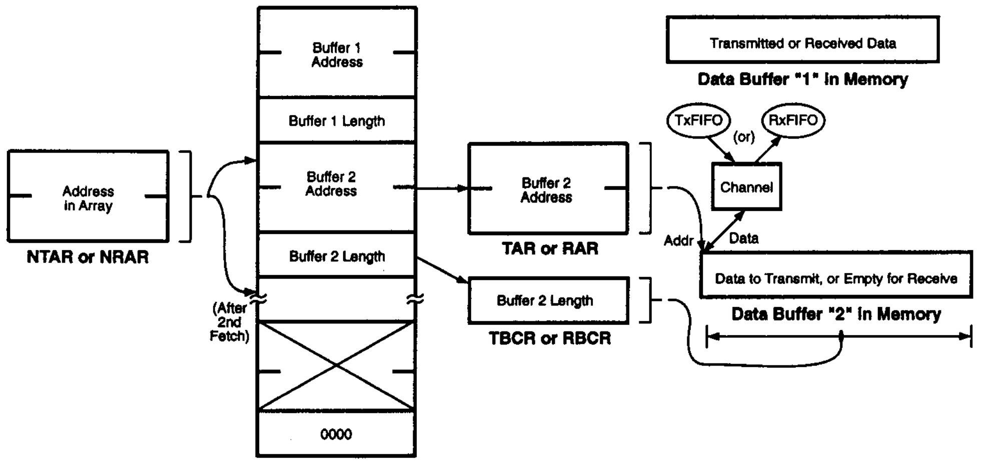
As in other modes, a DMA channel typically finishes a buffer when it has decremented the buffer's byte count to zero, and/or if the TermE bit in the channel's DMA Mode Register (TDMR9 or RDMR9) is 1 to enable the Early Buffer Termination feature and the serial controller signals that the current transfer includes the last character of a frame or message.

6.5 ARRAY MODE (Continued)

(1) Host Software Sets Up the Array and Starts the Channel, which Fetches the First Entry



(2) The Channel Moves to Buffer 2



(3) The Channel Reaches the End of the Array

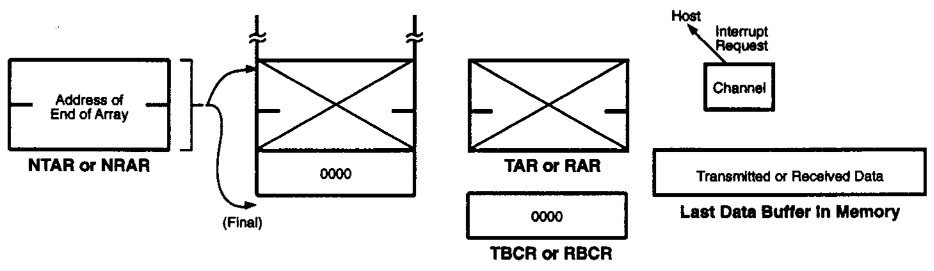


Figure 6-5. Array Mode DMA Operation

On the Receive side, if Receive Status Blocks are enabled as described in Chapter 5, after the Receive DMA channel stores a character marked with RxBound status, the Receiver maintains its request to the DMA channel until the latter has read out the 2- or 4-byte RSB. (The Receiver does this whether or not Early Termination is enabled.) If the RSBinA/L bit in the Receive DMA Mode Register (RDMR13) is 0, the channel writes the RSB into the data buffer after the last character of the frame. If RSBinA/L is 1, the channel writes the RSB into the array entry after the byte count, and then writes zero to the next two or four bytes.

The DMA channel then tries to advance to the next buffer in the array, reading the next address and byte count as it did for the first buffer. When the channel fetches a zero byte count from an array entry, it goes to an inactive state, in which case the software must reprogram the channel and restart it before it can perform further DMA transfers.

In Array mode a channel uses NTBCR or NRBCR only as a temporary holding register, so software does not have to set up this register. In particular, NxBCR does NOT specify the length of the array—rather, a zero in the byte count field of an entry signals the end of the array.

Software can program the IUSC to interrupt when the DMA channel and/or the serial controller completes each data buffer, and/or when the DMA channel reaches the end of the array. Host software can track the channel's progress through the array by reading back the address in the NTAR or NRAR.

In Array mode a DMA channel becomes more autonomous and independent of processor response than in Pipelined mode. In general, this mode is less dependent on processor action than is Pipelined mode. This is particularly important when several short frames arrive and must be placed into consecutive buffers

But in one way Array mode is more dependent on host processor response than is Pipelined mode. When the DMA channel comes to the zero buffer length that signals the end of the array, it becomes inactive and waits for host processor software action, just as in Single Buffer mode. Presumably on the transmit side, each array can be made to end at the end of a message or frame, so that this characteristic should not cause any problems. But, on the receive side, the serial controller may be subject to FIFO overruns and lost data if the host processor software does not reprogram the DMA channel with a new array in a timely manner.

6.6 LINKED LIST MODE

This mode is similar to Array mode, particularly in its capability to switch buffers rapidly for each of multiple successive short frames, but it adds a capability for dynamic updating as in Pipelined mode.

In Linked List mode the DMA channel fetches a buffer address and a byte count from the first six bytes of a list entry for each buffer, just as in Array mode, but in Linked List mode these entries do not have to follow one another in memory. The difference between array entries and list entries is that each list entry includes the 32-bit address of the next entry. As in array mode, a zero in the byte count field of an entry signals the end of the list, and the other fields in such a final entry do not matter.

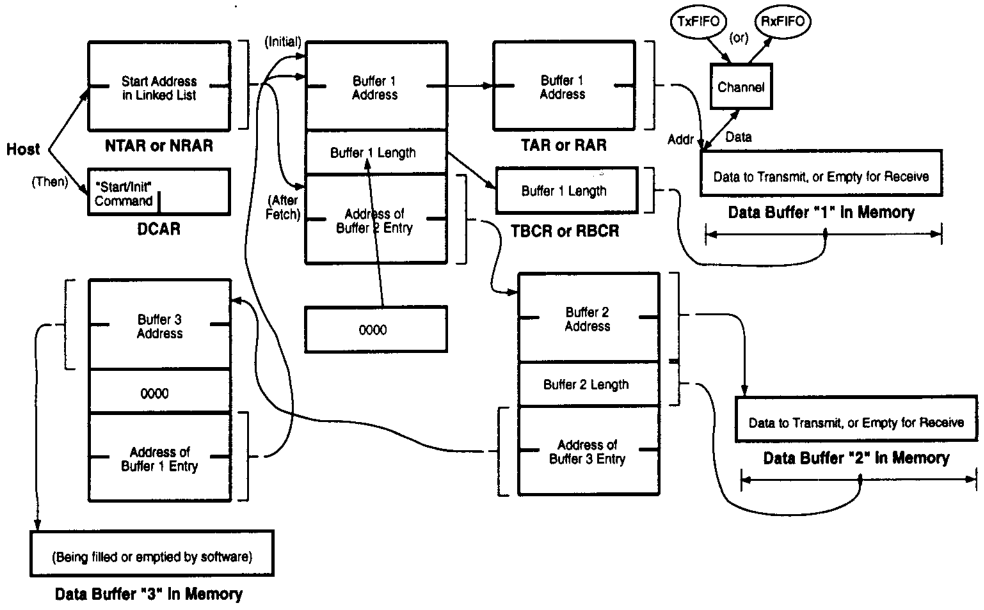
List entries can be ten bytes long or 16 bytes long, depending on whether they include a Transmit Control Block or Receive Status Block, as described in the later sections *Fetching TCBs* and *Storing RSBs*. (Figure 6-6 shows 10-byte entries to keep it as simple as possible.) With either entry format, the first four bytes of each entry are the 32-bit buffer address and the next two bytes are the 16-bit byte count for the buffer.

As in Array mode, software can control the order in which the channel fetches the two halves of each address. When a channel is restricted to byte transfers, this option controls the order in which it fetches the four bytes of the address and the two bytes of the byte count.

After programming a Channel Mode Register for Linked List mode, host software typically starts the channel by programming NTAR or NRAR to point to the linked list at the address of the first buffer to be used, and then writing a "Start/Init This Channel" command to the DCAR. This command differs from "Start This Channel" in that it set the INITG bit in the channel's DMA Mode Register (TDMR4 or RDMR4) as well as the BUSY bit (TDMR5 or RDMR5), which makes the DMA channel fetch the first list entry before beginning DMA data transfer.

After the channel fetches a buffer's address and byte count, and verifies that the byte count is non-zero, if the ClearCount bit in the channel's DMA Mode Register (TDMR12 or RDMR12) is 1, the channel clears the byte count field of the entry, by writing zero to it. This feature is especially valuable when software arranges the linked list in a "ring" structure, as described later.

(1) Host Software Sets Up the Linked List and Starts the Channel, which Fetches the First Entry and then Clears the Byte Count.


TxFIFO
(or)
RxFIFO
Channel
Start Address in Linked List
NTAR or NRAR
Host
(Then)
"Start/Init" Command
DCAR
(Initial)
(After Fetch)
Buffer 1 Address
Buffer 1 Length
Address of Buffer 2 Entry
0000
Buffer 1 Address
TAR or RAR
Buffer 1 Length
TBCR or RBCR
Addr
Data
Data to Transmit, or Empty for Receive
Data Buffer "1" in Memory
Buffer 3 Address
0000
Address of Buffer 1 Entry
Buffer 2 Address
Buffer 2 Length
Address of Buffer 3 Entry
Data to Transmit, or Empty for Receive
Data Buffer "2" in Memory
(Being filled or emptied by software)
Data Buffer "3" in Memory

(2) Software Finishes Filling or Emptying Buffer #3 and Sets Its Length

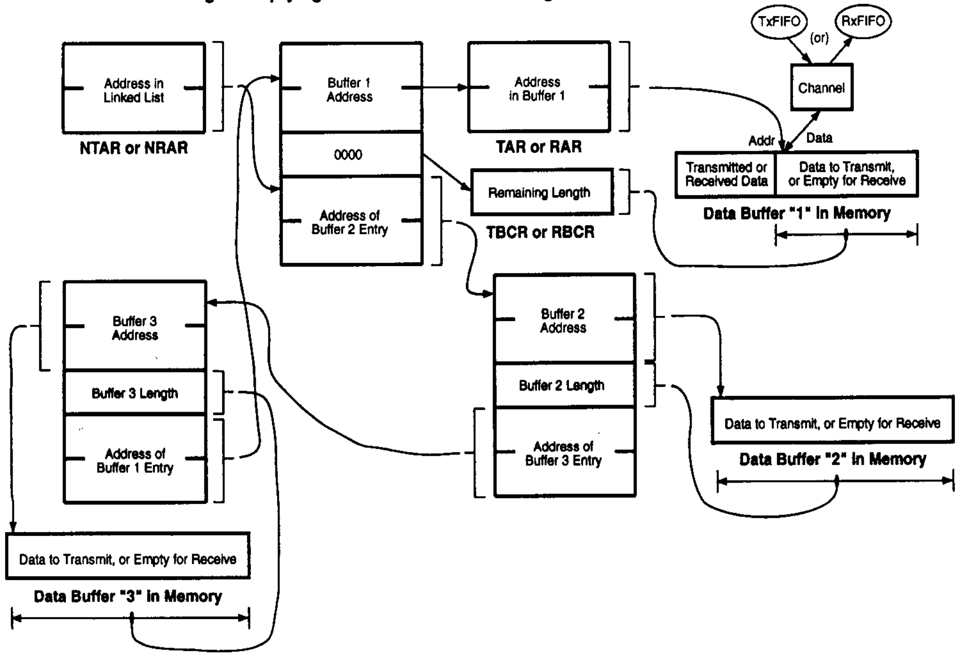

TxFIFO
(or)
RxFIFO
Channel
Address in Linked List
NTAR or NRAR
Buffer 1 Address
0000
Address of Buffer 2 Entry
Address in Buffer 1
TAR or RAR
Remaining Length
TBCR or RBCR
Addr
Data
Transmitted or Received Data
Data to Transmit, or Empty for Receive
Data Buffer "1" in Memory
Buffer 3 Address
Buffer 3 Length
Address of Buffer 1 Entry
Buffer 2 Address
Buffer 2 Length
Address of Buffer 3 Entry
Data to Transmit, or Empty for Receive
Data Buffer "2" in Memory
Data to Transmit, or Empty for Receive
Data Buffer "3" in Memory

Figure 6-6a. Linked List DMA Mode with a Three-Buffer Ring (1 of 2)

6.6 LINKED LIST MODE (Continued)

(3) The Channel Moves to Buffer 2, and Requests a Host Interrupt

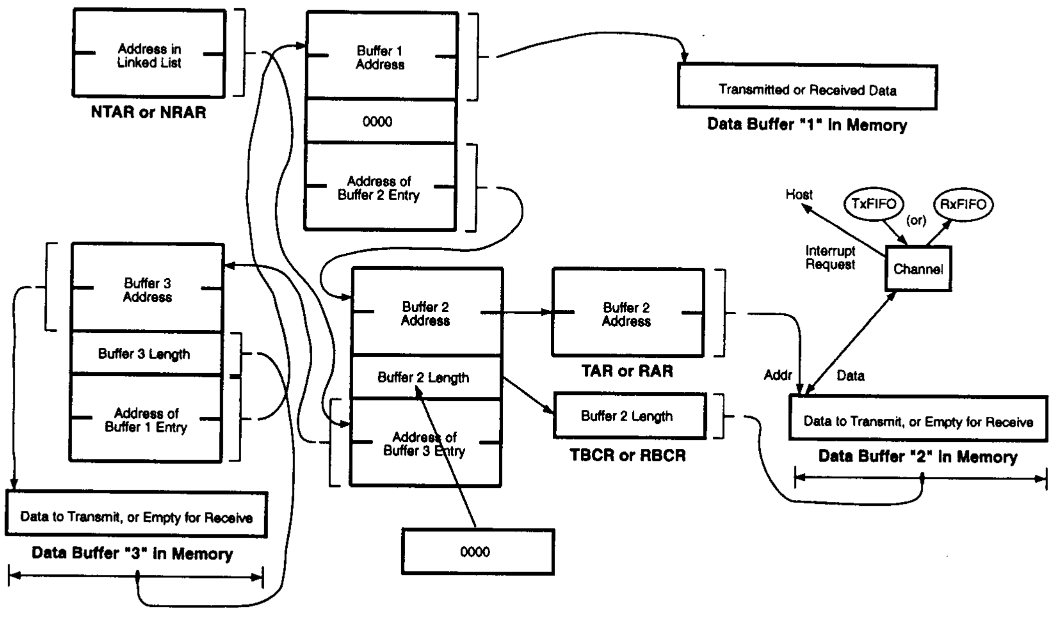

Address in Linked List
NTAR or NRAR
Buffer 1 Address
0000
Address of Buffer 2 Entry
Transmitted or Received Data
Data Buffer "1" In Memory
Host
TxFIFO
(or)
RxFIFO
Interrupt Request
Channel
Buffer 3 Address
Buffer 3 Length
Address of Buffer 1 Entry
Buffer 2 Address
Buffer 2 Length
Address of Buffer 3 Entry
Buffer 2 Address
TAR or RAR
Buffer 2 Length
TBCR or RBCR
Addr
Data
Data to Transmit, or Empty for Receive
Data Buffer "2" In Memory
Data to Transmit, or Empty for Receive
Data Buffer "3" In Memory
0000

Figure 6-6b. Linked List DMA Mode with a Fixed Three-Buffer Ring (2 of 2)

On the Transmit side, if the channel's TCBinA/L bit (TDMR13) is 1, the channel then reads the next six bytes of the entry. For the first entry of a list, or if a subsequent entry aligns with the start of a frame, the IUSC interprets the first four bytes as a Transmit Control Block. It always ignores/ discards the last two bytes.

After fetching this much of a list entry, when the internal Request signal from the Transmitter or Receiver is true, the DMA channel transfers data to or from the first buffer in memory, as described earlier in *DMA Fundamentals*. (On the Transmit side, if TCBinA/L is 0 but Transmit Control Blocks are enabled, the Transmitter will interpret the first four bytes from the buffer as a TCB.)

As in other modes, a DMA channel typically finishes a buffer when it has decremented a buffer's byte count to zero, and/or if the TermE bit in the channel's DMA Mode Register (TDMR9 or RDMR9) is 1 to enable the Early Buffer Termination feature, and the serial controller signals that the current transfer includes the last character of a frame or message.

On the Receive side, if Receive Status Blocks are enabled as described in Chapter 5, after the Receive DMA channel stores a character marked with RxBound status, the Receiver maintains its DMA request until the channel has read out the 16- or 32-bit RSB. (It does this whether or not Early Termination is enabled.) If the RSBinA/L bit in the Receive DMA Mode Register (RDMR13) is 0, the channel writes the RSB into the data buffer after the last character of the frame. If RSBinA/L is 1, the channel writes the RSB into the list entry after the byte count, and then writes zeroes to the next two or four bytes.

The DMA channel then tries to advance to the next buffer in the list, by first fetching the address of the next entry (this address follows the byte count if the TCBinA/L or RSBinA/L bit is 0, otherwise it follows the last unused byte). Then the DMA channel fetches the buffer address and byte count from the next entry.

If the next byte count is non-zero, the channel continues to transfer data to or from the new buffer. If the byte count is zero, the channel goes to an inactive state, in which case software must reprogram and restart the channel before it can transfer any more data.

Software can program the IUSC to interrupt the processor when the DMA channel and/or serial controller completes each data buffer, and/or when the DMA channel reaches the end of the linked list. Host software can track the channel's progress through the list by reading back the address in the NTAR or NRAR.

6.6.1 Using Linked List Mode to Create a Buffer Ring

Figure 6-6 illustrates operation in Linked List mode. In the application shown, DMA transfers and software processing of the data rotate among a fixed set of three buffer areas in memory. The next-entry addresses in their list entries configure the list as a "circular ring". This is the kind of application for which the ClearCount bits are provided on the Z16C32.

In the first part of the Figure, software starts the DMA channel, to transfer data into or out of buffer "1". The host processor (or another hardware element) is putting new transmit data into buffer "3", or is taking received data out of "3". While it is doing so, the byte count field for buffer 3 remains zero.

As described earlier, when a channel's ClearCount bit (TDMR12 or RDMR12) is 1, the channel writes zero into the byte count field of each buffer's list entry, after it has read the count and found it to be non-zero. This zero byte count prevents the DMA channel from circling around the ring and reusing the buffer again, before the software has filled or emptied the buffer and then "refreshed" the byte count.

In the second part of the Figure, software (or whatever) finishes filling or emptying buffer "3", and software places a non-zero byte count in its list entry. It needs to do this with care to avoid problems if the DMA channel accesses the end of the list at (more or less) the same time. The following procedure is recommended:

1a. On a 16-bit bus, store the byte count using a 16-bit write operation, OR

1b. on an 8-bit bus, write the Master Bus Request Enable Bit (MBRE, DCAR8) to 0, then write the two halves of the byte count, then write MBRE back to 1.

2. In systems that use the MaxXfers and/or MaxCLKs fields of the Burst/Dwell Control Register (BDCR) to "throttle" the IUSC's DMA activity (as described in a later section), add a few "No-ops" or a short timing loop at this point. The delay should cover the case when the IUSC fetches a zero byte count, but then gives up the bus because of the BDCR throttling, before it fetches the rest of the terminating entry and clears the BUSY bit.

3. Read the TDMR or RDMR and test the DMA channel's BUSY bit. If it is 0, the channel fetched the byte count as zero before we stored the new value, and must be restarted—go to a routine that does this. If BUSY is still 1, the newly filled or emptied buffer has been successfully added to the list and will be handled by the DMA channel.

In the third part of the Figure, the channel finishes sending data from buffer "1" or receiving data into it. It requests an interrupt on the host processor, and goes on to buffer "2", clearing its byte count. The interrupt triggers the software to empty or fill buffer "1" and then set its new byte count.

6.6.2 Adding a Buffer to the End of a List

On other systems, buffers are not arranged in a ring, but are passed from one software routine to another as they are filled and emptied. In such systems, software may set the ClearCount bit for progress-tracking reasons, but does not need to do so. In this case, the procedure that software needs to perform carefully is that of adding a buffer to the end of a linked list for an operating DMA channel. It should do so as follows:

1. Create a list entry for the new buffer—often one exists and simply needs its buffer address and/or byte count "refreshed". Place the address and count in the entry, along with the TCB for a transmit buffer if this feature is used.

2. Place the address of an "end of list" entry (one that includes a zero byte count) in the next entry address field of the new list entry.

3. Locate the list entry for the previous last buffer in the list. (This entry will also have its "next entry address" pointing to an "end-of-list" entry.)

4a. If the processor and system bus are both 32 bits wide, or if it can be otherwise ensured that the software can write a 32-bit address into memory without interference from Z16C32 activity, software can simply write the address of the new entry into the next entry address field of the entry for the previously last buffer.

4b. Otherwise, software should write the Master Bus Request Enable bit (MBRE; DCAR8) to 0, then write the address of the new entry into the next entry address field of the entry for the previously last buffer, and then set MBRE back to 1 again.

5. In systems that use the MaxXfers or MaxCLKs fields of the Burst/Dwell Control Register (BDCR) to "throttle" the DMA activity of the Z16C32 (as described in a later section), it might be a good practice to include a few "No-ops" at this point. There should be enough NOPs to eliminate the case in which the DMA channel fetches the link address to the "end of list" entry from the previously-last entry, before we store the new one in step 4, but then releases the bus for a while because of this throttling, before it fetches the zero byte count.

6. Read the TDMR or RDMR and test the DMA channel's BUSY bit. If it is 0, the channel got to the end-of-list before our new link address could prevent this, and the channel must be restarted—go to a routine that does this. If BUSY is still 1, the new buffer has been successfully added to the list and will be handled by the DMA channel.

6.7 FETCHING TRANSMIT CONTROL BLOCKS

In Array and Linked-List modes, if software enables the Transmit Control Block feature of the serial controller (see *DMA Support Features: Transmit Control Blocks* in Chapter 5 for more information about this feature), the Transmit DMA channel can fetch the TCBs in two ways.

The **TCBinA/L** bit in the Transmit DMA Mode Register (TDMR13) controls whether the channel fetches TCBs from Array and Linked List entries. This bit also controls the length of the entries. If TCBinA/L is 0, Array entries are six bytes long, Linked List entries are ten bytes long, and the channel handles TCBs the same way that it does in Single Buffer or Pipelined mode. That is, it fetches a 32-bit TCB from the data buffer just before it fetches the first character of each frame. In this case, the length of the TCB is included in the Byte Count of the buffer (but not in the length of the frame for the TCC).

If TCBinA/L is 1, Array entries are 12 bytes long and Linked List entries are 16 bytes long. The channel fetches TCBs from the first array or list entry, and from the subsequent entries, if the start of their buffers aligns with the start of a frame. For such entries, the channel fetches the four bytes of the TCB after it has read (and if the ClearCount bit is 1 written zero back to) the byte count in the array or list entry, and then reads and discards the next two bytes. For a subsequent entry that does not align with the start of a frame, the channel simply reads and discards the six bytes that follow the byte count.

IUSCs with a datecode of 9239 or earlier did not fetch a TCB from the first entry of an array or linked list. For applications that might run on such early devices, software can tell whether a given device fetches the first TCB as described in *Determining the Device Revision Level* in Chapter 8.

Figures 6-7a and 6-7b show two examples of TCBs with TCBinA/L=1.

The length of a TCB in an Array/List entry is not included in the DMA channel's byte counts nor in the frame length values for the TCC.

With either kind of TCB placement, the DMA and serial controllers operate fairly independently, without a lot of context-signaling between them, and it is important that software do what is needed to keep them co-ordinated and synchronized. These measures include:

1. With TCBinA/L=0, allow six bytes for each array entry or ten bytes for each list entry, place the 32-bit TCB before the start of each frame, and include the length of TCBs in the byte counts of the buffers in which they are included.

2. With TCBinA/L=1, allow 12 bytes for each array entry or 16 bytes for each list entry, and place the TCB in the 7th through 10th bytes of each entry that starts a frame. (The Z16C32 ignores these locations in entries for subsequent buffers within an ongoing frame.)

3. With TCBinA/L=1, either ensure that a frame never starts in the middle of a buffer, or else place a TCB in the buffer before the start of each frame that does (as when TCBinA/L=0). In a Little-Endian system or with an 8-bit data bus, it is acceptable to use the Early Buffer Termination feature (described later) as a simple way to ensure that a frame never starts in the middle of a buffer.

But for a 16-bit or wider bus in a Big-Endian system, the DMA channel is only guaranteed to access the final byte of a frame correctly if software programs the Byte Count of the last buffer of the frame correctly, to match the TCC frame length. (In this case, there is no reason to enable Early Termination.)

4. With either kind of TCB placement, write a "Load TCC" command to the CCAR before restarting the Transmit DMA channel. This is necessary because when the Transmit DMA channel fetches the terminating entry of an array or linked list, it presents the TCB information from the entry to the Transmitter before it checks whether the Byte Count is zero. This conditions the Transmitter to interpret the next transfers done by the channel as serial data. The "Load TCC" command makes the Transmitter interpret the next transfer(s) done by the DMA channel as a TCB.

6.7 FETCHING TRANSMIT CONTROL BLOCKS (Continued)

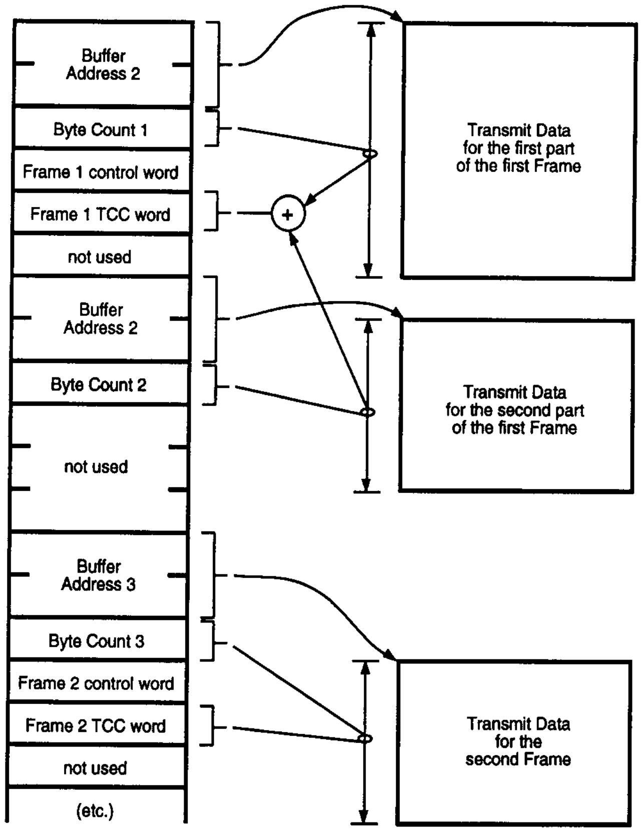


Figure 6-7a. Array Mode Transmit Control Blocks with TCBinA/L=1

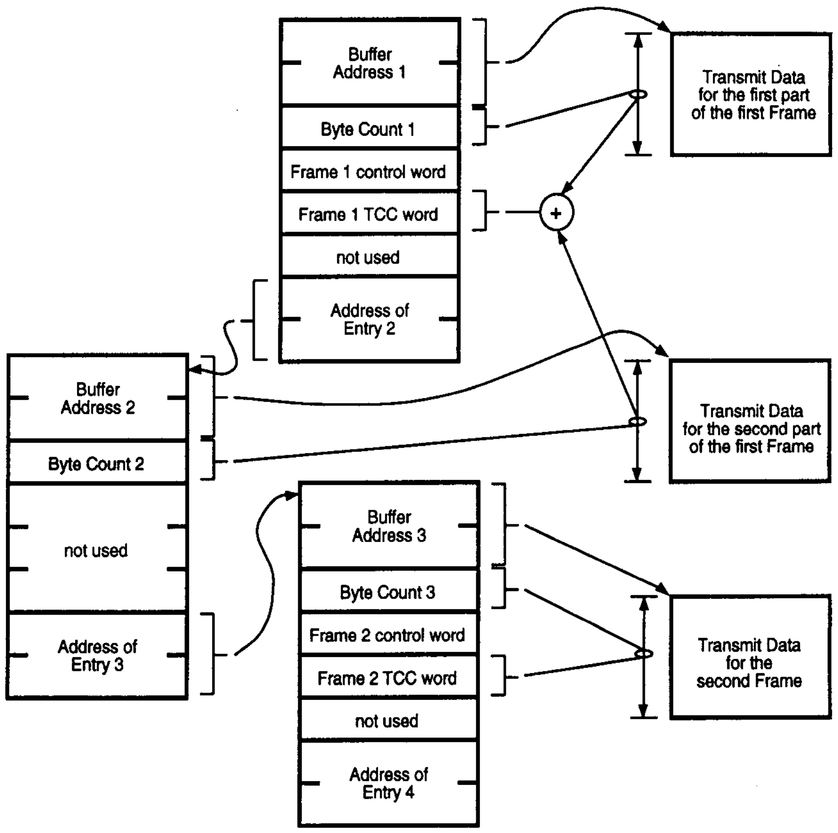


Figure 6-7b. Linked List Transmit Control Blocks with TCBinA/L=1

6.8 STORING RECEIVE STATUS BLOCKS

Similarly, if software enables the Receive Status Block feature as described in Chapter 5, in Array or Linked List mode a Receive DMA channel can store RSBs in two ways as shown in Figure 6-8.

If the **RSBinA/L** bit in the Receive DMA Mode Register (RDMR13) is 0, the channel handles RSBs as it does in Single Buffer and Pipelined modes. Array entries are 6 bytes long, Linked List entries are 10 bytes long, and the channel stores the RSB after the last byte of each frame. In these cases, software should allow for the length of RSBs in the byte counts of the buffers in which they are stored. (RCC residual values never reflect RSB bytes.)

But when RSB's are stored in the data buffers, software has to read the RCC FIFO to determine the length of each frame received, so that it can find the RSB's. (Because of this, there's no reason to ever use a 32-bit RSB in this mode.) Since the RCC FIFO is only four deep, software must read it in a reasonably timely manner. In Array and Linked List modes, this software response requirement can be eased by programming 32-bit RSBs and the RSBinA/L bit 1.

When RSBinA/L is 1, Array entries are 12 bytes long, Linked List entries are 16 bytes long, and the DMA channel stores an RSB in the (7th-8th or) 7th-10th bytes of the last array or list entry for each frame, after it has placed the last character of the frame in the buffer. When RSBinA/L is 1, software can ignore the RCC FIFO, and need not respond to IUSC interrupts as promptly. After the Rx DMA channel has stored the RSB in the array or list entry, it writes 2 (or 4) zero bytes to skip over that many "extra" bytes in the entry. These extra bytes maintain 32-bit boundary alignment of the addresses in the array or list entries, as required by some processors.

Software MUST NOT read the RCC FIFO when using 32-bit RSBs, because the hardware takes the RCC residual value in the RSB from the RCC FIFO.

When RSBinA/L is 1, the length of the RSB is not included in either the DMA channel's byte counts nor in the RCC residual values.

As on the Transmit side, software has to take certain steps to ensure that the Receiver and the DMA channel work together:

1. With RSBinA/L=0, allow six bytes for each array entry or ten bytes for each list entry, and allow for Receive Status Blocks in the byte counts of buffers in which they are stored.

2. With RSBinA/L=0, read the RCC FIFO once for each frame and use these RCC residual values as described in Chapter 5, to determine the length of each frame. Knowing the frame lengths, software can then find the RSBs, which follow the last character of each frame.

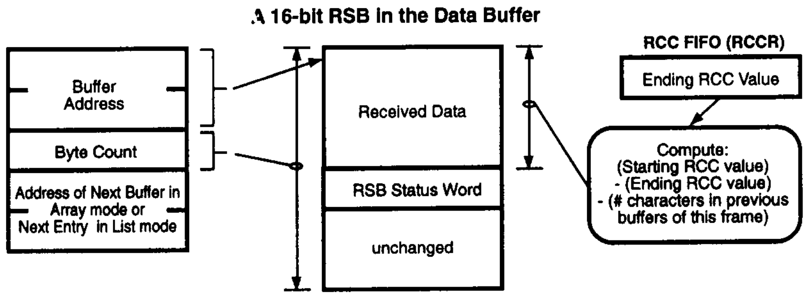
3. With RSBinA/L=1, always program the TermE bit in the Receive DMA Mode Register (RDMR9) to 1 to enable the Early Buffer Termination feature.

4. With RSBinA/L=1, allow 12 bytes for each array entry or 16 bytes for each list entry. The length of RSB's need not be included in buffer byte counts for the DMA channel, nor in a maximum frame length value for the RCC.

With RSBinA/L=1, the channel stores an RSB in the 7th-10th (or 7th-8th) bytes of the array or list entry for each buffer in which it stores a character marked with RxBound status. It zeroes these locations in array or list entries for preceding buffers within the same frame, that is, those that it fills before the frame ends.

Zilog guarantees that all versions of the IUSC will always store bit 13 of the first word of an RSB as zero. Thus, with RSBinA/L=1, software can initialize such status word locations to hex 2000, and can test bit 13 to see when an IUSC has finished with a buffer. When it encounters an entry with bit 13 cleared, software should next check the RxBound bit (bit 3); those in which this bit is 1 represent buffers that include the end of a frame.

On the Receive side, there are actually two internal termination signals that the serial controller asserts to the Receive DMA channel. [The DMA channel honors these signals only when its TermE bit (RDMR9) is 1.] The serial controller asserts one of these signals as the DMA channel takes a byte marked with RxBound status out of the RxFIFO. If software hasn't enabled the Receive Status Block (RSB) feature in the Channel Control Register (CCR7-6), the serial controller asserts the other signal at the same time, otherwise it asserts the other termination signal when the DMA channel stores the last of the two or four bytes of the RSB. If the DMA channel is in Array or Linked List mode and has been programmed to store RSB's in the array or list, it uses the first signal to shift from storing in the data buffer to storing in the array or list, and uses the second signal to shift from storing in the array or list to fetching information for the next buffer. In all other modes, the channel simply uses the second signal to know when it has stored all the information for the current buffer.


A 16-bit RSB in the Data Buffer
Buffer Address
Byte Count
Address of Next Buffer in Array mode or Next Entry in List mode
Received Data
RSB Status Word
unchanged
RCC FIFO (RCCR)
Ending RCC Value
Compute: (Starting RCC value) - (Ending RCC value) - (# characters in previous buffers of this frame)

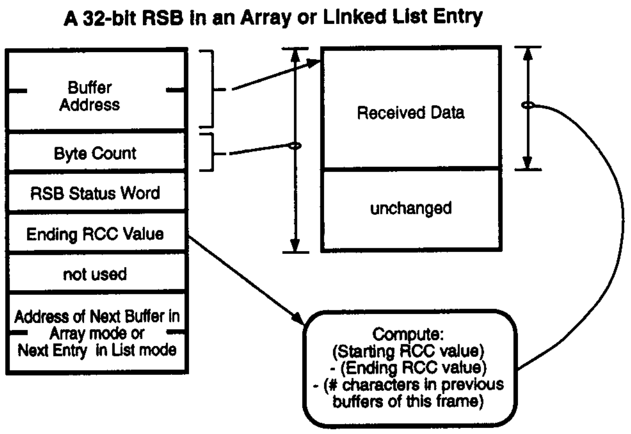

A 32-bit RSB in an Array or Linked List Entry
Buffer Address
Byte Count
RSB Status Word
Ending RCC Value
not used
Address of Next Buffer in Array mode or Next Entry in List mode
Received Data
unchanged
Compute: (Starting RCC value) - (Ending RCC value) - (# characters in previous buffers of this frame)

Figure 6-8. Receive Status Blocks

6.9 CHANNEL STATUS

The earlier Figure 6-1 shows the less significant byte of each DMA Mode Register (TDMR or RDMR), which contains eight bits indicating the status of that channel. A channel clears these bits to 0 in response to a hardware Reset or when software writes a Reset command to the DMA Command/Address Register (DCAR). All of them can be set and/or cleared by the DMA channel. Some of them are affected by commands other than Reset, and/or when software reads the (LS byte of the) register.

Some of these bits exist solely for the information of host software. The DMA channel uses many of them as part of its internal state.

CONT (TDMR7 or RDMR7): this bit is used only in Pipelined mode. In this mode the IUSC sets it to 1 when software writes a "Start/Continue This Channel" command to the MS byte of the DCAR. A channel checks CONT when it has decremented its Byte Count Register (TBCR or RBCR) to zero, or, if software has enabled early buffer termination, if/when the serial controller signals for such a termination. If CONT is 0 at this time, the channel clears the BUSY bit (xDMR5) and stops. Otherwise, the channel clears CONT to 0 and continues operating. It transfers the contents of its Next Address Register to its Address register (NTAR to TAR, or NRAR to RAR) and transfers the contents of its Next Byte Count Register to its Byte Count Register (NTBCR to TBCR, or NRBCR to RBCR). Then it resumes transferring serial data to or from the new buffer, as requested by the serial controller.

Software may need to take special precautions to avoid issuing the "Start/Continue" command while the channel is testing the CONT bit, as described in the earlier section, *Pipelined Mode*.

GLink (TDMR6 or RDMR6): this bit can only be 1 in Linked List mode, while the channel is reading the address of the next list entry from memory. GLink stays set if the channel clears BUSY (xDMR5) while reading the link address, because of a command or a hardware Abort. In this case software must clear GLink by issuing a "Reset This Channel" command before it restarts the channel.

BUSY (TDMR5 or RDMR5): this bit is set to 1 by any of the Start commands, and remains 1 while the channel is still operating in response to the command. It is 0 if the channel has stopped and will need software attention (including another Start command) before it can resume operation. The channel sets BUSY when host software writes a Start, Start/Init, or Start/Continue command (for one or all channels) to the DCAR. The channel clears BUSY when one of the following occurs:

1. a hardware Reset,
2. a Reset, Pause, or Abort command, for this channel or all channels,
3. if external hardware asserts the /ABORT pin low during a transfer by the channel,
4. reading a zero byte count in Array or Linked List mode,
5. decrementing the byte count (TBCR or RBCR) to zero in Single Buffer mode,
6. if software has enabled early buffer termination in Single Buffer mode, and the serial channel signals for such a termination, or
7. if the channel tests the CONT bit as zero in Pipelined mode, after it has decremented the Byte Count Register to zero, or, if software has enabled early buffer termination, after the serial channel has signaled for such a termination.

INITG (TDMR4 or RDMR4): this bit is used only in Array and Linked List modes. It indicates whether the channel is reading from the array or list. The channel sets INITG to 1 when software issues a Start/Init command, and/or when the channel decrements its byte count (TBCR or RBCR) to zero, and/or if software has enabled the early termination feature and the serial controller signals for buffer termination. The channel clears INITG to 0 after it has read the address and byte count of the next buffer from memory.

INITG stays set if a channel clears the BUSY bit while reading array or list information, due to a zero byte count or some other reason. In this case software should clear the bit using a "Reset This Channel" command, before restarting the channel using a Start This Channel command. (There's no need to clear INITG before a Start/Init.)

EOA/EOL *End Of Array/End Of List* (TDMR3 or RDMR3): a channel set this bit to 1 in Array or Linked List mode after it has read a zero byte count from the array or list in memory. A channel clears EOA/EOL to 0 in response to a hardware or software Reset, and when software reads the bit as 1.

EOB *End Of Buffer* (TDMR2 or RDMR2): a channel sets this bit to 1 in any mode when it decrements its Byte Count Register to zero. It also sets EOB if software has enabled the early termination feature and the serial controller signals for buffer termination. A channel clears EOB to 0 in response to a hardware or software Reset, and when software reads the bit as 1.

HAbort (TDMR1 or RDMR1): a channel clears BUSY and sets this bit to 1 in any mode, if external hardware asserts the /ABORT pin low during a bus cycle by the channel. A channel clears HAbort to 0 in response to a hardware or software Reset, and when software reads the bit as 1.

SAbort (TDMR0 or RDMR0): a channel clears BUSY and sets this bit to 1 in any mode, if host software writes an Abort command for this channel (or all channels) to the DCAR. A channel clears SAbort to 0 in response to a hardware or software Reset, and when software reads it as 1.

Chapter 7 describes how each of the EOA/EOL, EOB, HAbort, and SAbort bits has a corresponding Interrupt Arm (IA) bit in the channel's DMA Interrupt Arm Register (TDIAR or RDIAR). If a status bit's corresponding IA bit is 1, the IUSC can request an interrupt when the channel sets the status bit to 1.

Since the channel clears the EOA/EOL, EOB, HAbort, and SAbort bits each time software reads the LS byte of its Channel Mode Register, software should take care when reading this register so that important events are not inadvertently lost. Specifically, any time software reads the LS byte of TDMR or RDMR, it should check and handle any and all of these four conditions/bits that are possible and significant.

6.10 COMMANDS AND /BUSREQ ENABLE

The DMA Command/Address Register (DCAR), shown in Figure 6-9, is a "shareable register", meaning that there's only one DCAR and that its contents apply to both of the IUSC's DMA channels. Software can use the LS byte of the DCAR for indirect register addressing, as described in Chapter 2, and can write commands for the DMA channels to the MS byte. Such commands can be directed to a specific channel or to all channels. The MS byte also contains one bit that enables or disables all operation of the DMA channels, by allowing or blocking assertion of the IUSC's /BUSREQ output.

The MS byte of the DCAR can be viewed as including a four-bit **DCmd** field in DCAR15-12 and the **Rx/Tx Reg** bit in DCAR9. For commands that affect one channel, the latter bit selects whether the command is for the Transmit or Receive channel. For other commands the IUSC ignores the Channel Select bit. Since there are only two channels and only six channel-specific commands, it is probably simpler to regard DCAR15-9 as a 7-bit field encoded as follows:

DCAR15-9	Command
0000000	Null (no operation)
0001000	Reset Tx Channel
0001001	Reset Rx Channel
0010000	Start Tx Channel
0010001	Start Rx Channel
0011000	Start/Continue Tx Channel
0011001	Start/Continue Rx Channel
0100000	Pause Tx Channel
0100001	Pause Rx Channel
0101000	Abort Tx Channel
0101001	Abort Rx Channel
0111000	Start/Init Tx Channel
0111001	Start/Init Rx Channel
1000000	Reset Highest IUS
1001000	Reset All Channels
1010000	Start All Channels
1011000	Start/Continue All Channels
1100000	Pause All Channels
1101000	Abort All Channels
1111000	Start/Init All Channels

Other combinations of the DCAR15-9 bits are reserved by Zilog and should not be written to the DCAR.

The **Master Bus Request Enable** bit (**MBRE**/DCAR8) controls whether the DMA channels can assert the /BUSREQ output to request control of the external bus from the host processor or central arbiter. Carrying the integration of the MS byte value one step further, note that the IUSC always captures a new state for MBRE whenever software writes the MS byte of DCAR. Note also that there is a strong link between certain commands and a particular state of MBRE. The following table gives typical full-byte hexadecimal values that can be written to the MS byte of DCAR to accomplish various operations. For commands that deactivate one channel, the table includes two values. The first applies to full-duplex operation in which the two channels operate independently, while the second applies to half-duplex operation in which only one channel is active at a time.

DCAR15-8	Operation
00	Disable /BUSREQ (no other effect on the Channels)
01	Enable /BUSREQ (no other effect on the Channels)
11/10	Reset Tx Channel
13/12	Reset Rx Channel
21	Start Tx Channel
23	Start Rx Channel
31	Start/Continue Tx Channel
33	Start/Continue Rx Channel
41/40	Pause Tx Channel
43/42	Pause Rx Channel
51/50	Abort Tx Channel
53/52	Abort Rx Channel
71	Start/Init Tx Channel
73	Start/Init Rx Channel
81	Reset Highest DMA IUS (enable/BUSREQ)
90	Reset All Channels
A1	Start All Channels
B1	Start/Continue All Channels
C0	Pause All Channels
D0	Abort All Channels
F1	Start/Init All Channels

A **Reset** command to a channel clears all the status bits in the LS byte of its DMA Mode Register, including BUSY, thus disabling the channel. It also clears the register bits associated with interrupts from the channel, namely the IE, IP, and IUS bits, as described in Chapter 7.

A **Start** command to a channel sets the BUSY bit (xDMR5), which enables the DMA channel to operate when the Transmitter requests that its FIFO be filled, or when the Receiver requests that its FIFO be emptied. A Start command can be used to initially start up a channel, or to restart one after a Pause command.

A **Start/Continue** command operates identically to Start in Single Buffer, Array, and Linked List Modes. In Pipelined mode, it sets both the BUSY and CONT bits (xDMR7 and 5), so that after the buffer described by the xAR and xBCR, the channel goes on to another buffer that's described by NxAR and NxBCR. The channel does this by transferring the contents of NxAR to xAR, transferring the contents of NxBCR to xBCR, and clearing CONT but keeping BUSY set.

In Pipelined mode, software can use this command to start up a channel, after writing the xAR, xBCR, NxAR, and NxBCR, or to provide a subsequent buffer to a channel after writing just NxAR and NxBCR. In the latter case, software may need to take special precautions to avoid issuing the Start/Continue command while the channel is testing the CONT bit, as described in the earlier section, *Pipelined Mode.*

A **Start/Init** command operates identically to Start in Single Buffer and Pipelined modes. In Array and Linked List modes, Start/Init sets both the BUSY and INITG bits (xDMR5 and 4), so that the channel starts by loading the address and length of the initial buffer from the first entry in the array or list. Thereafter the channel transfers data as requested by the Transmitter or Receiver based on its FIFO status, just as for a Start command. Start/Init is intended only for starting an inactive channel in a new buffer.

A **Pause** command to a channel clears the BUSY bit (xDMR5), making the channel inactive until software restarts it by means of a Start command.

An **Abort** command to a channel similarly clears BUSY (xDMR5), but it also sets the SAbort bit (xDMR0), which can cause an interrupt if enabled. Software can use this command (instead of Pause) to stop a DMA channel when the channel will not be restarted to continue operation in the current buffer.

A **Reset Highest DMA IUS** command clears the IUS bit of the highest priority DMA channel that has the bit set (if any). See Chapter 7 for complete information on IUSC interrupt facilities.

DCmd				Reserved (0)		Rx/Tx Cmd	MBRE	Rx/Tx Reg	B/W	RegAddr					U/L
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 6-9. The DMA Command/Address Register (DCAR)

6.11 ADDRESS SEQUENCING

The **AddrMode** field of each DMA Mode Register (TDMR11-10 and RDMR11-10) controls how the channel sequences the buffer address from one data cycle to the next:

AddrMode	Address Sequencing
00	Channel increments xAR
01	Channel decrements xAR
10	xAR stays the same
11	Reserved; do not program

The "increment" mode is the most commonly used. The "decrement" mode is included primarily to match the capabilities of other DMA channels that were used for applications such as magnetic tape that could be read backward. The "fixed" mode is useful to transfer data to and from external FIFO devices, although the only handshaking provided for such applications is via the /WAIT/ /RDY line.

Figure 6-10 shows the shareable DMA Control Register (DCR), the fields of which may affect one or both DMA channels. Its **AddrSeg** field (DCR1-0) controls how far the DMA channels will propagate a carry when incrementing or decrementing a memory address:

AddrSeg	Address Incr/Decr Range
00	All 32 bits are affected
01	Reserved: do not program
10	The LS 16 bits are affected; A31-A16 are fixed
11	The LS 24 bits are affected; A31-A24 are fixed

This field applies to the incrementing and decrementing of addresses in data buffers and, for the Array and Linked List modes, to the incrementing of addresses while reading from arrays and lists. It applies to both the receive and transmit channels. In the latter two cases, if a channel attempts to increment or decrement an address over the implied boundary, the address instead wraps around to the opposite end of the same 64 Kbyte block (for 10) or the same 16 Mbyte block (for 11).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ChanPri		Pre Empt	ALBVO	ReArbTime		Reserved (0)		Reserved (0)		Min Off39	DCSD Out	1Wait	UAS All	AddrSeg	

Figure 6-10. The DMA Control Register (DCR)

6.12 BINARY FORMAT IN ARRAYS AND LISTS

In Array and Linked List modes the IUSC DMA channels can fetch addresses and byte counts from memory in either of the two ways that different microprocessors may store them. The **ALBVO** bit in the DMA Control Register (DCR12; the name stands for Array/List Binary Value Order) controls how the DMA channels fetch binary values from memory. If ALBVO is 0, they fetch the less-significant portions of binary values from lower-addressed memory locations, which is compatible with the Zilog Z80 and most Intel processors. If ALBVO is 1, the channels fetch the more significant portions of binary values from lower-addressed locations, which is compatible with the Zilog Z8000 and most Motorola processors. The channel fetches these values using 16-bit transfers if 16-bit (BCR2) is 1 and 8/16 (TDMR8 or RDMR8) is 0, or using 8-bit transfers if 16-bit is 0 and/or 8/16 is 1. Figures 6-11a and 6-11b show how the IUSC expects the 32-bit addresses and 16-bit counts to appear on the AD pins for the various ALBVO options, data widths, and TCB/RSB locations.

ALBVO (DCR12) = 0 (little-endian)
16BIT (BCR2) = 1 and 8/16 (xDMR8) = 0
TCBinA/L or RSBinA/L (xDMR13) = 0

Address	AD15 — AD0
N	Buffer Address 15-0
N+2	Buffer Address 31-16
N+4	Byte Count
N+6	Next Buffer or Link Address 15-0
N+8	Next Buffer or Link Address 31-16

ALBVO (DCR12) = 1 (big-endian)
16BIT (BCR2) = 1 and 8/16 (xDMR8) = 0
TCBinA/L or RSBinA/L (xDMR13) = 0

Address	AD15 — AD0
N	Buffer Address 31-16
N+2	Buffer Address 15-0
N+4	Byte Count
N+6	Next Buffer or Link Address 31-16
N+8	Next Buffer or Link Address 15-0

ALBVO (DCR12) = 0 (little-endian)
16BIT (BCR2) = 0 and/or 8/16 (xDMR8) = 1
TCBinA/L or RSBinA/L (xDMR13) = 0

Address	AD15/7 — AD8/0
N	Buffer Address 7-0
N+1	Buffer Address 15-8
N+2	Buffer Address 23-16
N+3	Buffer Address 31-24
N+4	Byte Count 7-0
N+5	Byte Count 15-8
N+6	Next/Link Address 7-0
N+7	Next/Link Address 15-8
N+8	Next/Link Address 23-16
N+9	Next/Link Address 31-24

ALBVO (DCR12) = 1 (big-endian)
16BIT (BCR2) = 0 and/or 8/16 (xDMR8) = 1
TCBinA/L or RSBinA/L (xDMR13) = 0

Address	AD15/7 — AD8/0
N	Buffer Address 31-24
N+1	Buffer Address 23-16
N+2	Buffer Address 15-8
N+3	Buffer Address 7-0
N+4	Byte Count 15-8
N+5	Byte Count 7-0
N+6	Next/Link Address 31-24
N+7	Next/Link Address 23-16
N+8	Next/Link Address 15-8
N+9	Next/Link Address 7-0

Figure 6-11a. The Order of Binary Values in Arrays and Linked Lists

6.12 BINARY FORMAT IN ARRAYS AND LISTS (Continued)

**ALBVO (DCR12) = 0 (little-endian)
16BIT (BCR2) = 1 and 8/16 (xDMR8) = 0
TCBinA/L or RSBinA/L (xDMR13) = 1**

Address	AD15 — AD0
N	Buffer Address 15-0
N+2	Buffer Address 31-16
N+4	Byte Count
N+6	TCB Control or RSB Status
N+8	TCC Length, RCC Residual, or not used
N+10	not used
N+12	Next Buffer or Link Address 15-0
N+14	Next Buffer or Link Address 31-16

**ALBVO (DCR12) = 1 (big-endian)
16BIT (BCR2) = 1 and 8/16 (xDMR8) = 0
TCBinA/L or RSBinA/L (xDMR13) = 1**

Address	AD15 — AD0
N	Buffer Address 31-16
N+2	Buffer Address 15-0
N+4	Byte Count
N+6	TCB Control or RSB Status
N+8	TCC Length, RCC Residual, or not used
N+10	not used
N+12	Next Buffer or Link Address 31-16
N+14	Next Buffer or Link Address 15-0

**ALBVO (DCR12) = 0 (little-endian)
16BIT (BCR2) = 0 and/or 8/16 (xDMR8) = 1
TCBinA/L or RSBinA/L (xDMR13) = 1**

Address	AD15/7 — AD8/0
N	Buffer Address 7-0
N+1	Buffer Address 15-8
N+2	Buffer Address 23-16
N+3	Buffer Address 31-24
N+4	Byte Count 7-0
N+5	Byte Count 15-8
N+6	Control or Status 7-0
N+7	Control or Status 15-8
N+8	TCC/RCC 7-0 or not used
N+9	TCC/RCC 15-8 or not used
N+10	not used
N+11	not used
N+12	Next/Link Address 7-0
N+13	Next/Link Address 15-8
N+14	Next/Link Address 23-16
N+15	Next/Link Address 31-24

**ALBVO (DCR12) = 1 (big-endian)
16BIT (BCR2) = 0 and/or 8/16 (xDMR8) = 1
TCBinA/L or RSBinA/L (xDMR13) = 1**

Address	AD15/7 — AD8/0
N	Buffer Address 31-24
N+1	Buffer Address 23-16
N+2	Buffer Address 15-8
N+3	Buffer Address 7-0
N+4	Byte Count 15-8
N+5	Byte Count 7-0
N+6	Control or Status 15-8
N+7	Control or Status 7-0
N+8	TCC/RCC 15-8 or not used
N+9	TCC/RCC 7-0 or not used
N+10	not used
N+11	not used
N+12	Next/Link Address 31-24
N+13	Next/Link Address 23-16
N+14	Next/Link Address 15-8
N+15	Next/Link Address 7-0

Figure 6-11b. The Order of Binary Values in Arrays and Linked Lists

6.13 CONDITIONS FOR DMA OPERATION

Several conditions must be met before the IUSC will request use of the host bus and then operate as a DMA bus master:

1. The Master Bus Request Enable bit (MBRE) in the DMA Command/Address Register (DCAR8) must be 1, and

2. the BUSY bit (xDMR5) of one or both of the DMA channels must be set due to a Start command, and

3a. the Receiver or Transmitter, associated with a Busy channel, must be requesting DMA transfer, OR

3b. a Busy channel must be in Array or Linked List mode with its INITG bit (xDMR4) set, either because of a Start/Init command or because of the termination of the previous buffer, and

4a. the BRQTP bit in the Bus Configuration Register (BCR3) must be 1, indicating that this IUSC should drive the /BUSREQ signal full-time, OR

4b. /BUSREQ must be high, and

5. the minimum time between bus requests must have elapsed. The MinOff39 bit (DCR5) controls the exact minimum time.

Once the IUSC has driven /BUSREQ low because these conditions are met, it continues to do so until one of the following occurs:

a. the duration of this period of bus mastership exceeds either of the two programmable limits in the Burst/Dwell Control Register (BDCR), or

b. one channel runs out of things to do, for example, because the serial controller negates its request because the TxFIFO becomes full or the RxFIFO becomes empty, and the other channel is not requesting to use the bus, or

c. a channel clears its BUSY bit, for any of the reasons given in an earlier section, and the other channel is not requesting to use the bus, or

d. software clears MBRE to 0, or

e. the external hardware negates the /BIN input after asserting it for at least three CLK cycles in response to this bus request.

The following sections cover various aspects of the conditions described above.

6.14 DMA REQUESTS BY THE RECEIVER AND TRANSMITTER

Aside from fetching addresses and counts from array and linked lists, the IUSC's DMA channels will only transfer data when the serial controller requests that they do so.

The Transmitter asserts its internal DMA Request to the transmit channel when:

1a. the Transmitter is enabled (in TMR1-0), and

1b. the Transmitter is not sending the end of a frame, as described below, and

1c. the transmitter is not waiting for a Trigger command, as described below, and either

1d1. the number of empty character positions in the TxFIFO is larger than the DMA Request Level value programmed into TICR15-8 after a "Select TICRhi = TxREQ Level" command, or

1d2. the USC channel is already asserting its internal Tx DMA Request and the TxFIFO is not full, or

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).

Point 1b reflects the fact that, in HDLC/SDLC, HDLC,SDLC Loop, 802.3, Transparent Bisync, the Transmitter stops requesting further DMA transfers after the Transmit DMA channel 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 possible loading of the TCB information for a new frame does not affect sending the end of the preceding frame.

Point 1c above applies when the Wait4TxTrig bit in the Channel Control Register (CCR13) is 1 in HDLC/SDLC, HDLC/SDLC Loop, 802.3, Transparent Bisync. In this case, after sending the end of a message or frame (and thus leaving the "sending the end of a frame" state noted in 1b), the Transmitter does not assert its internal Tx DMA Request until software writes a "Trigger Tx DMA" command to the RTCmd field of the Channel Command/ Address Register (CCAR15-11).

The Receiver asserts its internal DMA request to the Rx DMA channel when:

1. the Receiver is enabled (in.RMR1-0), and

2. the Receiver is not waiting for a Trigger command as described below, and either

3a. the Receiver is forcing out completed frame(s) as described below, or

3b. the number of received characters in the RxFIFO is larger than the DMA Request Level value programmed into RICR15-8 after a "Select RICRhi = TxREQ Level" command, or

3c. the Receiver is already asserting its internal Rx DMA Request, it did not just complete forcing out a frame, and the RxFIFO still has at least two characters in it on a 16-bit bus, or at least one character in it on an 8-bit bus.

"Forcing out a frame" in item 3a above applies only in HDLC/SDLC, HDLC/SDLC Loop, 802.3, Transparent Bisync, or 1553B mode, and operates differently on IUSC revisions. On the 16C32s without the SL1660 topmarking the Receiver sets a state that forces the internal Rx DMA request to be asserted when the end of a frame is received, and clears this state whenever the Rx DMA controller reads out the last character of a frame. The SL1660 product operates similarly when no Receive Status Blocks or 16-bit RSBs are enabled, except that they also clear the state when the Rx DMA channel reads out a character with Overrun status. (The latter avoids a problem called "scribbling" wherein the Receiver kept requesting DMA transfer constantly if the end of a frame arrived while the Receiver was Overrun.)

In either of the above cases, the Receiver maintains an EOF-forcing internal Rx DMA Request until the Rx DMA channel reads out a complete Receive Status Block if RSBs are enabled, else until it reads out the last received character of a frame (which is typically part of a CRC).

When 32-bit RSBs are enabled, the IUSC asserts the internal Rx DMA Request whenever the RCC FIFO is not empty, that is, when the RCCFAvail bit (CCSR14) is 1. Since these devices take the second word of a 32-bit RSB from the RCC FIFO, this approach represents a frame-forcing mechanism that operates optimally even when more than one end-of-frame character is in the RxFIFO at the same time.

"Waiting for a Trigger command" (in item 2 above) occurs only when the Wait4RxTrig bit in the Channel Control Register (CCR5) is 1 in HDLC/SDLC, HDLC/SDLC Loop, 802.3, or Transparent Bisync. In this case, after the Rx DMA channel reads out the end of a message or frame including the RSB if any (thus clearing the EOF-forcing state of 3a above), the Receiver negates the internal Rx DMA Request and does not assert it again until software writes a "Trigger Rx DMA" command to the RTCmd field of the Channel Command/Address Register (CCAR15-11). This interlock can be used to reprogram the Rx DMA channel for the next frame.

The Receive Character Counter feature cannot force the internal DMA Request nor early buffer termination.

6.14.1 Programming the DMA Request Levels

As noted in other chapters, the MS bytes of the Transmit and Receive Interrupt Control Registers (TICR and RICR) may each represent any of several registers. The content of each MS byte 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, software must first write the "Select RICRHi=/RxREQ Level" or "Select TICRHi=/TxREQ Level" command (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 is known that none of the commands 0100-0110 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 one less than the number of empty TxFIFO positions, at which the Transmitter should start asserting its internal Request to the Transmit DMA channel. The Receive DMA Request Level should be programmed with one less than the number of received characters in the RxFIFO, at which the Receiver should start asserting its internal Request to the Receive DMA channel. 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.

Both DMA Request Levels must be programmed to at least 1 when using 16 bit DMA transfers.

It is good programming practice to follow the writing of Request Level(s) with writing a "Select RICRHi=FIFO Status" command to the RCSR, and/or a "Select TICRHi=FIFO Status" command to the TCSR as applicable, to protect the Request Level(s) from inadvertent modification when other parts of the software change the IA bits in the LSbyte of the RICR or TICR.

Code that writes or reads a DMA Request threshold must ensure that no interrupts will occur between the time it writes the "Select xICRHi=REQ Level" command to the TCSR or RCSR, and when it writes or reads the value in the TICR or RICR, if such interrupts can lead to other code writing a different Select command (for TSA data, the FIFO Fill level, or interrupt threshold) to the same Command/ Status Register.

Note that a Purge Tx FIFO (or Purge Rx and Tx FIFO) command can make the Transmitter immediately assert its DMA request.

6.15 INTER-CHANNEL OPERATION AND PRIORITY

If/when both DMA channels are active, three fields in the DMA Control Register (DCR) control how they share use of the external bus.

The **ChanPri** field (DCR15-14) selects the relative priority of the two channels for use of the bus, that is, which one gets to use the bus first if both are requesting at the time of a bus grant:

ChanPri	Channel Priority
00	Transmit channel has priority
01	Receive channel has priority
10	Alternating: whichever channel uses the bus first in one bus grant, has the will lower priority in the next one.
11	Reserved; do not program

The **PreEmpt** bit (DCR13) selects whether the higher-priority channel (as defined by the ChanPri field) can take over control of the bus if it starts requesting control while the lower-priority one is using the bus. If PreEmpt is 0, once a channel starts using the bus it continues to do so until one of four events occurs:

1. it fills or empties its FIFO, or
2. it reaches the time limit for use of the bus, or
3. it clears its BUSY bit, or
4. software clears MBRE.

If PreEmpt is 1, the lower-priority channel relinquishes bus control to the higher one, after it completes any bus cycle that was in progress when the higher-priority channel started requesting.

Software should avoid programming pre-emptive receive channel priority unless it (1) uses 32-bit TCBs, (2) is running on an IUSC with the SL1660 topmarking and (3) sets the UnderWait bit in the Transmit Command/Status Register (TCSR11) to 1. The latter combination makes the Transmitter delay starting to send a frame until either the TxFIFO is full or an entire Tx frame has been placed in the TxFIFO.

PreEmpt=1 and ChanPri=01 will allow the Rx DMA channel to seize control of the bus right after the Tx channel has placed the first one or two characters of a new Tx frame in the TxFIFO. Unless UnderWait is 1 and 32-bit TCBs are used on the IUSC this make the Transmitter start sending the frame. If the Rx DMA channel is doing a lengthy operation like fetching a new array or list entry followed by storing a full RxFIFO, the Transmitter will encounter an Underrun condition before the Rx DMA channel is done.

When PreEmpt is 0, the **ReArbTime** field (DCR11-DCR10) determines when the IUSC reselects which channel is using the bus:

ReArbTime	Channel Re-arbitration Time
00	The IUSC reselects the active channel at the start of each bus grant, and one channel can use the bus after the other within the same period of bus control.
01	Once a DMA channel has started using the bus, it continues to do so until its part of the serial controller request has released its request, even if this takes several periods of bus control. However, once this occurs, the other channel can use this bus for the duration of the same period of bus control.
1x	Reserved; do not program

When PreEmpt is 1, ReArbTime must be programmed as 00.

6.16 BUS ACQUISITION AND RELEASE TIMING

Figure 6-12 shows typical bus acquisition and release sequences. If the IUSC is asserting /BUSREQ when it first samples /BIN low at a rising edge of CLK, it starts preparing to take control of the bus, otherwise it drives /BOUT low. Two CLK cycles after first sampling /BIN low with /BUSREQ low, the IUSC samples /BIN again. If /BIN is still low, then from the next rising edge the IUSC places the more-significant half of the initial memory address on the AD lines, and starts driving /UAS, /AS, /DS, R//W, /RD, /WR, plus S//D and D//C if the DCSDOut bit in the DMA Control Register (DCR4) is 1. From the next rising edge of CLK, it drives /UAS to low. There is one more CLK period of address setup between the AD lines and the first rising edge of /UAS after bus acquisition, than there is for subsequent /UAS pulses (if any) within the same period of bus control.

The IUSC will release control of the bus if the bus grant on /BIN goes false/high while it is using the bus. A following section, 'Master Bus Cycles', shows the timing for the withdrawal of /BIN.

Typically, the IUSC makes the decision to release the bus during a bus cycle, which is the case shown in the latter part of Figure 6-12. It drives or releases /BUSREQ to high from the rising CLK edge that is 4.5 CLK periods after the falling edge from which it drives /DS and (/RD or /WR) to High. It also releases the /UAS, /AS, /DS, R//W, /RD, and /WR lines, and if necessary the AD lines, S//D, and D//C, from the same rising edge.

If the IUSC makes the decision to release the bus later than is needed to achieve the timing shown in Figure 6-12, it still drives or releases /BUSREQ to high from the same rising edge on CLK at which it releases the various other bus signals.

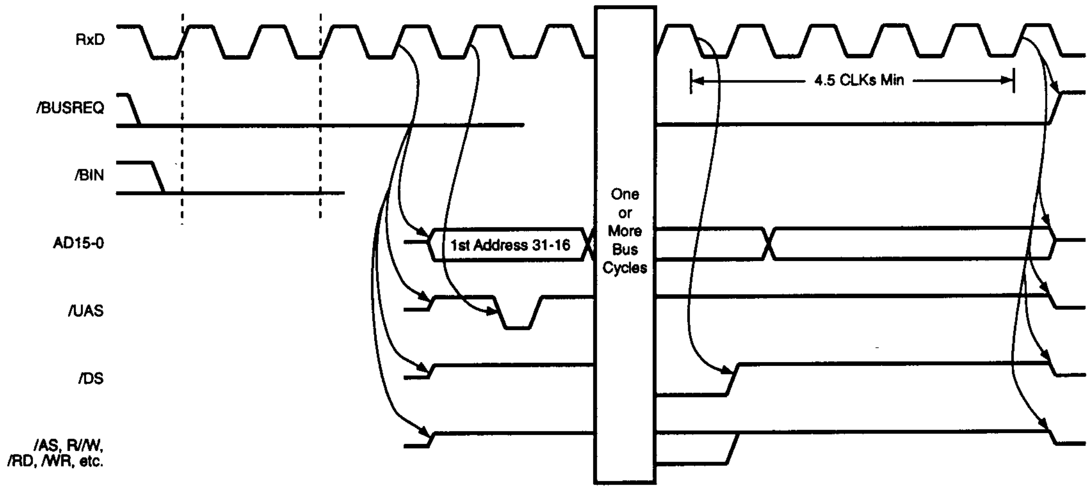


Figure 6-12. Bus Acquisition and Release

6.17 BUS CYCLE OPTIONS

Three bits in the shareable DMA Control Register (DCR; see Figure 6-10) affect how the DMA channels operate as bus masters—that is, how they act once they have control of the bus. This information is presented both here and in Chapter 2, *Bus Interfacing*.

6.17.1 D//C, S//D Status Output

The **DCSDOut** bit (DCR4) controls whether the IUSC drives the S//D and D//C pins when it is the bus master. If DCSDOut is 1, the IUSC drives S//D Low for Tx channel operations and High for Rx channel cycles, and drives D//C High during transfers of serial data and Low during array or linked-list fetching. When this bit is 1, on a multiplexed bus S//D and D//C cannot be connected directly to any of the address/data lines with the AD pins, and external drive on the S//D and D//C pins must be tri-stated (released) while the IUSC is the bus master.

If neither external logic nor monitoring equipment (like a logic state analyzer) has any use for the information described above, software can program DCSDOut as 0. In this case the IUSC never drives S//D and D//C, and these pins can be connected directly to AD lines on a multiplexed bus, or can be driven full-time by external drivers on a non-multiplexed bus.

6.17.2 Wait Insertion

If the **1Wait** bit (DCR3) is 1, the IUSC extends the data portion of each master bus cycle by 1 CLK period. This allows use of slower memories for a given CLK frequency, or use of a faster CLK frequency with a particular memory type. Signaling on /WAIT//RDY can be used to extend master bus cycles regardless of the state of this bit. When 1Wait is 1 the IUSC starts actively sampling /WAIT//RDY one CLK period later than when it is 0.

6.17.3 /UAS Frequency

Since the DMA channels maintain 32-bit addresses but have only a 16-bit external bus, they present each address in two parts. They signal the availability of the more significant half of an address with a strobe on the /UAS pin, and signal the LS half of each address with a strobe on /AS. The **UASAll** bit (DCR2) controls how often the channels present the more-significant half of the address. If UASAll is 1, every master bus cycle includes presentation of the more-significant half of the address on the AD15-AD0 pins, with a low-going pulse on /UAS. This means that every bus cycle takes at least four cycles of CLK.

If UASAll is 0, the IUSC includes a /UAS sequence only in cycles that meet one or more of the following criteria:

1. in the first cycle after taking control of the bus from another master, or
2. in the first cycle after switching from one channel to the other, or
3. in Pipelined mode, in the first cycle after switching from one buffer to the next, or
4. in Array or Linked List mode, in the first data cycle after switching between data buffer accesses and array/list accesses, or vice-versa, or
5. in the first cycle after incrementing a memory address results in a carry out from A15, even if the AddrSeg field (DCR1DCR0) is 10 so that the carry is blocked.

When the IUSC includes a /UAS sequence in a bus cycle, the cycle is at least 4 CLK periods long, while if it does not, the bus cycle can be as short as three CLKs.

UASAll should be programmed as 1 only if required by unusual external hardware. For example, if the IUSC and another bus master share an upper-address latch and the other bus master can insert cycles between IUSC cycles within the same bus grant, UASAll would want to be 1.

6.18 MASTER BUS CYCLES

Figures 6-13 and 6-14 show DMA Read and Write cycles with the IUSC as bus master and the UASAll bit (DCR2) set to 0. In each case two cycles are shown. The first includes a /UAS strobe and is four CLK periods long. The second does not include a /UAS and is three CLK periods long. In both cases, to achieve these minimum bus-cycle times, the /WAIT//RDY signal should setup and hold in the "ready" state, around the falling edge of CLK that follows the rising edge of /AS. As noted in the preceding section, if the 1Wait bit in the DMA Control Register (DCR3) is 1, the IUSC delays its first sampling of /WAIT/RDY by one CLK period. In this case, bus cycles that include a /UAS strobe are at least five CLK periods long, and those that do not are at least four CLKs long. For each falling edge of CLK at which the IUSC samples /WAIT//RDY as "Not Ready", it extends the length of the cycle by one CLK period.

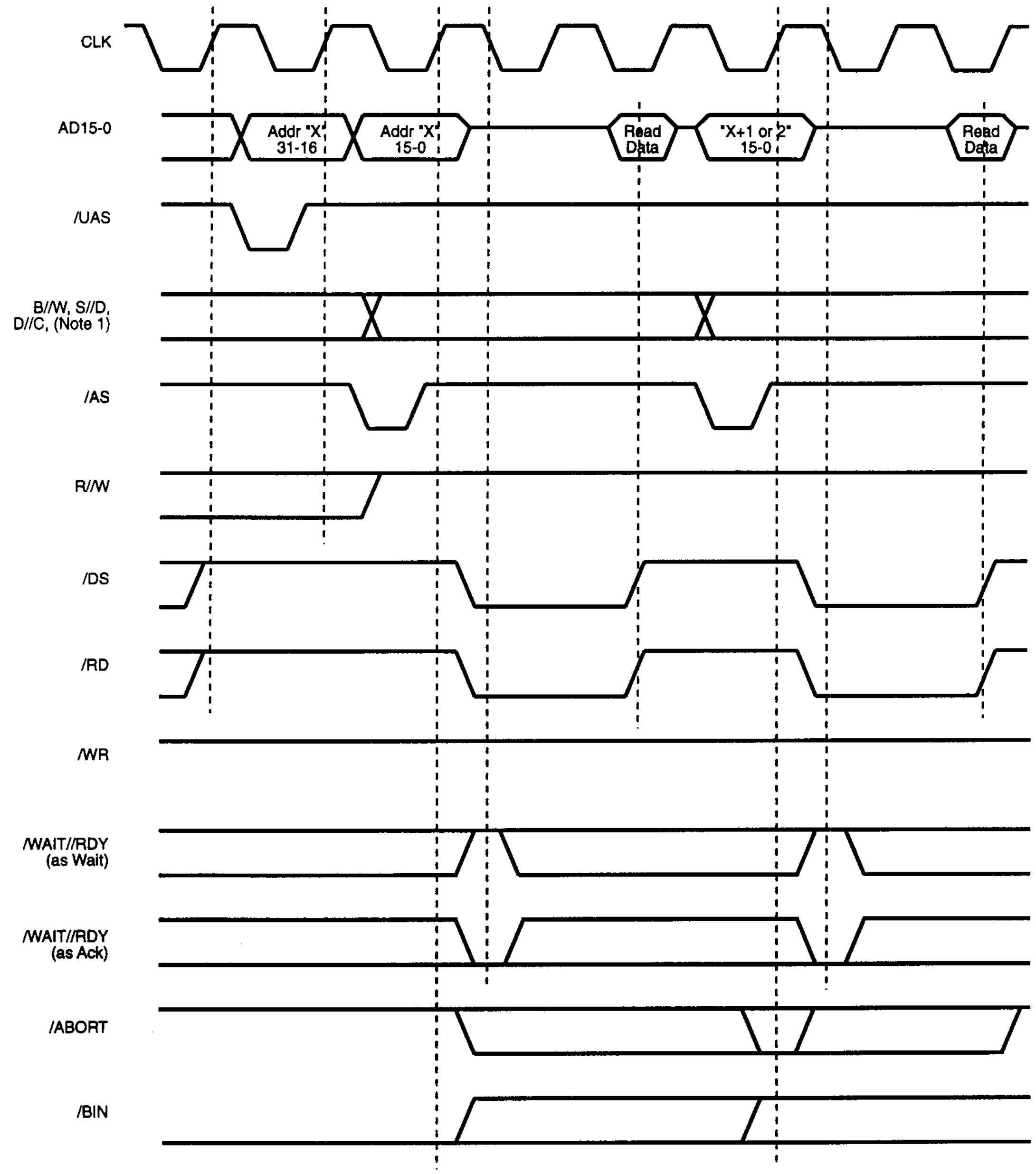
As shown in the Figures, the "ready" state is High for "Wait" signaling and Low for "Acknowledge" signaling. The kind of signaling on /WAIT//RDY depends on whether the S//D pin was High or Low at the time that software wrote the Bus Configuration Register (BCR) after the last Reset.

Note also that in DMA Read operations, read data from memory should set up and hold around the rising edge of the /DS and /RD lines. This gives the memory subsystem some extra access time as compared to having to set up to the falling CLK edge from which the IUSC drives /DS and /RD to high, but this characteristic must be considered in the memory design and the /WAIT//RDY logic. Given that the Figures assume the UASAll bit (DCR2) is 0, the "possibly low" states at start of the /DS and /RD or /WR traces in the Figures illustrate the inter-cycle timing when there is a carry out of A15 during address incrementing (that is, when address "X" has 16 low-order zeroes). When UASAll is 0, this is the only case in which a cycle that includes a /UAS will directly follow another cycle. The other occasions that force a /UAS strobe in the middle of a period of bus control all involve several CLK period delays for internal "housekeeping" functions, between the preceding cycle and the cycle that includes the /UAS, as follows. (All of the values above are in addition to the one CLK cycle needed for the /UAS sequence itself.)

Condition Forcing /UAS:	**Number of extra CLK periods before the /UAS cycle**
Inter-channel switch	4
Pipelined mode buffer switch	8
Serial Data Transfer to Array or List Fetch	8
Array or List Fetch to Serial Data Transfer	8

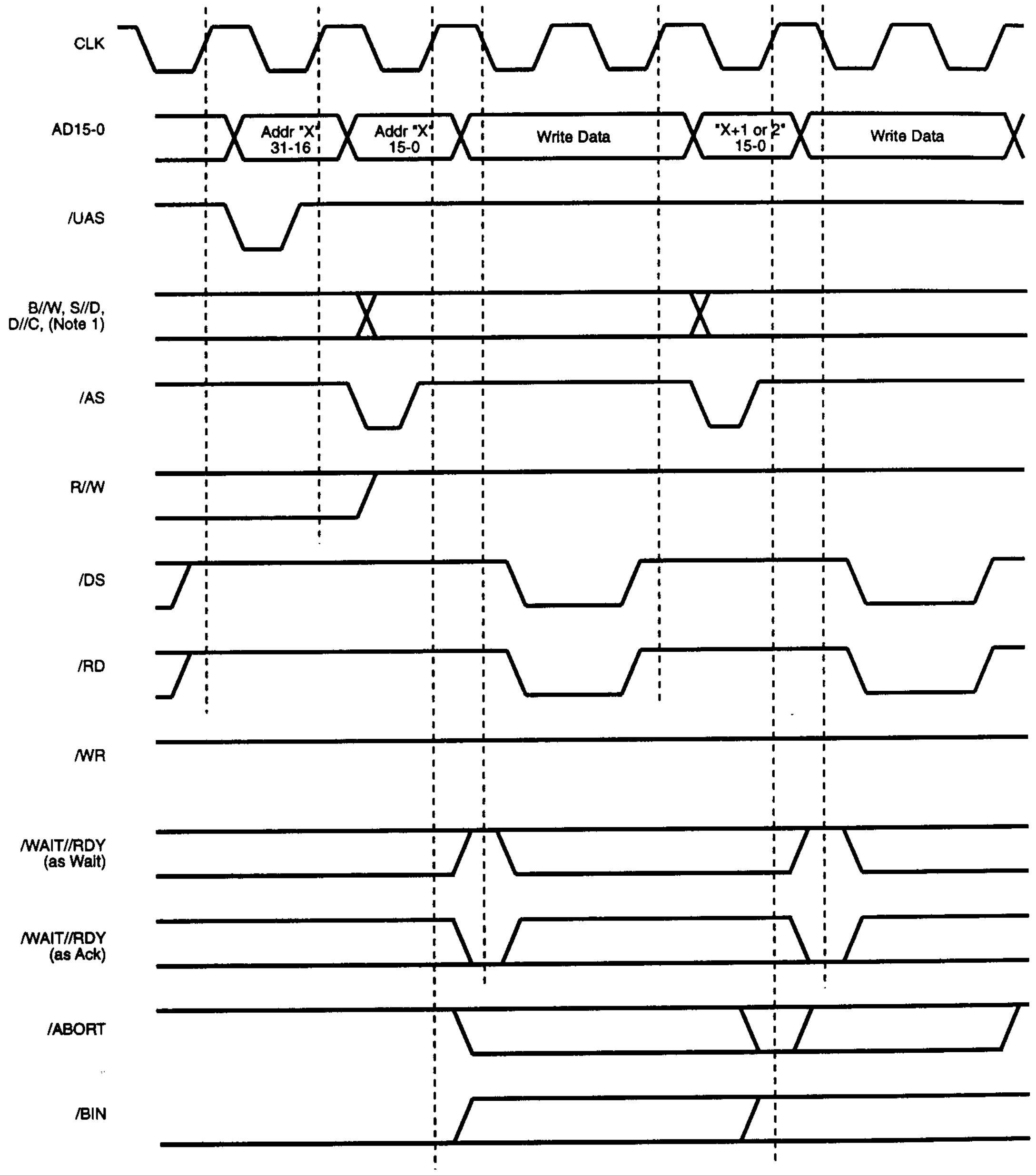
The last two signals in Figures 6-13 and 6-14 illustrate the timing of the /ABORT and /BIN inputs. Both inputs are effective at the rising edge of CLK that immediately precedes the falling edge of CLK at which the IUSC samples /WAIT//RDY "ready". If DMA operation is to be aborted in the bus cycle shown for "address X+1 or 2", then /ABORT must set up and hold Low around that edge. To force the IUSC to give up bus control after the bus cycle shown for "address X+1 or 2", /BIN must set up High to that edge.

6.18 MASTER BUS CYCLES (Continued)



Note 1: S//D and D//C are driven only if the DCSDOut bit in the DMA Control Register (DCR4) is 1.

Figure 6-13. Master Read Cycles



Note 1: S//D and D//C are driven only if the DCSDOut bit in the DMA Control Register (DCR4) is 1.

Figure 6-14. Master Write Cycles

6.19 BUS OCCUPANCY THROTTLING

In some systems it may be necessary or desirable to limit the IUSC's use of the host bus. For example, in a dedicated control system it may be necessary to guarantee a maximum interrupt response time, and IUSC DMA activity may be a factor in the interrupt response time of the host processor. As well as responding to an external withdrawal of its bus grant as described in the preceding section, the IUSC allows its DMA activity to be programmatically limited. This can be done in terms of the maximum duration that the part will use the bus for each bus grant. Bus activity can also be limited in terms of the minimum time that the IUSC will stay off the bus before requesting it again.

The **MinOff39** bit in the DMA Control Register (DCR5) controls the minimum time that the IUSC will keep /BUSREQ inactive/high. If MinOff39 is 0, this minimum is seven CLK periods, while if MinOff39 is 1, the IUSC will not "rerequest" the bus for at least 39 CLKs.

The shareable Burst/Dwell Control Register (BDCR) controls the maximum duration for which the IUSC will use the bus, per bus grant. Figure 6-15 shows the BDCR. If the **MaxXfers** field (BDCR15-8) is non-zero, the IUSC treats its contents as the largest number of bus transactions it will do in response to one bus grant. If the **MaxCLKs** field (BDCR7-0) is non-zero, the IUSC will use the bus for up to eight times that number of CLK periods, in response to each grant. If both values are zero (as they are after Reset), for each bus grant the IUSC will use the bus until it runs out of things to do, e.g., until the RxFIFO is empty and/or the TxFIFO is full. If both values are non-zero, the IUSC limits its bus usage according to whichever one expires first.

Reaching one of these limits never terminates a cycle in progress; a limit takes effect only after a cycle is over. If a time-out on the length of a cycle is desired, for example to detect an access to a non-existent memory address, it must be implemented externally using the /ABORT pin.

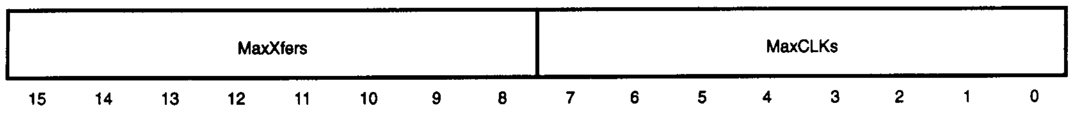


Figure 6-15. The Burst/Dwell Control Register (BDCR)

6.20 OPERATING FLOWCHARTS

The three parts of Figure 6-16 show the complete operating sequence of an IUSC DMA channel, as a function of its various operating modes. These flowcharts rely on certain assumptions to simplify them:

1. They do not show individual byte vs 16-bit accesses to array and linked-list entries. For example, fetching a buffer address from an array or linked list is shown as a single operation, while in real life it may consist of two 16-bit cycles or four 8-bit cycles. On the other hand, the difference between byte and 16-bit accesses to data buffers is shown explicitly, because the difference between these two alternatives can be important.

2. They do not show the process of requesting the bus by asserting /BUSREQ and then waiting for /BIN to be asserted.

3. They do not shown the effects of bus occupancy throttling by means of the BDCR, nor the effects of revoking /BIN nor asserting /ABORT, nor those of any software commands other than Start and Start/Init.

4. They assume that all TCBs are 32 bits long. The 16 bit option is of little or no use.

5. They assume the AddrMode field is 00 so that memory addresses are incremented.

The Figures show how long the IUSC takes to perform the various major steps shown, as "N1" through "N14" cycles of the CLK pin. Since the flowcharts are meant to show "what a channel does", three kinds of delays are not shown therein: the time to get on and off the bus nor the time to switch between the DMA channels within the same period of bus occupancy.

The symbol "extW" stands for wait states imposed by external signalling on the /WAIT//RDY line, over and above the automatic ones imposed if the "1Wait" bit (DCR3) is set to 1.

Symbol	Number of CLK cycles to perform the step on a 16-bit bus with the 8/16 bit=0
N1	7 (+2 if 1Wait) (+1 if UASAll) + 2*extW
N2	3 (+1 if 1Wait) (+1 if UASAll) + extW
N3	3 (+1 if 1Wait) (+1 if UASAll) + extW
N4	6 (+2 if 1Wait) (+2 if UASAll) + 2*extW
N5	3 (+1 if 1Wait) (+1 if UASAll) + extW
N6	8
N7	4 (+1 if 1Wait) + extW for 1st access to buffer or first of bus occupancy period else 3 (+1 if 1Wait) (+1 if UASAll) + extW
N8	8
N9	8
N10	4 (+1 if 1Wait) + extW
N11	3 (+1 if 1Wait) (+1 if UASAll) + extW
N12	3 (+1 if 1Wait) (+1 if UASAll) + extW
N13	3 (+1 if 1Wait) (+1 if UASAll) + extW
N14	3 (+1 if 1Wait) (+1 if UASAll) + extW if Rx channel and RSBinA/L = 1 else 4 (+1 if 1Wait) + extW

6.20 OPERATING FLOWCHARTS (Continued)

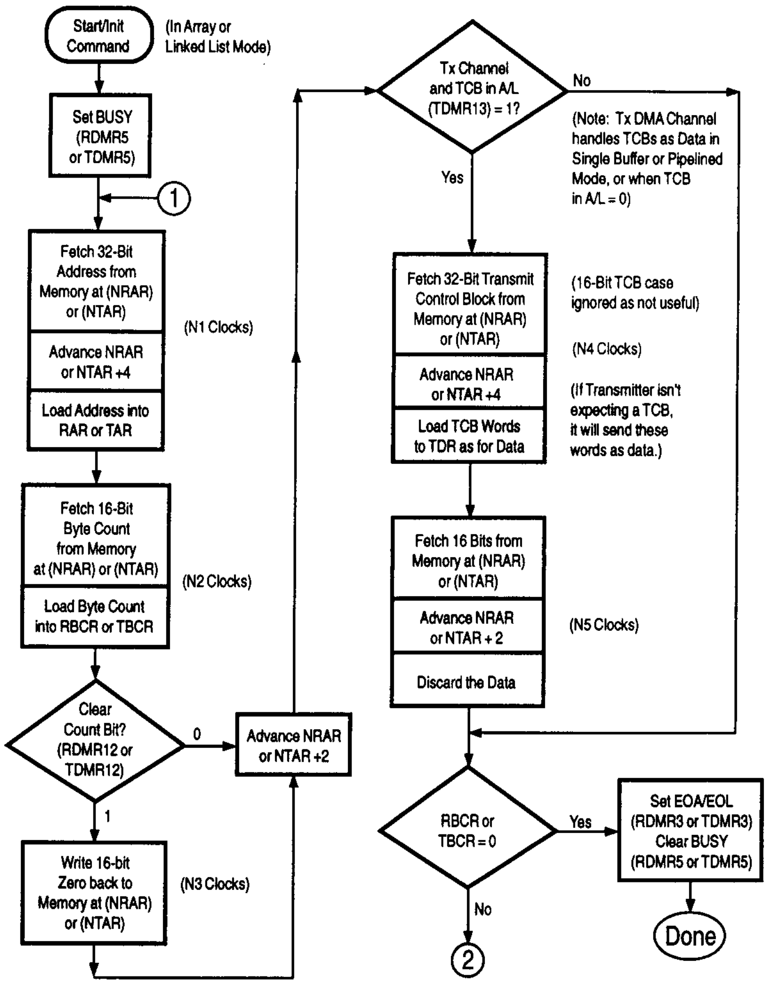

Start/Init Command
(In Array or Linked List Mode)
Set BUSY (RDMR5 or TDMR5)
1
Fetch 32-Bit Address from Memory at (NRAR) or (NTAR)
Advance NRAR or NTAR +4
Load Address into RAR or TAR
(N1 Clocks)
Fetch 16-Bit Byte Count from Memory at (NRAR) or (NTAR)
Load Byte Count into RBCR or TBCR
(N2 Clocks)
Clear Count Bit? (RDMR12 or TDMR12)
0
1
Advance NRAR or NTAR +2
Write 16-bit Zero back to Memory at (NRAR) or (NTAR)
(N3 Clocks)
Tx Channel and TCB in A/L (TDMR13) = 1?
No
Yes
(Note: Tx DMA Channel handles TCBs as Data in Single Buffer or Pipelined Mode, or when TCB in A/L = 0)
Fetch 32-Bit Transmit Control Block from Memory at (NRAR) or (NTAR)
Advance NRAR or NTAR +4
Load TCB Words to TDR as for Data
(16-Bit TCB case ignored as not useful)
(N4 Clocks)
(If Transmitter isn't expecting a TCB, it will send these words as data.)
Fetch 16 Bits from Memory at (NRAR) or (NTAR)
Advance NRAR or NTAR + 2
Discard the Data
(N5 Clocks)
RBCR or TBCR = 0
Yes
No
Set EOA/EOL (RDMR3 or TDMR3) Clear BUSY (RDMR5 or TDMR5)
Done
2

Figure 6-16a. DMA Channel Operation Flowchart (1 of 3)

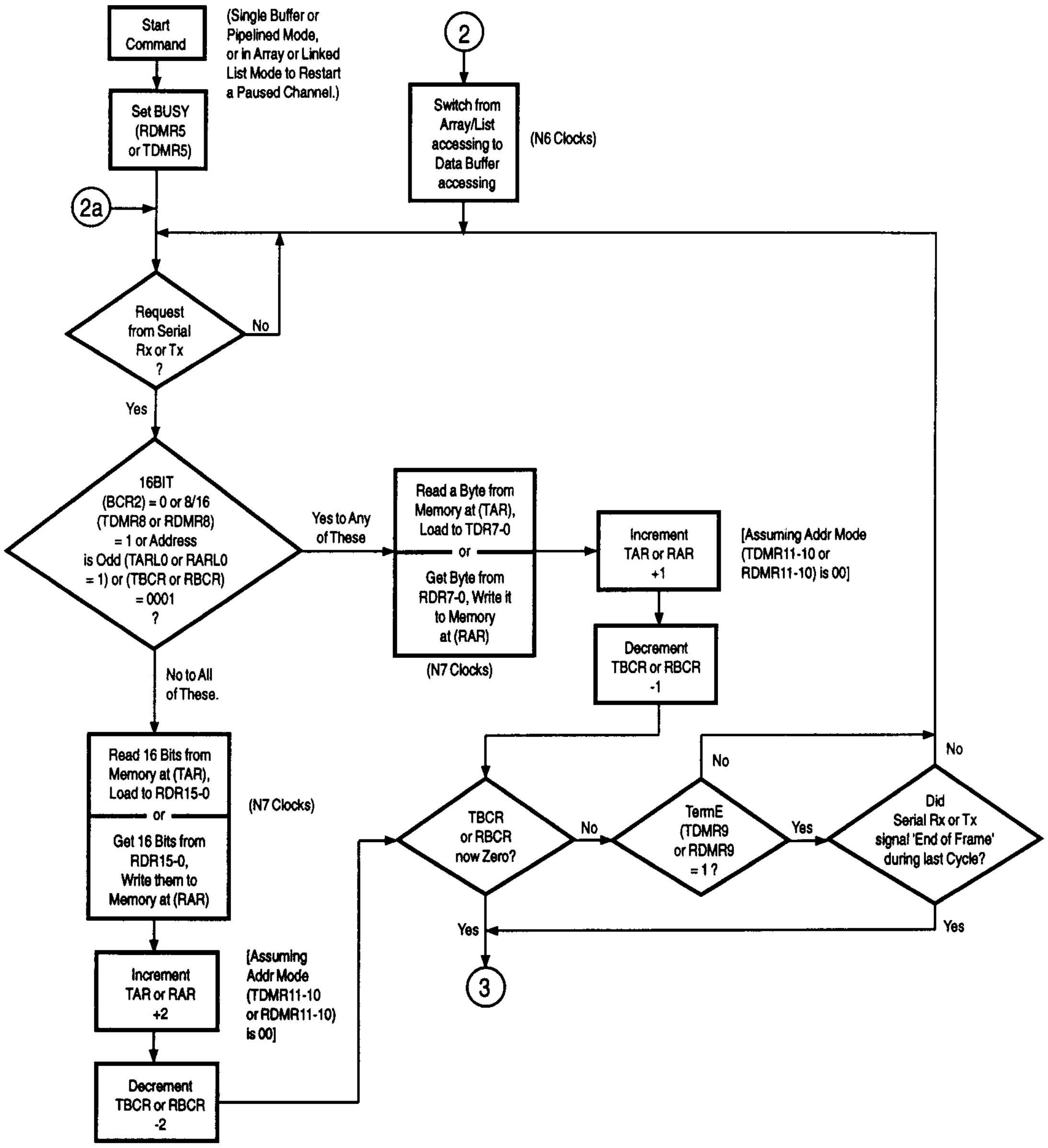

Start Command
(Single Buffer or Pipelined Mode, or in Array or Linked List Mode to Restart a Paused Channel.)
Set BUSY (RDMR5 or TDMR5)
2a
2
Switch from Array/List accessing to Data Buffer accessing
(N6 Clocks)
Request from Serial Rx or Tx ?
No
Yes
16BIT (BCR2) = 0 or 8/16 (TDMR8 or RDMR8) = 1 or Address is Odd (TARL0 or RARL0 = 1) or (TBCR or RBCR) = 0001 ?
Yes to Any of These
No to All of These.
Read a Byte from Memory at (TAR), Load to TDR7-0
or
Get Byte from RDR7-0, Write it to Memory at (RAR)
(N7 Clocks)
Increment TAR or RAR +1
[Assuming Addr Mode (TDMR11-10 or RDMR11-10) is 00]
Decrement TBCR or RBCR -1
Read 16 Bits from Memory at (TAR), Load to RDR15-0
or
Get 16 Bits from RDR15-0, Write them to Memory at (RAR)
(N7 Clocks)
Increment TAR or RAR +2
[Assuming Addr Mode (TDMR11-10 or RDMR11-10) is 00]
Decrement TBCR or RBCR -2
TBCR or RBCR now Zero?
No
TermE (TDMR9 or RDMR9 = 1 ?
Yes
No
Did Serial Rx or Tx signal 'End of Frame' during last Cycle?
No
Yes
Yes
3

Figure 6-16b. DMA Channel Operation Flowchart (2 of 3)

6.20 OPERATING FLOWCHARTS (Continued)

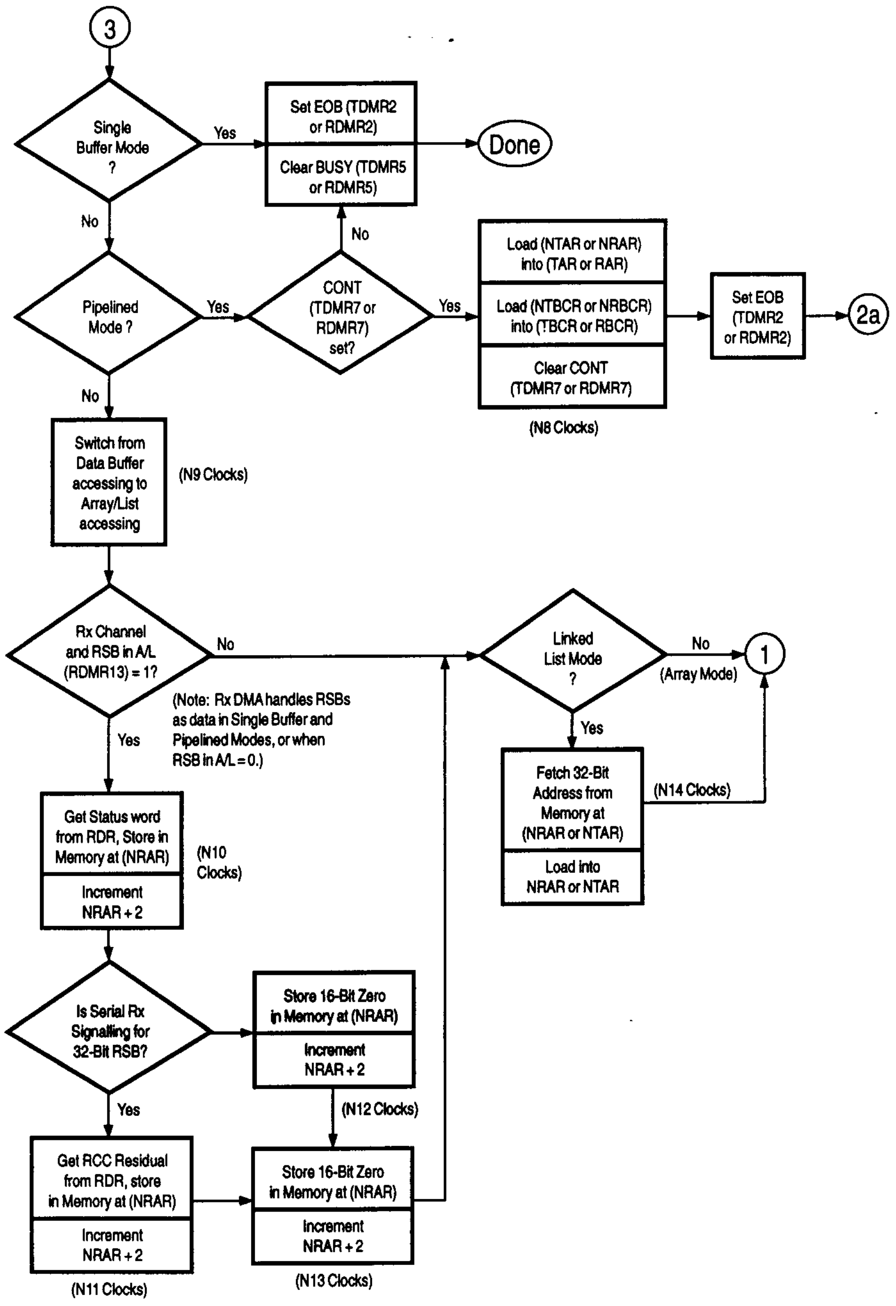

3
Single Buffer Mode ?
Yes
No
Set EOB (TDMR2 or RDMR2)
Clear BUSY (TDMR5 or RDMR5)
Done
Pipelined Mode ?
Yes
No
CONT (TDMR7 or RDMR7) set?
No
Yes
Load (NTAR or NRAR) into (TAR or RAR)
Load (NTBCR or NRBCR) into (TBCR or RBCR)
Clear CONT (TDMR7 or RDMR7)
(N8 Clocks)
Set EOB (TDMR2 or RDMR2)
2a
Switch from Data Buffer accessing to Array/List accessing
(N9 Clocks)
Rx Channel and RSB in A/L (RDMR13) = 1?
No
Yes
(Note: Rx DMA handles RSBs as data in Single Buffer and Pipelined Modes, or when RSB in A/L = 0.)
Linked List Mode ?
No
(Array Mode)
1
Yes
Fetch 32-Bit Address from Memory at (NRAR or NTAR)
Load into NRAR or NTAR
(N14 Clocks)
Get Status word from RDR, Store in Memory at (NRAR)
Increment NRAR + 2
(N10 Clocks)
Is Serial Rx Signalling for 32-Bit RSB?
Yes
Store 16-Bit Zero in Memory at (NRAR)
Increment NRAR + 2
(N12 Clocks)
Get RCC Residual from RDR, store in Memory at (NRAR)
Increment NRAR + 2
(N11 Clocks)
Store 16-Bit Zero in Memory at (NRAR)
Increment NRAR + 2
(N13 Clocks)

Figure 6-16c. DMA Channel Operation Flowchart (3 of 3)

6.21 ARRAY AND LINKED LIST FETCHING STATUS

In Array and Linked List modes, the INITG and GLink bits in the TDMR or RDMR provide a first level of information by which software can read the state of a channel that is fetching information from an array or linked list. More detailed status about array and linked-list fetching is available in the shareable DMA Array Count Register (DACR). Figure 6-17 shows the DACR, which contains separate **RALCnt** and **TALCnt** fields (DACR7-DACR4 and DACR3-DACR0 respectively) for the two channels.

The DMA channels sequence these fields from all ones downward as they go through the steps of fetching array and list entries and transferring data to or from the buffers that the entries describe.

In Linked List mode a channel sequences TALCnt or RALCnt with GLink=0 while fetching the buffer address and count, and then goes through further states with GLink=1 while fetching the next entry address.

A Z16C32 DMA channel will use 16-bit transfers to access the array or linked-list if 16-bit (BCR2) is 1 and 8/16 (TDMR8 or RDMR8) is 0. If 16-bit=0 and/or 8/16=1, the channel will use 8-bit transfers and will thus go through more states.

The TCBinA/L (TDMR13), RSBinA/L (RDMR13), and ClearCount (TDMR12 or RDMR12) bits also affect the state sequence that the DMA channels follow and show in TALCnt and RALCnt.

Table 6-1 shows all the values that TALCnt and RALCnt can assume, and their meaning, with notes indicating in which contexts each state can occur.

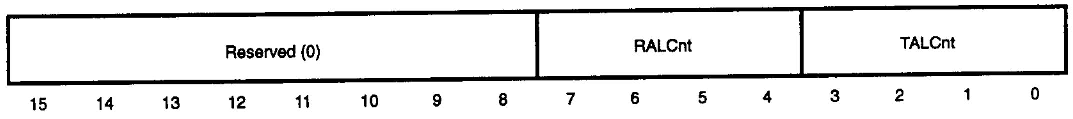


Figure 6-17. The DMA Array Count Register (DACR)

6.21 ARRAY AND LINKED LIST FETCHING STATUS (Continued)

Table 6-1. States of a DMA Channel

INITG	GLink	TALCnt RALCnt	State of the Channel	This State Can Occur For: Data Width	Mode	Clear Count	TCBInA/L RSBInA/L
0	0	0000	Single Buffer or Pipelined Mode	8/16	SB/P	(na)	(na)
1	0	1111	Array fetch pending,	8/16	A	0/1	0/1
			First list fetch pending, or		L		
			Link Address fetched		L		
1	0	1110	1st byte of Buffer Address fetched	8	A/L	0/1	0/1
1	0	1101	1st half of Buffer Address fetched	8/16	A/L	0/1	0/1
1	0	1100	3rd byte of Buffer Address fetched	8	A/L	0/1	0/1
1	0	1011	Buffer Address fetched	8/16	A/L	0/1	0/1
1	0	1010	1st byte of Byte Count fetched	8	A/L	0/1	0/1
1	0	1001	Byte Count fetched	8/16	A/L	0/1	0/1
0	0	1001	Receiving into data buffer, or	8/16	A/L	0	0/1
			Transmitting from data buffer			0	0
1	0	1000	1st byte of Byte Count cleared to zero	8	A/L	1	0/1
1	0	0111	Byte Count cleared to zero	8/16	A/L	1	0/1
0	0	0111	Receiving into data buffer, or	8/16	A/L	1	0/1
			Transmitting from data buffer			1	0
1	0	0110	1st byte of TCB fetched, or 1st byte of RSB (or zero) stored	8	A/L	0/1	1
1	0	0101	TCB control word fetchedm, or RSB status word (or zero) stored	8/16	A/L	0/1	1
1	0	0100	3rd byte of TCB fetched (ignored if 16-bit TCB) or 3rd byte of RSB (or zero) stored	8	A/L	0/1	1
1	0	0011	TCC frame length fetched (ignored if 16-bit TCB) or RCC residual (or zero) stored	8/16	A/L	0/1	1
1	0	0010	11th byte of entry read/ignored (Tx), or 11th byte of entry cleared to zero (Rx)	8	A/L	0/1	1
1	0	0001	6th word of entry read/ignored (Tx), or 6th word of entry cleared to zero (Rx)	8/16	A/L	0/1	1
0	0	0001	Transmitting from data buffer	8/16	A/L	0/1	1
1	1	1111	Link Address Fetch Pending	8/16	L	0/1	0/1
1	1	1110	1st byte of Link Address fetched	8	L	0/1	0/1
1	1	1101	1st half of Link Address fetched	8/16	L	0/1	0/1
1	1	1100	3rd byte of Link Address fetched	8	L	0/1	0/1

CHAPTER 7
INTERRUPTS

7.1 INTRODUCTION

The interrupt subsystem of the IUSC 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 is easy to interface the IUSC to interrupt other processors as well. This chapter describes the IUSC's interrupt capabilities and how to use them in various system applications.

The IUSC dedicates four pins to interrupts. It uses the /INT output to request an interrupt on the host processor. The /INTACK input signals 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 be detected at the IUSC, software can simulate such cycles.)

The Interrupt Enable In (IEI) and Out (IEO) pins 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. Alternatively, external interrupt control logic can process interrupt requests in a round-robin or dynamic-priority fashion among one or more IUSCs and/or other peripheral devices.

7.2 INTERRUPT ACKNOWLEDGE DAISY CHAINS

Figure 7-1 shows an interrupt acknowledge daisy chain. The highest-priority (or only) 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. Unless the IUSC is the only daisy-chainable device that can request an interrupt, 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 (or only) daisy-chainable interrupting device, which has its IEO pin left unconnected.

With the IUSC 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.

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 IUSC's interrupt subsystem provides the most efficient interrupt handling possible.

7.2 INTERRUPT ACKNOWLEDGE DAISY CHAINS (Continued)

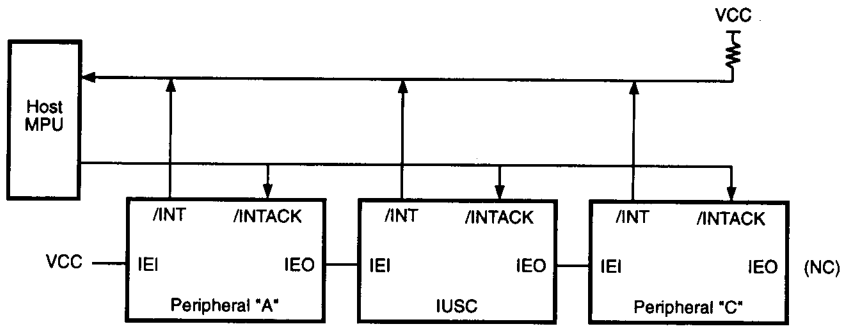


Figure 7-1. An Interrupt Daisy Chain

7.3 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 IUSCs all having similar baud rates and serial traffic, the strict priority that is inherent in a daisy chain might endanger proper interrupt servicing for the device(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 acknowledgments among the devices on a round-robin (rotating-priority) basis. Such schemes target "fairness" rather than priority in interrupt servicing among the devices.

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 IUSCs, and are determined dynamically rather than being known as the system is being programmed. (An IUSC'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 IUSC's data rate.

Both rotating-priority and dynamic-priority systems can be arranged as shown in Figure 7-2. The interrupt control logic maintains the IEI inputs of the IUSCs high most or all of the time, so that they can assert their /INT outputs. The logic may simply OR the /INT outputs of the various IUSCs 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 IUSCs to different processor levels.

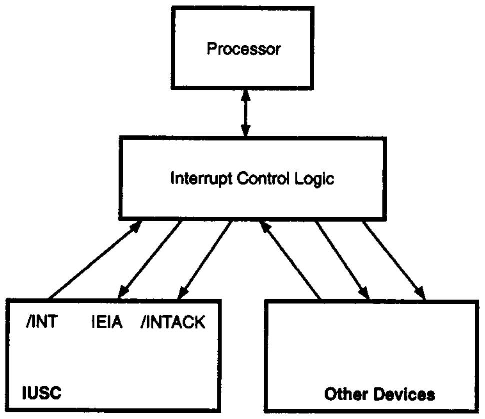


Figure 7-2. External Interrupt Control

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 /INTACK signal and possibly other bus control signals. Or, it can simply present the cycle only to the selected IUSC, typically using a single pulse on /INTACK.

7.4 MEGACELLS, TYPES, AND SOURCES

Internally, the IUSC uses a daisy-chaining scheme much like that described earlier. Thus its benefits are available, to some extent, even in systems that do not include any other Zilog-compatible peripherals. At the first level, the IUSC's serial and DMA sections ("megacells") have separate interrupt subsystems. Their request lines are logically OR'ed to make the /INT output. The IUSC's IEI pin is connected to the IEI input of the serial controller; the serial controller's IEO output is internally connected to the DMA controller's IEI input, and the DMA controller's IEO output is routed to the IUSC's IEO pin. This arrangement means that serial controller interrupts have higher priority than DMA controller interrupts. The two sections also have fully independent interrupt vectors.

The IUSC carries interrupt daisy-chaining further, to a second level of internal resolution. Each section or megacell includes several interrupt "types" — six for the serial controller and two for the DMA section. The various types in each megacell are arranged in a fixed priority order on an internal daisy-chain. Many of the types include several separate interrupt stimuli or "sources".

Figure 7-3 shows all of the interrupt types and sources in the IUSC, arranged with the highest priority types at the top.

7.4 MEGACELLS, TYPES, AND SOURCES (Continued)

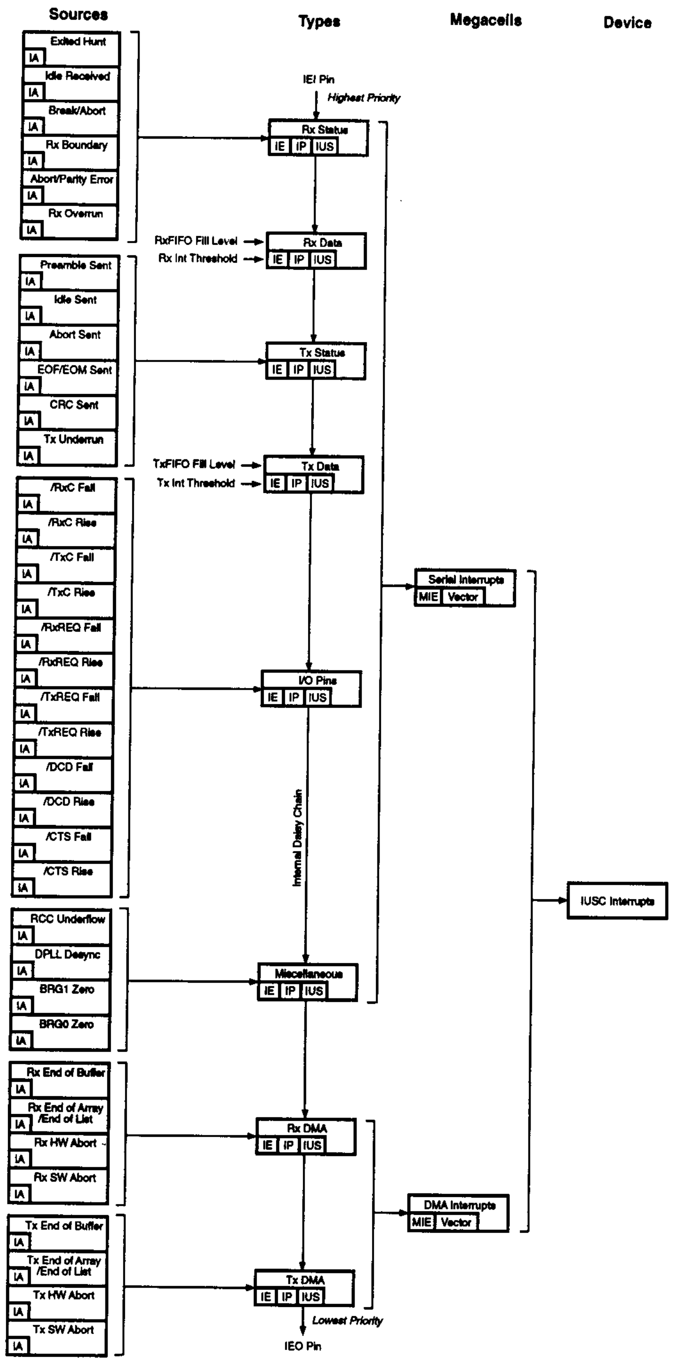


Figure 7-3. The IUSC Interrupt Subsystem

7.5 INTERNAL INTERRUPT OPERATION

Figure 7-4 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 IUSC's interrupt logic rather than the exact logic; it is included only as an aid to understanding the interrupt subsystem.

Each individual source has an associated register bit that we will 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.

Each source and its IA bit are logically ANDed. A rising edge on the logical OR of these terms, for all the sources in the type, sets an "Interrupt Pending" (IP) bit for the type. For the IUSC and other 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 7-4 will show that setting of IP is delayed if an "armed" source comes true during an interrupt acknowledge cycle, but that is not particularly important for understanding the IUSC's interrupt subsystem.

A second register bit associated with each type is the Interrupt Enable or IE bit. This bit is 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 when the associated IP bit is set, 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 is currently requesting an interrupt. (This includes types in higher-priority external devices and higher-priority types within the IUSC.) 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 7-4 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 megacell must be set to 1.

7.5 INTERNAL INTERRUPT OPERATION (Continued)

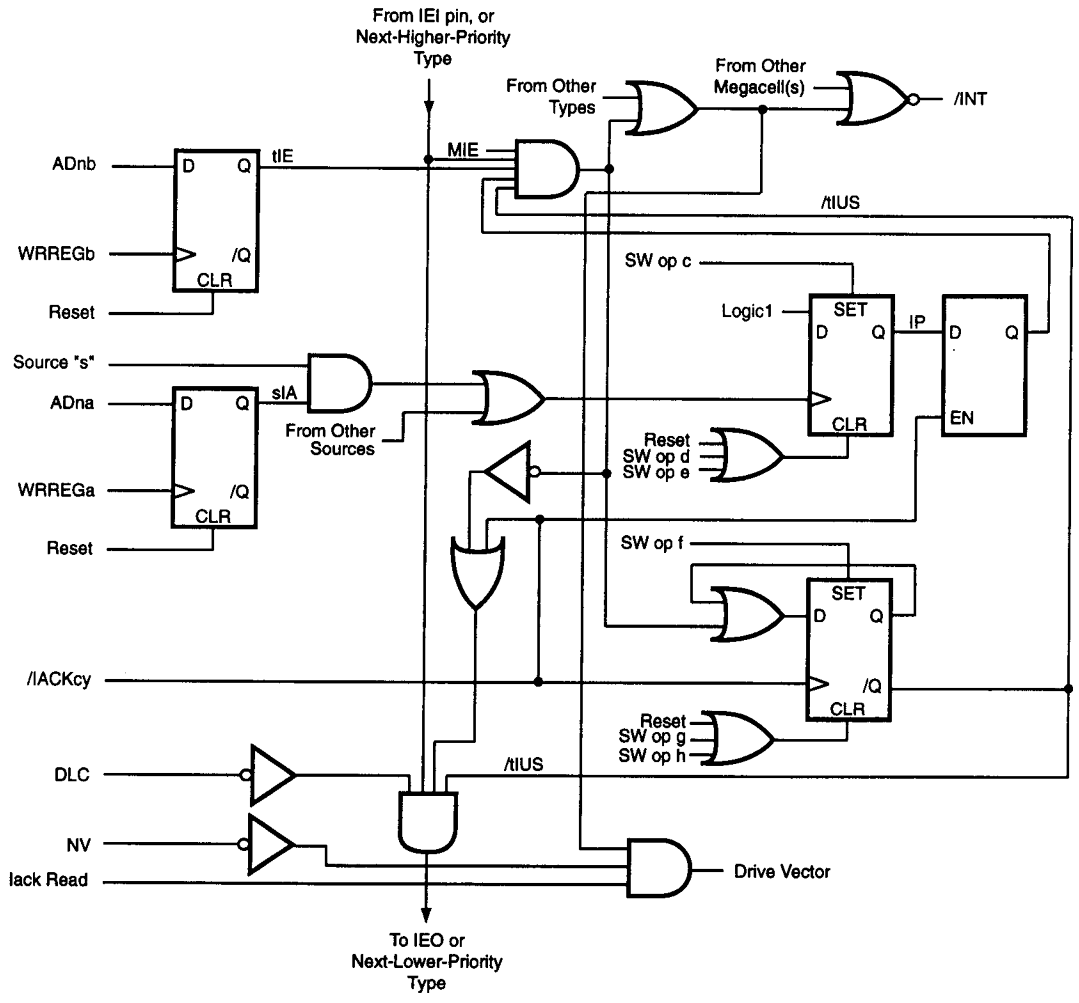


Figure 7-4. A Model of the Interrupt Logic for Source "s" and Type "t"

7.6 DETAILS OF THE MODEL

The IA and IE bits appear near the left side of Figure 7-4, as D-type flip-flops that capture the state of an AD line when software writes a specific register. The IP bit appears as a D-type flip-flop and a latch that are set "by hardware" as described above; software can set and clear the latch. The signal labeled /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 through 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 try to clear IP during an interrupt service routine, and typically also has to clear IUS, there are often several ways to clear these bits, as shown by the multiple "SW op" signals for these functions in the Figure. One thing not shown in Figure 7-4 is how the typical command "Reset Highest IUS" is implemented—including this function would have considerably increased the complexity of the logic.

The two downward-pointing gates in Figure 7-4 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 megacell; if/when DLC is 1 the megacell's IEO output 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 labeled "Drive Vector" controls when the megacell places an interrupt vector on the data bus during an interrupt acknowledge cycle. There is a register bit No Vector (NV) in each megacell; NV=1 prevents driving a vector. The bus interface logic derives the signal "IACK Read" from R//W and /DS, /RD, or /INTACK, depending in part on a field in the Bus Configuration Register (BCR) that specifies how /INTACK works. Typically IACK Read is true during the latter part of the time that /IACKcy is true. The megacell provides a vector on AD7-AD0 while IACK Read is true, if NV is 0 and any of the types in the megacell is the highest priority interrupting type.

To keep its complexity reasonable, Figure 7-4 does not include the mechanism by which the content of a returned interrupt vector can reflect the identity of the highest-priority interrupting type within the megacell.

7.7 INTERRUPT OPTIONS IN THE BCR

Two fields in the Bus Configuration Register (BCR) affect the interrupt subsystem. The following is also presented in Chapter 2, *Bus Interfacing*.

The **IAckMode** field (BCR5-BCR4) tells the IUSC how the host processor drives the /INTACK pin. 00 makes the IUSC capture the state of /INTACK at the start of each bus cycle. It does this at rising edges on /AS on a bus with multiplexed addresses and data, or at falling edges on /DS or /RD on a non-multiplexed bus.

This field should be written as 01 if /INTACK carries a single low-active pulse during an interrupt acknowledge cycle.

The 10 value in IAckMode is reserved and should not be programmed.

IAckMode should be written as 11 if /INTACK carries a double pulse during an interrupt acknowledge sequence. This mode is compatible with several Intel microprocessors.

If the **/IRQTP** bit (BCR1) is 0, the IUSC drives its /INT pin in a totem-pole fashion (both high and low). If /IRQTP is 1, the IUSC drives /INT in an open-drain fashion (low only) so that the request can be wire-ORed, in which case an external pull-up resistor should be provided.

7.8 INTERRUPT ACKNOWLEDGE CYCLES

The IUSC does not require Interrupt Acknowledge cycles. The system designer can simply pull up the /INTACK pin, and software can read the Interrupt Pending (IP) bits in the Daisy Chain Control Register (DCCR) and the Set DMA Interrupt Register (SDIR), which are described in later sections.

Even if the host processor does Interrupt Acknowledge cycles, the IUSC does not have to provide a vector. If IEI is high and the NV bit in the Interrupt Control Register (ICR) or DMA Interrupt Control Register (DICR) is 1, the IUSC 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 7-5 shows the kind of interrupt acknowledge cycle that the IUSC expects when the IAckMode field (BCR5-4) is 00, 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 signaling. Since the timing is the same for either strobe, Figure 7-5 simply shows a trace labeled "/DS or /RD".)

If the IUSC samples /INTACK 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 IUSC 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-AD0 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 IUSC is not requesting an interrupt, it does not respond to the cycle.

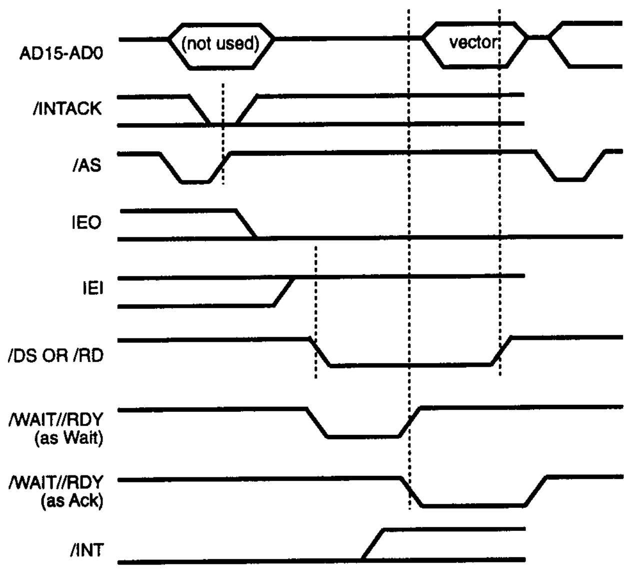


Figure 7-5. An Interrupt Acknowledge Cycle with IACKMODE=00 on a Multiplexed Bus

7.8 INTERRUPT ACKNOWLEDGE CYCLES (Continued)

Figure 7-6 shows the kind of interrupt acknowledge cycle that the IUSC expects when the IAckMode field (BCR5-4) is 00, 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 signaling. Since the timing is identical for either strobe, Figure 7-6 simply shows a trace labeled "/DS or /RD".)

Here the IUSC freezes its internal interrupt state in response to a falling edge on /INTACK; 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 /INTACK 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 IUSC 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-AD0 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 IUSC is not requesting an interrupt, it does not respond to the cycle.

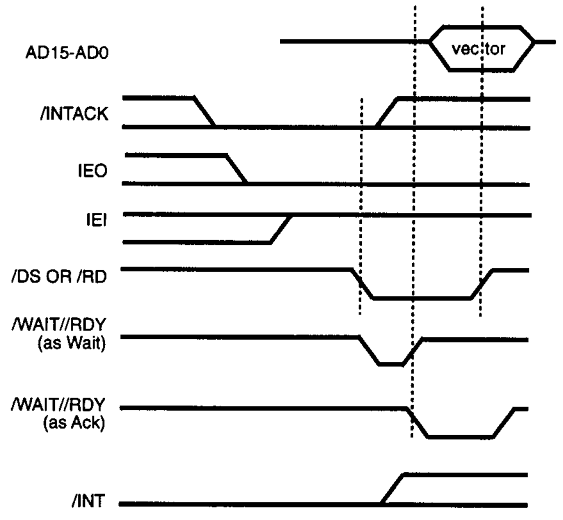


Figure 7-6. An Interrupt Acknowledge Cycle with IACKMODE=00 on a Non-Multiplexed Bus

Figure 7-7 shows the kind of interrupt acknowledge cycle that the IUSC expects when the IAckMode field is 01. Here a single pulse on /INTACK 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 nonmultiplexed 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 /INTACK 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 the IUSC or another device should respond to each interrupt acknowledge cycle. Instead, external logic like that shown in Figure 7-2 must decide which requesting device 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 requesting devices' 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 /INTACK. If IEI is high and the IUSC is requesting an interrupt at that point, it responds to /INTACK by driving a vector onto the AD7-AD0 pins and driving /WAIT//RDY appropriately to signal when the vector is valid. If IEI is low at the leading/falling edge of /INTACK, and/or if the IUSC is not requesting an interrupt at that point, it does not respond to the cycle.

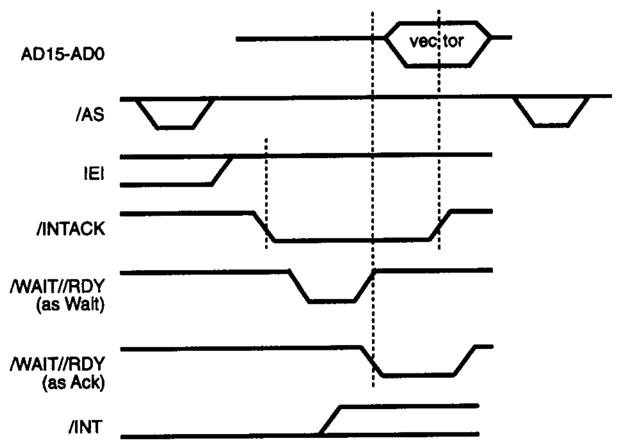


Figure 7-7. An Interrupt Acknowledge Cycle with IACKMODE=01

7.8 INTERRUPT ACKNOWLEDGE CYCLES (Continued)

Figure 7-8 shows the kind of interrupt acknowledge cycle that the IUSC expects when the IAckMode field is 11. Here, two consecutive low pulses on /INTACK 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 /INTACK. These minima are similar to those between /AS and /DS or /RD in register read cycles.

In this "double pulse mode" the IUSC keeps an internal state bit that distinguishes the two /INTACK pulses in each pair. The IUSC freezes its internal interrupt state in response to the first falling edge on /INTACK. 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 IUSC does not otherwise respond to the first cycle.

In this mode the IEI pin must be valid for a specified setup time before /INTACK goes low for the second pulse. If IEI is high at this point and the IUSC is requesting an interrupt, it responds to the second /INTACK pulse by setting the IUS bit of its highest-priority requesting type of interrupt, driving a vector onto the AD7-AD0 pins, and driving /WAIT//RDY appropriately to signal when the vector is valid. If IEI is low at the leading edge of /INTACK, and/or if the IUSC is not requesting an interrupt, it does not respond to the cycle.

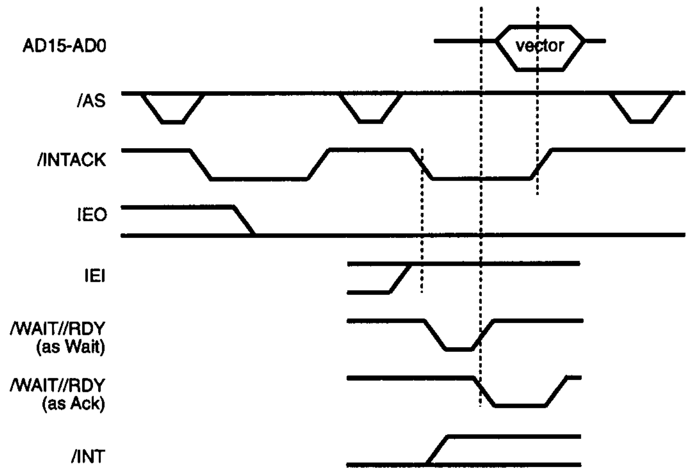


Figure 7-8. An Interrupt Acknowledge Cycle with IACKMODE=11

7.9 INTERRUPT ACKNOWLEDGE VS READ CYCLES

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

- With IAckMode=00 on a multiplexed bus, /INTACK acts like an address line. When an IUSC samples /INTACK low at a rising edge on /AS, it ignores the address on the AD lines.
- On a non-multiplexed bus with IAckMode=00, each leading edge of /RD or /DS captures the state of /INTACK.
- With IAckMode=00 and /DS signaling, the state of R//W does not matter for a cycle in which the IUSC samples /INTACK low. (In other cycles R//W differentiates Read cycles from Writes.)
- When the /WAIT//RDY pin carries the Wait function, the IUSC asserts the pin during interrupt acknowledge cycles, but never does so during register Read or Write cycles.
- When /WAIT//RDY carries the Acknowledge function, the IUSC 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.

7.10 SERIAL CONTROLLER INTERRUPT TYPES

The serial controller section of the IUSC 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.

7.10.1 Receive Status Interrupt Sources and IA Bits

Any of six interrupt sources can set the 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 7-9. The following descriptions of the RCSR status bits are similar to those in the *Detailed Status in the RCSR* section of Chapter 5.

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 should not 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 is marked with RxBound status. Such marking reflects an address character in Nine-Bit 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.

Abort/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, or, in HDLC/SDLC mode with the QAbort bit (RMR8) set, a character that was followed by an Abort sequence.

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 is marked with Overrun status. The character so marked is the first one that arrived while the FIFO was full. An intermediate number of characters after it may have been lost. See 'Handling Overruns and Underruns' in Chapter 5 for more information.

7.10 SERIAL CONTROLLER INTERRUPT TYPES (Continued)

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

Figure 7-9. The Receive Command/Status Register (RCSR)

"RTSA data" if last RCSR15-12 command 4-7 was 4 "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	TCOR Sel
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 7-10. The Receive Interrupt Control Register (RICR)

As described in Chapter 5, 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, Abort/PE, and RxOver, this action also clears the RCSR bit. The Break/Abort (in Async modes) and IdleRcved bits in the RCSR do not 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 7-10 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, Abort/PE, and Overrun conditions affect how the corresponding RCSR bits operate, as described in Chapter 5.

In order to ensure that future interrupts are requested properly when more than one Receive Status condition is Armed in the RICR, a Receive Status interrupt service routine must clear all of the IA bits in the RICR and then set the desired ones again, after it has cleared the RS IP bit and the sources it has serviced.

When software wants to change the IA bits in the RICR after the register is first initialized, it should write only the LSbyte of the register rather than all 16 bits, to avoid inadvertently changing a threshold or time slot assigner setting in the MSbyte.

7.10.2 Receive Data Interrupts

This interrupt type has only one source, so there is 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 when a character is received, 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 the 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 IUSC should start requesting a Receive Data interrupt, minus one, to the MSbyte of the Receive Interrupt Control Register (RICR). For example, if the IUSC 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.

It is good programming practice to follow these two steps with writing a "Select RICRHi=FIFO Status" command to the RCSR, to protect the Request Level from inadvertent modification when other parts of the software change the IA bits in the LSbyte of the RICR.

Code that writes or reads the Receive Data Interrupt Request threshold must ensure that no interrupts will occur between the time it writes the "Select RICRHi=/INT Level" command to the RCSR, and when it writes or reads the value in the RICR, if such interrupts can lead to other code writing a different Select command (for TSA data, the FIFO Fill level, or DMA request threshold) to the RCSR.

Figure 7-11 shows a sample service routine for Receive Data interrupts. While it is 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, or message), 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.

7.10 SERIAL CONTROLLER INTERRUPT TYPES (Continued)

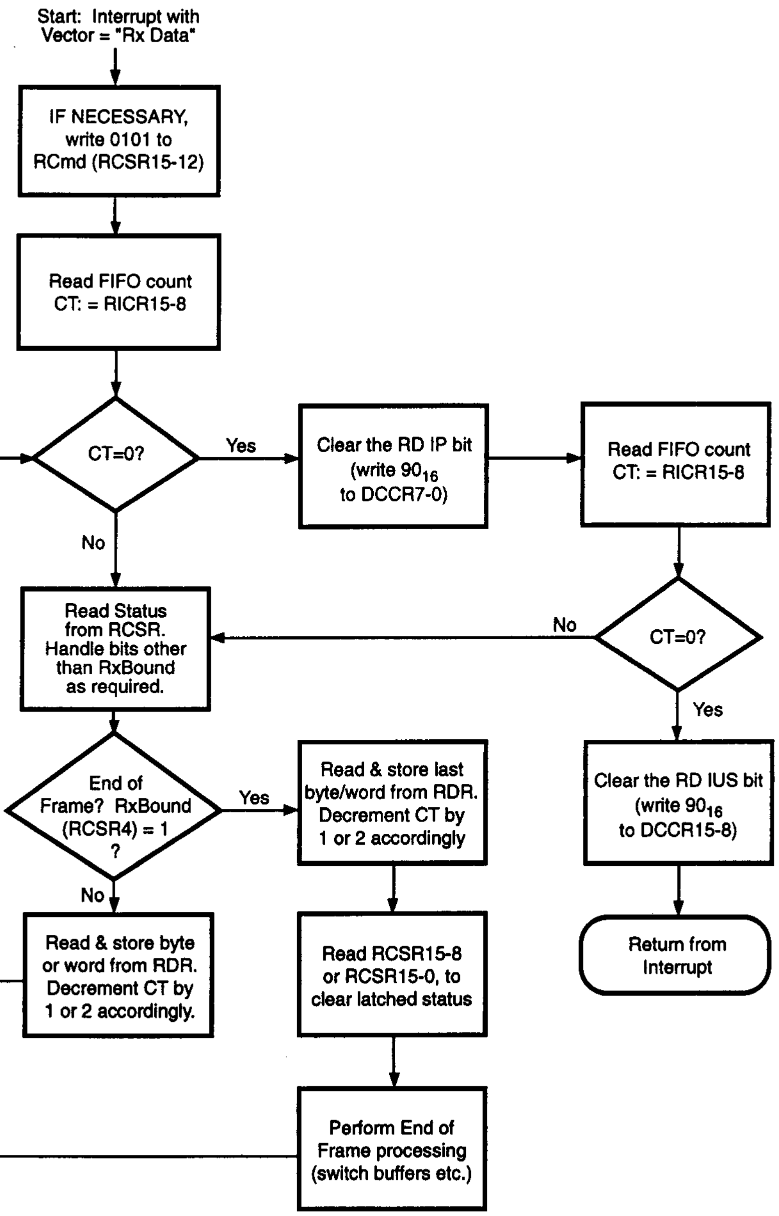

Start: Interrupt with Vector = "Rx Data"
IF NECESSARY, write 0101 to RCmd (RCSR15-12)
Read FIFO count CT: = RICR15-8
CT=0?
Yes
No
Clear the RD IP bit (write 90_{16} to DCCR7-0)
Read FIFO count CT: = RICR15-8
CT=0?
No
Yes
Read Status from RCSR. Handle bits other than RxBound as required.
End of Frame? RxBound (RCSR4) = 1 ?
Yes
No
Read & store last byte/word from RDR. Decrement CT by 1 or 2 accordingly
Clear the RD IUS bit (write 90_{16} to DCCR15-8)
Read & store byte or word from RDR. Decrement CT by 1 or 2 accordingly.
Read RCSR15-8 or RCSR15-0, to clear latched status
Return from Interrupt
Perform End of Frame processing (switch buffers etc.)

Figure 7-11. A Sample Service Routine for Receive Data Interrupts

7.10.3 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 7-12. The following descriptions of the TCSR bits are similar to those in the *Detailed Status in the TCSR* section of Chapter 5:

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 is 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.

Once one of these TCSR bits is 1, it must be cleared to 0 by writing a 1 to that bit position in TCSR.

In order to ensure that future interrupts are requested properly when more than one Transmit Status condition is Armed in the TICR, a Transmit Status interrupt service routine must clear all of the IA bits in the TICR and then set the desired ones again, after it has cleared the TS IP bit and the sources it has serviced.

Each of these six sources has a separate **Interrupt Arm (IA)** bit in the LSbyte of the Transmit Interrupt Control Register (TICR). Figure 7-13 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.

When software wants to change the IA bits in the TICR after the register is first initialized, it should write only the LSbyte of the register rather than all 16 bits, to avoid inadvertently changing a threshold or time slot assigner setting in the MSbyte.

TCmd				Under Wait	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 7-12. The Transmit Command/Status Register (TCSR)

"TTSA data" if last TCSR15-12 command 4-7 was 4 "TxFIFO fill level" if last TCSR15-12 command 4-7 was 5 "Tx Int Req level" if last TCSR15-12 command 4-7 was 6 "Tx DMA Req 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 7-13. The Transmit Interrupt Control Register (TICR)

7

7.10.4 Transmit Data Interrupts

This interrupt type has only one source, so there is no need for an IA bit for it. The interrupt logic sets the Transmit Data IP bit whenever the Transmitter finished sending a character, and the number of empty character positions in the TxFIFO is equal to the value programmed as the "Transmit Data Interrupt Request Level". Since this event is immediately followed by taking the next character out of TxFIFO, the net effect if that the IUSC requests a Transmit Data interrupt when there are more empty character positions in the TxFIFO than the Request Level value.

If transmitted data is to be handled by the Tx 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 IUSC should start requesting a Transmit Data interrupt, minus one, to the MSbyte of the Transmit Interrupt Control Register (TICR). For example, if the IUSC 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 TICR15-8.

It is good programming practice to follow these two steps with writing a "Select TICRHi=FIFO Status" command to the TCSR, to protect the Request Level from inadvertent modification when other parts of the software change the IA bits in the LSbyte of the TICR.

Code that writes or reads the Transmit Data Interrupt Request threshold must ensure that no interrupts will occur between the time it writes the "Select TICRHi=/INT Level" command to the TCSR, and when it writes or reads the value in the TICR, if such interrupts can lead to other code writing a different Select command (for TSA data, the FIFO Fill level, or DMA request threshold) to the TCSR.

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

- it had not been doing so,
- the IEI pin is high,
- its TD IE and MIE bits are 1, and
- its TD IUS and all higher-priority IUS bits are 0.

As with all IUSC 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; neither of these bits is cleared by simply writing data into the TxFIFO.

7.10.5 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, namely /RxC, /TxC, /RxREQ, /TxREQ, /DCD, and /CTS. The following description is similar to that in the *Edge Detection and Interrupts* section of Chapter 4.

Software can program the IUSC 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 7-14 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 IUSC 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 the IUSC detects an edge that is enabled in the SICR, it records the event in an internal latch that is not directly accessible in the IUSC's register map. Instead, as shown in Figure 7-15, 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 IUSC 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 does not matter what value software writes to the "data" bits.

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 7-14. 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 7-15. The Miscellaneous Interrupt Status Register (MISR)

7

7.10.6 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 7-15. The following descriptions repeat some information from Chapters 4 and 5:

RCCUnder. If the RCCUnder IA bit is 1, the IUSC 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, the IUSC 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, the IUSC 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, the IUSC 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), as shown in Figure 7-14. 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 will not set the corresponding MISR bit, and thus the condition can't cause interrupts. Clearing an IA bit does not clear the corresponding bit in MISR.

7.11 SERIAL IP AND IUS BITS

Software can read, set, and clear the **Interrupt Pending (IP)** and **Interrupt Under Service (IUS)** bits, for all six interrupt types in the serial controller, in the Daisy-Chain Control Register (DCCR). Figure 7-16 shows the DCCR. The MSbyte deals only with the IUS bits, while the LSbyte deals with the IP bits but allows clearing of 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 MS bits of each byte always read as 00. When software writes the DCCR, the two MS bits of each byte can represent a command that is applied to the type(s) selected by ones written in the six LS bits of that byte. DCCR15-14 are an IUS Op field that the IUSC 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-DCCR6 are an IP Op field that the IUSC 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

The "clear both" option sounds efficient but in general is useful only during initialization sequences. The later section "Software Requirements" describes how an interrupt service routine should clear an IP bit before examining the device status, but should delay clearing the IUS bit until the ISR is (nearly) over.

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 IUSC 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 DCCR5-0 is 0, the IUSC 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 a serial controller interrupt service routine, regardless of which type initiated the routine. If software writes the Reset Highest Serial IUS command (00010) to the RTCmd field (CCAR15-11), it clears the highest-priority IUS bit that is set in the serial controller.

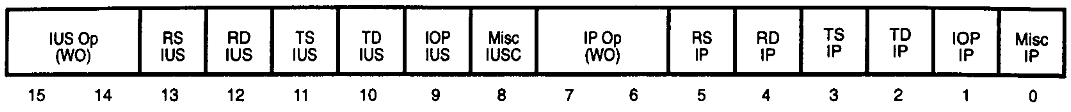


Figure 7-16. The Daisy Chain Control Register (DCCR)

7.12 SERIAL INTERRUPT ENABLE BITS

Software can read, set, and clear the **Interrupt Enable (IE)** bits for all six interrupt types in the serial controller, in the LSbyte of its Interrupt Control Register (ICR). Figure 7-17 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 IUSC applies to any and all IE bits selected by ones written to ICR5-0. The IUSC 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

7.13 SERIAL CONTROLLER INTERRUPT OPTIONS

Figure 7-17 shows that the MSbyte of the Interrupt Control Register (ICR) contains control bits that apply to all interrupts from the serial controller. 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 any of the types in the serial controller to request an interrupt.

Whenever the **Disable Lower Chain** bit (DLC; ICR14) is 1, the serial controller forces its IEO output low, so that neither the IUSC's DMA channels, nor external devices further down the daisy chain, can request interrupts nor respond to interrupt acknowledge cycles.

If the **No Vector** bit (NV; ICR13) is 1, the IUSC 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 serial controller. However, in such a case the IUSC 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 IUSC returns during an interrupt acknowledge cycle in which the highest-priority requesting type is in the serial controller, identifies the type or not. Such vector modification can be enabled for all types in the serial controller, or only for those above a selected priority:

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 and Status
1110	Receive Status only
1111	No types

If the contents of VIS allow the highest-priority type, that is 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 IUSC returns the 8-bit vector exactly as the host software programmed it.

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

Figure 7-17. The Interrupt Control Register (ICR)

7.14 SERIAL INTERRUPT VECTORS

The vectors returned by the IUSC for interrupts from the serial controller section are independent of those from the DMA section. Software can read and write serial interrupt vector information in the Interrupt Vector Register (IVR). This register is also the basis of the vector that the IUSC returns during an interrupt acknowledge cycle in which the highest priority requesting type is in the serial controller.

Figure 7-18 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 does not matter).

TypeCode	Meaning
000	No serial 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 serial controller decides whether to return IVR15-8 or IVR7-0 as the interrupt vector when it responds to an interrupt acknowledge cycle for which the highest priority requesting type is in the serial controller.

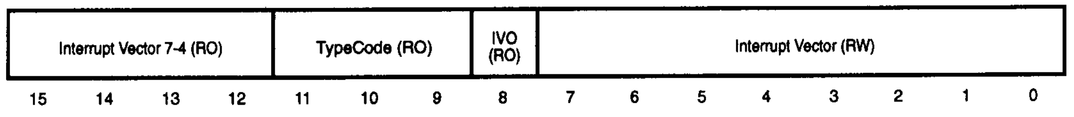


Figure 7-18. The Interrupt Vector Register (IVR)

7.15 DMA CONTROLLER INTERRUPT TYPES

There are only two interrupt types in the DMA Controller section of the IUSC, one each for the transmit and receive channels. Receive channel interrupts have higher priority than Transmit channel interrupts. Each DMA channel has one each IE, IP, and IUS bit, as described in an earlier section of this chapter. The interrupt capabilities of the two channels are identical and, except as noted, the information in the next five sections of this chapter applies equally to both.

7.16 DMA INTERRUPT SOURCES AND IA BITS

Software can set each DMA channel's IP bit in response to any of four possible interrupt sources, which are readable as the four LS bits of each DMA Mode Register (TDMR3-0 and RDMR3-0):

EOA/EOL. A DMA channel sets this bit (xDMR3) in Array and Linked List modes, when it goes inactive because it has fetched a zero Byte Count from an array or list entry, indicating the end of the array or list.

EOB. A DMA channel sets this bit (xDMR2) in any mode, when it decrements the Byte Count for the current buffer (TBCR or RBCR) to zero. It also sets this bit if software has enabled the Early Termination feature, when the serial controller signals for buffer termination. In Single Buffer mode the channel goes inactive at this time. The channel also goes inactive at this time in Pipelined mode, if the software hasn't provided a new buffer address and byte count and set the CONT bit (xDMR7).

HAbort. A channel sets this bit (xDMR1) in any mode, if external hardware drives the /ABORT pin low during a bus cycle by the channel. The channel goes inactive when this occurs, regardless of the mode.

SAbort. A channel sets this bit (xDMR0) in any mode, if host software writes an Abort This Channel or Abort All Channels command to the MSbyte of the DMA Command/Address Register (DCAR). The channel goes inactive when this occurs, regardless of the mode.

As noted in Chapter 5, the channel clears all four of these bits whenever software reads them in the LSbyte of its DMA Channel Mode Register (xDMR7-0).

Each of these four sources has a separate **Interrupt Arm (IA)** bit in each channel's DMA Interrupt Arm Register (TDIAR and RDIAR). Figure 7-19 shows the format of these registers. If an IA bit is 1, the interrupt logic sets the channel's IP bit when the corresponding status bit is 1. If an IA bit is 0, the corresponding status bit operates normally but has no effect on the channel's IP bit and thus cannot cause interrupts.

Reserved (0)												EOA/ EOL IA	EOB IA	HAbort IA	SAbort IA
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 7-19. The Transmit and Receive DMA Interrupt Arm Registers (TDIAR and RDIAR)

Reserved (0)						RxDMA IUS	TxDMA IUS	Reserved (0)						RxDMA IP	TxDMA IP
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 7-20. The Set and Clear DMA Interrupt Registers (SDIR and CDIR)

7.17 DMA IP AND IUS BITS

Software can read, set, and clear the **Interrupt Pending (IP)** and **Interrupt Under Service (IUS)** bits for both DMA channels using the shareable Set and Clear DMA Interrupt Registers (SDIR and CDIR). Figure 7-20 shows the arrangement of these registers. Software can read the current state of the bits from the SDIR at any time. Writing a one, to one or more of the four active bit positions in the SDIR, sets the corresponding bit(s), while writing a zero has no effect. Writing a one, to one or more of the four active bit positions in the CDIR, clears the corresponding bit(s), while writing a zero has no effect. The registers are defined like this to avoid interactions between hardware setting the IP and IUS bits and software clearing them.

In addition, one of the encoded commands that can be written to the DMA Command/Address Register (DCAR) allows for a general exit from a DMA interrupt service routine, regardless of whether it serviced the transmit or receive channel. If software writes the **Reset Highest DMA IUS** command (1000) to the DCmd field (DCAR15-12), the IUSC clears the highest-priority IUS bit that is set in the DMA section. Unfortunately, the command does not also clear the corresponding IP bit, so that an interrupt service routine has to do this explicitly for the particular channel that it is servicing.

7.18 DMA IE BITS

Software can read and write both channels' **Interrupt Enable (IE)** bits in the less significant byte of the shareable DMA Interrupt Control Register (DICR). Figure 7-21 shows the DICR. If a channel's IE bit is 1, then the IUSC requests an interrupt when its IP bit is 1 and its IUS bit is 0, provided that the channel's "IEI" signal from higher-priority types is true, and the DMA Controller's MIE bit (DICR15) is 1.

7.19 DMA-CONTROLLER-LEVEL INTERRUPT OPTIONS

Figure 7-21 also shows how the MSbyte of the DMA Interrupt Control Register (DICR) includes four control bits that affect all interrupts from the DMA section. These bits are fully under software control and can be read or written at any time.

The **Master Interrupt Enable** bit (MIE; DICR15) must be set to allow either of the DMA channels to request an interrupt.

Whenever the **Disable Lower Chain** bit (DLC; DICR14) is 1, the IUSC forces its IEO output low, so that devices further down the daisy chain can neither request interrupts nor respond to interrupt acknowledge cycles.

If the **No Vector** bit (NV; DICR13) is 1, the IUSC neither provides a vector nor drives the /WAIT//RDY pin, during an interrupt acknowledge cycle in which the highest-priority requesting type is one of the DMA channels. However, in such a case the IUSC still sets the IUS bit of the highest-priority requesting DMA channel.

The **Vector Includes Status** bit (VIS; DICR12) controls whether the vector returned, during an interrupt acknowledge cycle in which the highest-priority requesting type is one of the DMA channels, identifies the channel or not. If VIS is 0, the IUSC returns the vector programmed by the host software unchanged for both channels. If VIS is 1, bits 2-1 of the returned vector are 10 for a Tx channel interrupt and 11 for an Rx channel interrupt.

MIE	DLC	NV	VIS	Reserved (0)										RxDMA IE	TxDMA IE
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 7-21. The DMA Interrupt Control Register (DICR)

7.20 DMA INTERRUPT VECTORS

The vectors returned by the IUSC for interrupts from the DMA Controller are independent of those from the serial channel. Software can read and write interrupt vector information in the DMA Interrupt Vector Register (DIVR). This register is also the basis of the vector that the IUSC returns during an interrupt acknowledge cycle in which the highest-priority requesting type is one of the DMA channels.

Figure 7-22 shows the format of the DIVR. Software can read and write the basic vector in its LSbyte, and can read a modified version of the vector in its MSbyte. (Writing the MSbyte has no effect.) Bits 15-11 and 8 are the image of those in the corresponding bits of the LSbyte. DICR10-9 are a **TypeCode** for the highest priority DMA interrupt type that has its IP bit set (the state of its IUS bit does not matter):

TypeCode	Meaning
00	No DMA interrupt is pending
01	Reserved (will not be read)
10	Tx but not Rx interrupt
11	Rx interrupt

The state of the VIS bit (DICR12) has no effect on reading the DIVR. In fact, VIS simply determines whether the IUSC returns the MSbyte or LSbyte of the DIVR as the vector, during an interrupt acknowledge cycle in which the highest-priority requesting type is one of the DMA channels.

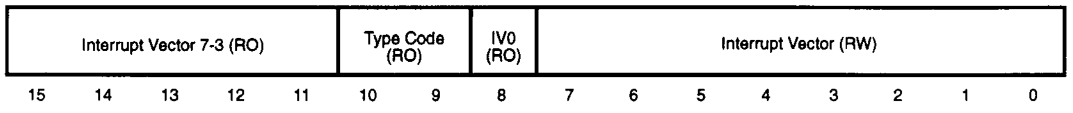


Figure 7-22. The DMA Interrupt Vector Register (DIVR)

7.21 SOFTWARE REQUIREMENTS

Having described all of the features and functions of the IUSC that relate to interrupts, this section will describe how these features should be used by interrupt service routines.

7.21.1 Nested Interrupts

An important characteristic of interrupt-driven systems is whether they allow nested interrupts, that is, whether they allow interrupt service routines (ISRs) to be themselves interrupted, or whether each ISR proceeds to completion before another interrupt can occur.

The IUSC supports nested interrupts by including an Interrupt Under Service (IUS) latch for each type of interrupt. When an IUSC that is requesting an interrupt sees an interrupt acknowledge cycle and its IEI pin is high, it automatically sets the IUS latch of the highest priority type that has its IP bit set. If interrupt acknowledge cycles are not visible to the IUSC, software can still allow nested interrupts by reading the IP bits from the LSbytes of the DCCR and SDIR, and explicitly setting the IUS latch of the highest priority type that has its IP bit set, in the MSbyte of the same register.

Regardless of whether the IUS bit is set automatically or explicitly by software, once it is set the ISR can re-enable processor interrupts to allow other interrupts. The IUSC in question will not request another interrupt for the same type nor any lower-priority type within it, until software clears the IUS bit near the end of the ISR.

Interrupts from other devices are controlled automatically if the devices are arranged in an interrupt daisy-chain; otherwise the central interrupt controller must control which devices can interrupt which ISRs.

When an ISR re-enables interrupts to allow nested interrupts from higher-priority types, it is a good practice to disable them once again, just before clearing the IUS bit near the end of the ISR. (They will be enabled again by the standard mechanism for the processor being used, e.g., an IRET or RETI instruction, after saved registers are restored from the stack.) This procedure prevents "tail recursion" when there's heavy interrupt traffic, wherein the stack gets filled with multiple copies of saved registers because another interrupt of the same type happens as soon as the IUS is cleared.

7.21.2 Which Type(s) to Handle?

If an interrupt service routine (ISR) is initiated by an interrupt acknowledge cycle that obtains a vector from the IUSC, and the "Vector Incudes Status" options of the serial and DMA sections are enabled, the service routine typically concerns itself only with the type identified by the vector, and returns from the interrupt after handling that single type.

Otherwise software should read the Interrupt Pending bits in the Daisy Chain Control Register (DCCR) and the Set DMA Interrupt Register (SDMR) to see which type(s) need service. This is particularly necessary on Personal Computers, in which interrupt acknowledge cycles are not visible to add-in peripherals.

If more than one IP bit is set in these registers, the ISR may handle only the most urgent type and return from the interrupt thereafter, like a "Vector Includes Status" ISR. Alternatively it may choose to handle all of the types that have their IP bits set, before returning to the interrupted process.

Without nested interrupts, worst-case interrupt response considerations may limit each ISR to handling just one type of interrupt before re-enabling interrupts and returning to the interrupted process. This allows the interrupt prioritizing mechanism to select which interrupt to handle next.

If nested interrupts can occur, it is more feasible for an IUSC ISR to handle all of the pending types within the device before returning to the interrupted process, because higher-priority ISRs will be able to run while it is doing so.

7.21.3 Handling a Type

The process of handling a single type of interrupt is the same regardless of whether the overall ISR handles only the highest priority pending type, or all the pending types within the device. The necessary steps vary for the various types in the IUSC.

The following descriptions don't attempt to cover everything that each type of ISR should do, only the minimum requirements needed to keep the interrupt subsystem operating correctly.

7.21.3.1 Receive Status or Transmit Status Type

1. Write the DCCR to clear the IP bit.

2. Read the RCSR or TCSR and handle the indicated conditions appropriately.

3. After all the conditions have been handled, write a byte to the LSbyte of the RCSR or TCSR, that has a 1 for each status bit that was handled and is armed by a 1 in the corresponding IA bit in the RICR or TICR. This clears/unlatches these status bits.

4. Write a zero byte to the LSbyte of the RICR or TICR, which disarms all the sources/status bits.

5. Write a byte to the same LSbyte, to re-arm those sources/status bits that should be armed for the future.

Steps 4 and 5 are needed only for these two types, to ensure that another interrupt will occur if the hardware sets armed sources/status bits after step 2, or the bits are otherwise left as 1 by the ISR.

7.21.3.2 I/O Pin or Miscellaneous Type

1. Write the DCCR to clear the IP bit.

2. Read the MISR and handle the indicated conditions appropriately.

3. After all the conditions have been handled, write a byte to the LSbyte of the MISR, that has a 1 in each "L/U" bit that was handled and is armed by a 1 in the corresponding IA bit in the SICR. This clears/unlatches these status bits. (Of course, software may want to write 1s to other L/U bits as well, such as those for unarmed conditions.)

7.21.3.3 Rx or Tx DMA Type

1. Write the CDIR to clear the IP bit.

2. Read the TDMR or RDMR. The four LSbits identify the interrupting condition(s). These bits are automatically cleared after they are read as 1, so software should take care to handle the End of Array/End of List condition as well as the End of Buffer condition, if it should read both as 1.

7

7.21 SOFTWARE REQUIREMENTS (Continued)

7.21.3.4 Receive Data Type

1. Write the DCCR to clear the IP bit.

2. Read the RDR often enough to bring the fill level below the "Rx Data Interrupt Request Level" in the RICR. Under some conditions, writing a Purge Rx FIFO command to the CCAR would eliminate the need to read the TDR.

Typically, the ISR wants to read the fill level from the RICR, and read the RDR the number of times indicated by that value. In HDLC and similar modes, because the "RD" interrupt occurs for the end of a frame as well as when the fill level reaches the Request Level, software can not blindly read the number of characters indicated by the Request Level.

On a 16-bit bus the minimum Request Level is 01 (meaning request when 2 characters have been received). In such a system it is acceptable for software to read only pairs of characters and leave the last (unpaired) character to be handled on the next interrupt. The exception is that in HDLC and similar modes, if the ISR gets a fill level of 01 from its first read of the RICR, the available character must be the last one of a frame, and as such should be read individually.

If the Request level is low and the serial rate is high, it might happen that enough characters arrive while software is reading the number indicated by the initial read from the RICR, so that the number of characters in the RxFIFO never falls below the Request Level. This is particularly possible if the Request Level is 01 (meaning interrupt when 2 empty slots) and software only reads character pairs from the RDR. If this can happen, after software finishes reading each block of data, it should read the RICR again, and read more data from the RDR if needed, to ensure that future Rx Data interrupts will occur.

In HDLC and similar modes, software will want to know where frames end. On a 16-bit bus, if the oldest character in the RxFIFO is the last one of a frame, and software tries to read 2 characters from the RDR, the (I)USC only removes the oldest character from the RxFIFO. The routine handling Receive Data interrupts can determine frame/message boundaries in two ways:

a. Read the RCSR after each read from the RDR. If the RxBound bit is set the previous read included the last character of a frame. In this case, if 1stBE in the RCSR is 1 then the last read yielded only 1 character, else it included 2 characters.

b. Enable nested interrupts and have the Rx Status ISR, when it sees an RxBound condition, do something to affect the operation of the RxData ISR when it resumes. This is tricky but is the sort of thing that can help make life as a programmer worthwhile.

7.21.3.5 Transmit Data Type

1. Write the DCCR to clear the IP bit.

2. Write the TDR often enough to bring the number of empty bytes in the TxFIFO below the "Tx Data Interrupt Request Level" in the TICR OR, write the (TCSR and) TICR with a smaller Request Level, to accomplish the same purpose. OR, write the ICR to disable the Transmit Data interrupt.

Typically the ISR wants to read the fill level from the TICR and write the TDR enough times to fill the TxFIFO, or write enough character pairs to fill it except for one empty position. If there is not enough data available to do this, the ISR might want to change the Request Level to 31 (hex 1F) so that the next Transmit Data interrupt will occur when the FIFO is empty, and then write all the available characters to the TDR.

If the Request level is low and the serial rate is high, it might happen that the Transmitter takes enough characters out of the TxFIFO while software is writing the number indicated by the initial read from the TICR, so that the number of empty slots never falls below the Request Level. This is particularly possible if the Request Level is 01 (meaning interrupt when 2 empty slots) and software only writes character pairs to the TDR. If this situation can happen, after software finishes writing a block of data to the TDR, it should read the TICR again and write more data to the TDR if needed, to ensure that future Tx Data interrupts will occur.

In HDLC and similar modes, the part of the ISR that handles Tx Data interrupts typically needs to take special actions at the end of each frame. It can do this with or without using the Transmit Character Counter (TCC), and can use the TCC either directly or by means of the 32-bit Transmit Control Block (TCB) feature.

Using the TCC directly:

a. At the start of each frame software should load the TCLR with the number of data characters in the frame/message, and then write a Load TCC command to the CCAR.

b. If, on a 16-bit bus after software has written enough characters to the TDR to decrement the TCC down to 0001, software writes 16 bits to the TDR, the (I)USC will only place the single character from the AD7-0 pins into the TxFIFO, ignoring the character on a AD15-8. In a Little-Endian (Intel-type) system this is acceptable. In a Big-Endian (Motorola-type) system software can avoid problems by either copying the last character of each Tx frame into the next-higher byte location after the memory buffer, or by writing the last byte of the frame using a byte write operation.

c. The hardware automatically tags the byte that corresponds to decrementing the TCC from 1 to 0. After this byte goes through the TxFIFO and out onto the link, the Transmitter finishes the frame, typically by sending the CRC and closing Flag.

e. Software can either read the TCC or use its own length-tracking mechanism, to know when each frame ends and thus when to write the TCLR again.

Using 32-bit TCBs:

a. Before the start of the first frame after a Reset, software has to write a Purge TxFIFO or a Load TCC command to the CCAR, to make the IUSC expect the first TCB. (For subsequent frames this step is not necessary.)

b. At the start of each frame software should write a 32-bit TCB to the TDR, of which the last 16 bits are the number of data characters in the frame.

c-f. Same as steps b-e above.

Not using the TCC:

a. Software doesn't need to do anything special to the IUSC at the start of a frame, other than to initialize its own frame-length-tracking mechanism.

b. While writing the frame to the TDR on a 16-bit bus, if there are two characters left in the frame, software must write the second-last character to the LSbyte of the TDR using a byte write operation.

c. Before writing the last character of the frame to the LSbyte of the TDR, software should write a 'Set EOF/EOM' command to the MSbyte of the TCSR.

d. After that byte goes through the TxFIFO and out onto the link, the Transmitter finishes the frame, typically by sending the CRC and closing Flag.

7.21.4 Exiting the ISR

If an IUS bit was set by an interrupt acknowledge cycle or explicitly by software, then after the ISR has handled one or more interrupt types as described above, it must clear the IUS bit that was set. (If nested interrupts were enabled, it is a good practice to first disable interrupts again, to avoid filling the stack with multiple copies of saved registers, in case another interrupt of the same type happens right after IUS is cleared. The normal mechanism provided by the processor for ending an ISR, e.g., an IRET or RETI instruction, will then re-enable interrupts after saved registers and such are restored from the stack.)

Software can clear IUS by writing to the MSbyte of the DCCR or CDIR, or by writing a "Reset Highest IUS" command to the MSbyte of the CCAR or DCAR. The latter method is more general than the former, but less than fully general in that the software has to remember whether the (highest priority) type was in the serial controller or the DMA section.

Software Summary 8

[illegible]

[illegible]

Zilog's General Terms and Conditions of Sale T

Zilog Sales Offices Representatives & Distributors Z

Zilog Literature Guide Ordering Information L

CHAPTER 8
SOFTWARE SUMMARY

8.1 INTRODUCTION

This chapter includes a bit by bit description of all the registers in the IUSC.

8.2 ABOUT RESETTING

The IUSC goes into 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 the IUSC 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 S//D pin (the one that selects between the Serial Controller and DMA sections of the IUSC in normal operation) determines whether the IUSC drives and receives the /WAIT//RDY pin as a "wait" or "acknowledge" handshake.

Aside from requiring the BCR write, software can reset the IUSC much like a hardware reset does. *Resetting the Serial Controller* 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. Software can also reset the DMA channels, by writing the "Reset All Channels" command (hex 90) to the MSByte of the DMA Command/Address Register (DCAR15-8).

After either a hardware or a software reset, all register bits in the IUSC are zero except for the following:

1. The following bits reflect the state of pins. The IUSC treats these as inputs until and unless software programs them as outputs.

MISR14	/RxC
MISR12	/TxC
MISR10	/RxREQ
MISR8	/TxREQ
MISR6	/DCD
MISR4	/CTS
PSR14	/PORT7
PSR12	/PORT6
PSR10	/PORT5
PSR8	/PORT4
PSR6	/PORT3
PSR4	/PORT2
PSR2	/PORT1
PSR1	/PORT0

2. The following bits are 1 because the TxFIFO is empty:

TCSR0	TxEmpty
TICR13	(indicates 32 empty entries)

8.3 PROGRAMMING ORDER

The IUSC 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 IUSC's internal logic and make it do strange things. Always initialize the IUSC in the following order:

1. Set the pin configurations in the IOCR and PCR. 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) bits in the serial controller and DMA sections. In general, you want to do this last so that interrupt service routines can assume that everything's set up in its starting configuration.

8.4 USING DMA TO INITIALIZE THE SERIAL CONTROLLER

Instead of initializing the serial controller and DMA channels together as described above, software can initialize the IUSC's Transmit DMA channel first and then use it to initialize the serial controller. To do this:

1. Initialize the shared DMA registers DCR and BDCR to match the system hardware and software configuration. There shouldn't be any need to use interrupts for this operation, but it might be a good idea to set up the DICR and DIVR as well.
2. Program the MSByte of the TDMR appropriately for the initializing transfer. Single Buffer mode should suffice.
3. Program the TAR with the address of a sequence of bytes or 16-bit words that will initialize the serial controller. 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.
4. Arrange the string/sequence to initialize the serial controller 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 IUSC will 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 IUSC will stop without fetching a following byte or word.
5. Program the TDMR 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.
6. Write a "Start Tx Channel" command including MBRE=1 (hex 21) to the MSByte of the DMA Command /Address Register (DCAR).
7. Write a "Trigger Channel Load DMA" command (hex 20) to the MSByte of the CCAR.
8. Assuming the processor is set up to grant use of the bus to the IUSC, the operation should complete very quickly. This should be verified by checking the LSByte of the TDMR for hex 04 (End of Block).

8.5 DETERMINING THE DEVICE REVISION LEVEL

8.5.1 Fetching First TCB

New designs should not have to bother with this distinction, but long-standing applications may have to.

Software can determine whether an IUSC was manufactured before or after September of 1992, in terms of whether it fetches a TCB from the first entry of an Array or Linked List, as follows:

1. Enable the Transmitter but disable its output on TxD.
2. Enable 32-bit TCBs.
3. Set up a list or array in which the TCB in the first entry contains one TCC value while the first data buffer starts with a TCB that includes a different value.
4. Issue a Load TCC or Purge Tx FIFO command to make the device expect a TCB.
5. Start/Init the Tx DMA channel on the array or list,
6. Read the TCLR to see which value it has, and save this result in a boolean variable for this IUSC.
7. Re-initialize the hardware by pausing the DMA channel, purging the TxFIFO, and re-enabling the Transmitter output onto TxD.

When it comes time to start the DMA channel in normal operation, software should examine the boolean variable set up in step 6 above to determine how to do so:

a. Write a Load TCC or Purge Tx FIFO command to the device, to ensure that it is expecting a TCB.

b. If the device is an old one that does not fetch the TCB from the first array or linked list entry, software should fetch these two "TCB words" from the first entry itself, and write them to the TDR.

c. Start/Init the Tx DMA channel.

8.5.2 Determining Later Revisions

Zilog makes every effort to improve devices like the 16C32 while preserving compatibility with software developed on earlier devices. Nonetheless, for some purposes (like using new features) software needs to tell which revision of the device it is operating on, and behave differently for different revisions.

The Test Mode Control Register (TMCR) is register number 00111 (typically address 0E), and the Test Mode Data Register (TMDR) is register 00110 (typically address 0C). If software writes the value 31 (hex 1F) to the TMCR and then reads the TMDR, IUSCs that do not bear the SL1660 topmarking will return the hex value 4453, while those manufactured after that time will return 4D44. If there are future functional revisions to the device, they will return some other value.

On an 8-bit bus, software can only read the LS byte of the TMDR (hex 53 or 40).

Software can use this feature to determine whether it can use new features of later devices, such as the Purge Rx command (described in the Commands section of Chapter 5, the UnderWait feature (described in the 'Handling Overruns and Underruns' section of Chapter 5, and the array-chaining feature (described in the *Array Mode* section of Chapter 6).

8.6 TIPS AND TECHNIQUES

This section describes some of the commonest ways that people have gotten in trouble using the IUSC, in hardware and software.

8.6.1 Common Hardware Problems

H1. /DS OR (/RD and /WR), not both
Interconnect /RD and /WR among multiple IUSCs, or interconnect /DS, but not all three. Each non-bused pin should be connected to an individual pullup resistor of about 10K ohms.

H2. No other strobe during pulsed IACK cycles
If your application uses "single pulsed" or "double pulsed" interrupt acknowledge signaling, you have to ensure that neither /RD nor /DS is asserted during the acknowledge cycle.

H3. More pullups!
IUSC designs need a lot of pullup resistors, for various reasons:

- Unused inputs or I/Os: PORT0-7, /IEI, /INTACK, /ABORT
- Outputs tri-stated until IUSC initialized: PORT0-7, /BUSREQ, /INT
- Bus control signals that aren't always driven by external logic when the IUSC(s) aren't doing so: /AS, R//W, /DS, /RD, /WR.
- Serial inputs that aren't driven by external logic in some cases: /TxREQ, /RxREQ, /TxC, /RxC, /CTS, /DCD, PORT7-0.

H4. /WAIT//RDY neither open-drain nor rescinded
/WAIT//RDY is a totem-pole output except when the IUSC is acting as bus master. This can be a time-critical signal, and RC rise times are not good in critical applications. The /WAIT//RDY outputs of multiple IUSCs have to be ORed (positive-logic ANDed) using a logic gate.

H5. Drive /AS whenever /RD, /WR, or /DS
Designs that synthesize an /AS pulse to multiplex a non-multiplexed bus (so that software does not have to write register addresses to indirect address registers) need to pulse /AS low in all cycles that include a pulse on /RD, /WR, or /DS. Several designers, including the writer, have gotten into trouble trying to save power and noise by only driving /AS low in host cycles targeted for the IUSC. It is acceptable to do this if /RD and /WR or /DS are similarly qualified, so that they occur only during cycles targeted to the IUSC. But if the logic blocks the /AS and then shows the part one of the other strobes, it figures it is still "chip selected" (after all, the last /AS showed /CS low) and responds to the cycle that is actually intended for a different slave.

8.6.2 Common Software Problems

S0. "Unreset"
The software Reset facility in the CCAR has to be set and then cleared. The part will not operate correctly if it is left as 1.

S1. Register Initialization Order
There are certain constraints on the order in which the various registers in the IUSC should be initialized, as described earlier in this Chapter. Many of them are common-sense points, but the some seemingly obvious approaches, like initializing the registers in address order or alphabetical order, are not likely to be successful.

S2. WordStatus problems
Always program the WordStatus bit (RICR3) according to whether software or the Rx DMA channel will subsequently read 8 or 16 bits from the RDR and RxFIFO. If your software writes the LSbyte of RICR to change the Rx Status IA bits, be sure it preserves the proper setting of WordStatus while doing so.

S3. Handling MBRE
Be sure to accompany DMA commands written to the DCAR with the proper setting of the Master Bus Request Enable bit (MBRE). As discussed in *Commands and /BUSREQ Enable* in Chapter 6, the proper setting is obvious for commands that Start one or both channels or Pause, Abort, or Reset both. For commands that Pause, Abort, or Reset one channel, it is probably best to include MBRE=1, in case the other channel is running.

For applications on a 16-bit bus that must write indirect register addresses to the DCAR, IUSCs that do not bear the 16C32 SL1660 topmarking require software to clear MBRE when doing so, then read or write the target register, and then set MBRE again. IUSCs with the SL1660 topmarking do not have this problem. If software knows it is running with such a device, it can omit this procedure, but must preserve the current setting of MBRE when writing register addresses.

S4. Transmit Data Length
Note that there are two controls required on the length of transmitted data. The TBCR in the Tx DMA channel, which is set from the fifth and sixth bytes of each entry in Array and Linked List modes, controls how many bytes the DMA channel takes from each buffer. The TBCR value must include any Transmit Control Blocks that are provided in DMA buffers, but not TCBs that are in the array and linked list entries.

The TCLR in the Transmitter, which is typically set from the second word of the TCB if TCBs are used, controls how many bytes the Transmitter sends in each frame, and should not include CRC bytes that the Transmitter calculates and sends, but should include CRC bytes that are "passed through" from a received frame without change.

S5. Receive Data Length
There is one required control, one optional control, and one reporting mechanism associated with the length of received data. The RBCR in the Rx DMA channel, which is set from the fifth and sixth bytes of each entry in Array and Linked List modes, controls the maximum number of bytes the Rx DMA channel will store in each buffer. The length of Rx memory buffers, and thus the RBCR values, should allow for storing CRCs if they are used, and also allow for 16-bit RSBs if they are stored in the buffer but need not allow for 32-bit RSBs if they are stored in Array and Linked List entries.

The optional control is the value of the RCLR, which can be set to the length of the longest frame that can be legally received, including the CRC. An optional interrupt when the RCC underflows can be enabled to notify software of an unduly long frame, which generally indicates the corruption of the Flag(s) between two frames.

The reporting mechanism is that the ending value of the RCC for each frame, is typically stored in 32-bit RSBs in Array and Linked List entries. The length of the frame, including CRC bytes, can be computed by subtracting this ending value from the starting value of the RCLR. If RCLR is set to all ones, the frame length is simply the ones complement of the ending value.

S6. FIFO Thresholds
The Tx and Rx DMA thresholds must be set to at least 1 on a 16-bit bus, meaning "request DMA transfer when at least two characters have been received or when there are at least two empty character locations in the TxFIFO." Many applications operate best if the DMA thresholds are set at about half full. Lower values provide greater protection against Rx Overruns and Tx Underruns, but reach a point of diminishing returns due to increasing overhead of getting on and off the external bus.

It is a good programming practice to protect the DMA thresholds from inadvertent destruction by word writes to the TICR and RICR, by writing "Select FIFO Status" commands to the TCSR and RCSR after the thresholds are set.

Assuming you do not want Tx nor Rx Data interrupts in IUSC applications, there is no need to program the Interrupt thresholds. Just leave the IE bits for these interrupts 0.

S7. Interrupts for Rx Overrun and Tx Underrun
These are the two things the IUSC needs software to deal with fairly immediately. Software should virtually always enable interrupts for these two conditions. They are preferable to related interrupts like Abort Sent because they occur earlier and give the software more time to deal with the conditions.

8.6 TIPS AND TECHNIQUES (Continued)

S8. Interrupt handling
A new section at the end of Chapter 7 gives specific requirements for each type of interrupt. In general, the strategy is 1) clear IP, 2) read the status bits, handle them including clearing/unlatching them, and 3) clear IUS. Interrupts can be lost if this order is not followed. The efficient-sounding command "clear IP and IUS" that the IUSC offers, should be avoided except in initialization sequences.

S9. Clearing all IA bits for Rx and Tx Status interrupts
For these two types of interrupts, software has to clear all of the IA bits after it reads and clears the status bits, and then set the desired IAs again, to ensure that any status conditions that have arisen since software last read the status, will cause a subsequent interrupt request.

S10. Do not use ReArbTime "one channel per grant"
The value 10 in the ReArbTime field (DCR11-10) leads to various kinds of erroneous operation, and is now Reserved and should not be programmed.

S11. Priming the Transmitter for Transmit Control Blocks
When using TCBs, after a hardware or software Reset, software must force the Transmitter to expect the initial TCB by issuing a Purge TxFIFO or Load TCC command. In Array or Linked List mode, after the Tx DMA channel has come to the end of the Array or List as indicated by a zero buffer byte count, software must similarly force the Transmitter to expect a TCB by issuing a Purge TxFIFO or Load TCC command, before it issues the Start/Init Tx channel command.

S12. Interlocks are "after end of frame" not "before start"
The three classes of interlocks for software synchronization between frames, Wait2Send/Send Frame, Wait4TxTrig/Trigger Tx DMA, and Wait4RxTrig/Trigger Rx DMA, all occur after the end of a frame, not before the first frame after the part is set up. Thus these three commands are not needed right after initialization.

S12. Preserving loopback/echo settings
If you want to set a loopback or echo mode in CCAR9-8, be sure to preserve it when writing commands to the MSbyte of CCAR. If your processor has an "OR to memory" command and the IUSC is memory-mapped, that instruction is a natural for issuing commands. Similarly, if your application is on a 16-bit bus and must write indirect register addresses to the CCAR, be sure to preserve CCAR9-8 when doing so.

8.7 SERIAL CONTROLLER TEST MODES

The serial controller portion of the IUSC includes a facility intended for Zilog's device testing, that gives software access to certain internal signals and registers that are not otherwise accessible. The low-order bits of the Test Mode Control Register (TMCR) serve to select which internal signals or registers are accessed by reading (or in two cases writing) the Test Mode Data Register (TMDR). The choices are as follows:

TMCR4-0	Signals/Register Selection	R/W Status
00001	TMDR15-8: Rx shift register, MSbyte	RO
	TMDR 7-0: Tx shift register, MSbyte	RO
00010	TMDR15-8: Rx CRC checker, LSbyte	RO
	TMDR 7-0: Tx CRC generator, LSbyte	RO
00011	TMDR15-8: Rx CRC checker, bits 15-8	RO
	TMDR 7-0: Tx CRC generator, bits 15-8	RO
00100	serial side of RxFIFO:	WO
	TMDR11: ShortF/CVType status bit	WO
	TMDR10: Abort/PE status bit	WO
	TMDR9: RxBound status bit	WO
	TMDR8: CRC Error status bit	WO
	TMDR7-0: data character	WO
00101	Clock MUX outputs (see Figure 8-1 below)	RO
00110	TMDR12-8: CTR1 value	RO
	TMDR4-0: CTR0 value	RO
00111	Clock MUX inputs (see Figure 8-2 below)	RO
01000	TMDR15: DPLL Adjust	RO
	TMDR14: DPLL Shorten/Extend	RO
	TMDR13: DPLL One/Two	RO
	TMDR12-8: DPLL State	RO
01001	TMDR15-8: Rx shift register, LSbyte	RO
	TMDR 7-0: Tx shift register, LSbyte	RO

TMCR4-0	Signals/Register Selection	R/W Status
01010	TMDR15-8: Rx CRC checker, bits 23-16	RO
	TMDR 7-0: Tx CRC generator, bits 23-16	RO
01011	TMDR15-8: Rx CRC checker, bits 31-24	RO
	TMDR 7-0: Tx CRC generator, bits 31-24	RO
01100	serial side of TxFIFO:	RO
	TMDR10: EOF/EOM bit	RO
	TMDR9: CRC enable bit	RO
	Transparent Bisync: insert DLE	RO
	Nine-Bit mode: Address/Data	RO
	TMDR7-0: data character	RO
01110	I/O and Misc status (see Figure 8-3 below)	WO
01111	internal interrupt daisy chain:	RO
	TMDR13: Rx Status IEO	RO
	TMDR12: Rx Data IEO	RO
	TMDR11: Tx Status IEO	RO
	TMDR10: Tx Data IEO	RO
	TMDR9: I/O Pin IEO	RO
	TMDR8: Misc IEO	RO
10110	Receive Count Holding Register (RCHR)	RO
11111	Device Version Code:	RO
4453	Device manufactured without SL1660 topmarking	RO
4D44	Device manufactured with SL1660 topmarking	RO

Some of this information may be of use to software. However, the hardware access time for reading the TMDR is considerably longer than for other IUSC registers. (See the Product Description for details.)

If software needs to read any of the above Test Mode information, the hardware design must provide more time for the data lines to become valid, than would otherwise be necessary. This may require the injection of more "wait states" into such read cycles than would be needed for other registers. In some cases the best solution is a software-programmable wait state generator that can extend accesses to TMDR but not penalize performance for other IUSC register accesses.

8.7 SERIAL CONTROLLER TEST MODES (Continued)

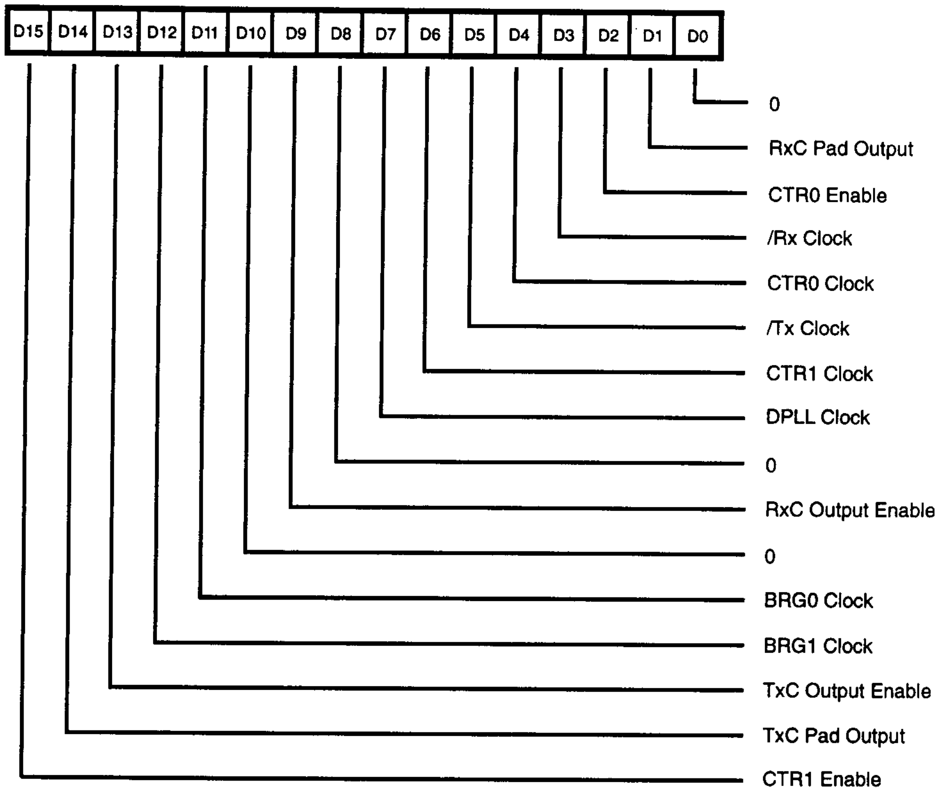


Figure 8-1. Test Mode Data Register with TMCR 4-0 = 00101 (Clock Mux Outputs)

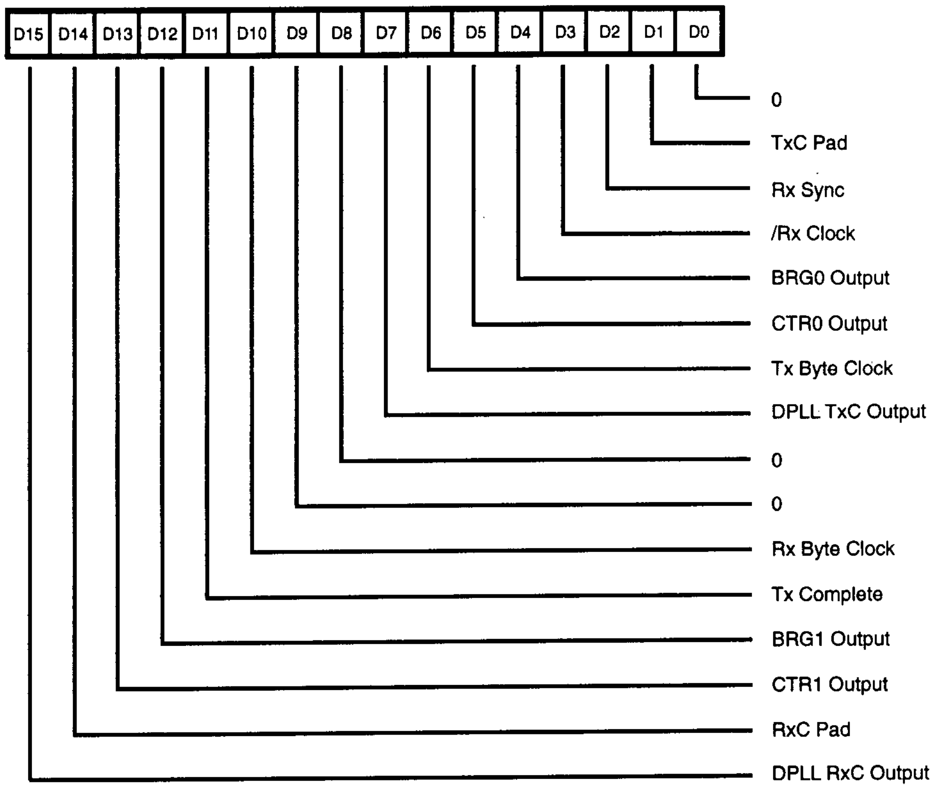


Figure 8-2. Test Mode Data Register with TMCR 4-0 = 00111 (Clock Mux Inputs)

8.7 SERIAL CONTROLLER TEST MODES (Continued)

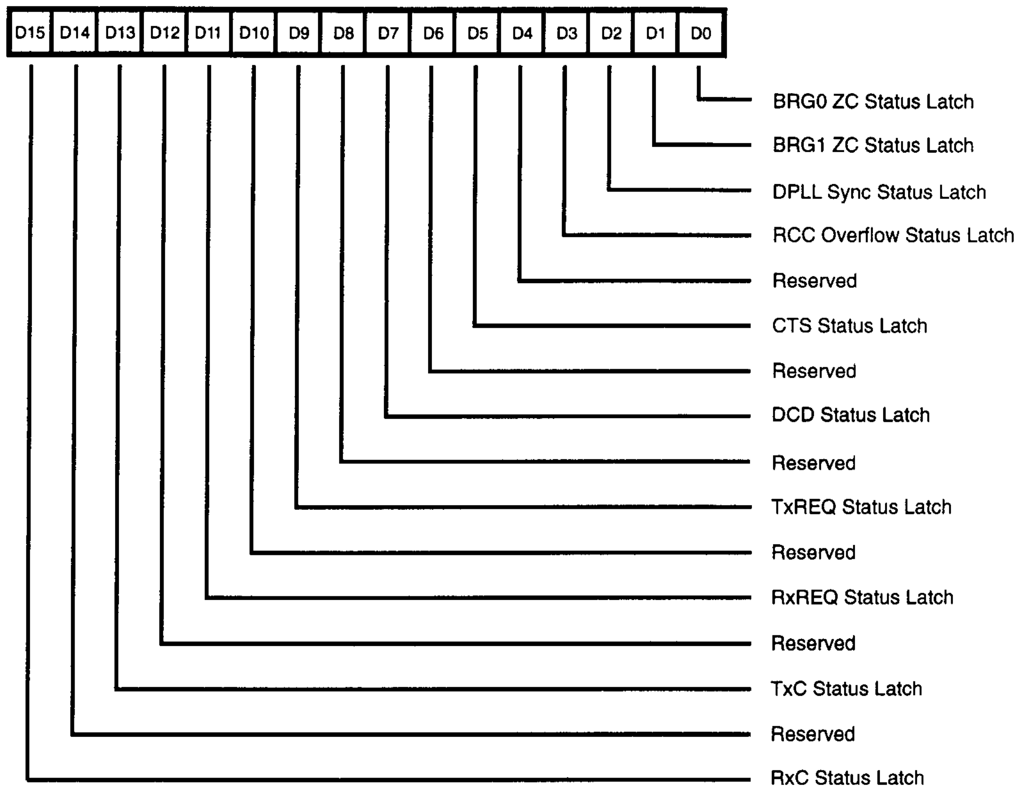


Figure 8-3. Test Mode Data Register with TMCR 4-0 = 01110 (I/O and Misc. Status)

8.8 REGISTER REFERENCE

8.8.1 Register Addresses

The following pages include all of the fields in all of the registers in the IUSC, including both the serial controller and DMA sections. They are arranged in alphabetical order by register name, like Table 2-2 in Chapter 2. (If you want to look up a register by its address/register number, look in Table 2-1 in Chapter 2 and then come back here...) These are located to the right of the name of each register on the following pages, and are shown as s d b aaaaa, where:

s is the address bit connected to the S//D pin (0=DMA, 1=serial);

d is the address bit connected to the D//C pin, or the bit in DCAR7 (0=serial control regs or DMA Tx, 1=serial Data regs or DMA Rx);

b is 1 for a byte access on a 16-bit bus (it is shown as "b" in all cases, like a placeholder);

aaaaa is the actual register address, from AD5-1, AD13-9, CCAR5-1, or DCAR5-1.

8.8.2 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.

8.8.3 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.

8.8.4 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 IUSC 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 is described next in the list.
WOC	The field is write only. After using its value the IUSC will clear it to zero, so that it points back to the indirect address register.
R,W1C	The bit is set by the IUSC hardware, writing a 1 to it clears it.
R,W1U	The bit is controlled by the IUSC hardware, writing a 1 to it "unlatches" it.

8.9 REGISTER TABLES

Burst/Dwell Control Register (BDCR) **Register Address 0 x b 01001**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MaxXfers								MaxCLKs							

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
BDCR15-8	MaxXfers		0=no effect; 1-255=maximum number of bus cycles/transfers the DMA channels will do per bus grant	RW	6: Bus Occupancy Throttling
BDCR7-0	MaxCLKs		0=no effect; 1-255=DMA channels limited to 8-2040 CLK periods per bus grant		

Bus Configuration Register (BCR) **No Address (First Write after /RESET)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SepAd	Reserved (Must be zero)									IAckMode		BRQTP	16Bit	/IRQTP	SRight A

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
		16-bit bus	Must be 0		
BCR5-4	IAckMode		00=sample /INTACK at start of each slave cycle; 01=single pulse on /INTACK; 11=double pulse on /INTACK		
BCR3	BRQTP		0=drive /BUSREQ open-drain, sample it first; 1=drive /BUSREQ totem pole (full time)		
BCR2	16-Bit		0=8-bit data on AD7-0; 1=16-bit data on AD15-0		
BCR1	/IRQTP		0=drive /INT pin totem pole (full time); 1=drive /INT open drain		
BCR0	SRightA	Muxed AD	0=use AD6-0 as B/W, RegAddr, U/L; 0=use AD7-1		

Channel Command/Address Register (CCAR) **Register Address 1 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 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 Data LSBit First 10101=Select Serial Data MSBit First 10110=Select D15-8 First 10111=Select D7-0 First 11001=Purge Rx All other values are Reserved and should not be programmed.	WO	5: Commands
CCAR10	RTReset		1=put Serial Controller in software Reset state; 0=release it from Reset state	RW	5: Resetting the Serial Controller

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	4: The RxD and TxD Pins

CCAR7	ChanLoad	Channel Load DMA	1=continue Channel Load operation; 0=terminate it	RW	8: Using DMA to Initialize the Serial Controller
CCAR6	B//W	16-bit bus	0=next access to CCAR will be 16 bits; 1=access MS or LS byte of register	WOC	2: Register Addressing
CCAR5-1	RegAddr		register address for next access to CCAR (see Tables 2-1 and 2-2)	WOC	
CCAR0	U//L		1=next access to CCAR will be to MSbyte of selected by RegAddr; 0=access LSByte or whole 16-bit register	WOC	

Channel Command/Status Register (CCSR) **Register Address 1 0 b 00010**

RCCF Ovflo	RFFC Avail	Clear RCCF	DPLL Sync	DPLL 2Miss	DPLL 1Miss	DPLLEdge		On Loop	Loop Send	Ctr Bypass	TxResidue			Reserved	
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	5: DMA Support Features: The RCC FIFO
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	4: More About the DPLL
CCSR11	DPLL2Miss	Biphase	1=DPLL has seen two consecutive missing clocks	R,W1C	
CCSR10	DPLL1Miss	Biphase, CVOK=0	1=DPLL has seen a missing clock	R,W1C	
CCSR9-8	DPLLEdge		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	5: Slaved Monosync Mode 5: HDLC/SDLC Loop Mode
		H/SDLC Loop	1=IUSC has inserted itself in the loop		
CCSR6	LoopSend	H/SDLC Loop	1=Transmit actively sending; 0=Transmit repeating Receive	RO	5: HDLC/SDLC Loop Mode
CCSR5	CtrBypass		0=route CTR1-0 outputs to Rx/TxCLK selection, BRG's, /RxC, /TxC output selection 1=route PORT1-0 pins direct to these uses	RW	4: Transmit and Receive Clocking: Using PORT0-1 for Bit Clocking
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	5: HDLC/SDLC Mode: Frame Length Residuals

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Channel Control Register (CCR) **Register Address 1 0 b 00011**

TxCtrBlk		Wait4 Tx Trig	Flag Pre-amble	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	TxCtrBlk		00=do not use Transmit Control Blocks; 10=use 32-bit TCB's	RW	5: DMA Support Features: Transmit Control Blocks
CCR13	Wait4TxTrig	Sync	1=hold Transmit DMA Request between frames/messages, until software issues "Trigger Tx DMA" command		5: Synchronizing Frames/ Messages with Software Response
CCR12	Flag Preamble	H/SDLC, CCR9-8 =01	1=send Flags as Preamble		5: Between Frames, Messages, or Characters
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		5: Asynchronous Mode
CCR11-10	TxPreL	Sync w/ Preamble	00=send 8-bit Preamble; 01=16-bit; 10=32-bit; 11=64-bit		5: Between Frames, Messages, or Characters
CCR9-8	TxPrePat	Sync w/ Preamble	00=all-zero Preamble; 01=all ones or Flags; 10=101010...; 11=010101...		
CCR7-6	RxStatBlk		00=do not use Receive Status Blocks;		5: DMA Support Features: Receive Status Blocks
		Ext Sync, T. Bisync, H/SDLC, 802.3, ACV (1553B)	00=use 16-bit RSB's; 10=use 32-bit RSB's		
CCR5	Wait4 RxTrig	Sync	1=hold Receive DMA Request between frames/ messages, until software issues "Trigger Rx DMA" command		5: Synchronizing Frames/ Messages with Software Response

Channel Mode Register (CMR) **Register Address 1 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 zeros.

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
CMR11-8	TxMode		**0000=Asynchronous**	RW	5: Asynchronous Mode
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**		5: External Sync Mode
CMR11-8	TxMode		**0010=2=Isochronous**	RW	5: Isochronous Mode
CMR14	TxSubMode	TxMode=2	0=send one stop bit; 1=two stop bits	RW	
CMR3-0	RxMode		**0010=2=Isochronous**	RW	
CMR11-8	TxMode		**0100=4=Monosync**	RW	5: Monosync and Bisync Modes
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	0=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 zeros.

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
CMR11-8	TxMode		**0101=5=Bisync**	RW	5: Monosync and Bisync Modes
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		
CMR11-8	TxMode		**0110=6=HDLC/SDLC**	RW	5: HDLC/SDLC Mode
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		
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	5: Transparent Bisync Mode
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	5: Nine Bit Mode
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	

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

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 zeros.

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
CMR11-8	TxMode		**1001=9=802.3 (Ethernet)**	RW	5: 802.3 (Ethernet) Mode
CMR15	TxSubMode	TxMode=9	0=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	5: Slaved Monosync Mode
CMR15	TxSubMode	TxMode =12	1=send CRC on Tx Underrun	RW	
CMR13			0=do not 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)		
CMR11-8	TxMode		**1101=13=Reserved**		
CMR3-0	RxMode				
CMR11-8	TxMode		**1110=14=HDLC/SDLC Loop**	RW	5: HDLC/SDLC Loop Mode
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				

Clear DMA Interrupt Register (CDIR)

Register Address 0 x b 01101

Reserved (0)						RxDMA IUS	TxDMA IUS	Reserved (0)						RxDMA IP	TxDMA 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
CDIR9	RxDMA IUS		1=clear Rx DMA IUS bit; 0=no change	WO	7: DMA IP and IUS Bits
CDIR8	TxDMA IUS		1=clear Tx DMA IUS bit; 0=no change		
CDIR1	RxDMA IP		1=clear Rx DMA IP bit; 0=no change		
CDIR0	TxDMA IP		1=clear Tx DMA IP bit; 0=no change		

Clock Mode Control Register (CMCR) Register Address 1 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	4: Tx and Rx Clocking: CTR0 and CTR1
CMCR13-12	CTR0Src		00=CTR0 disabled; 01=CTR0 input is PORT0/CLK0; 10=/RxC pin; 11=/TxC pin		
CMCR11-10	BRG1Src		00=BRG1 input is CTR0 output or PORT0; 01=CTR1 output or PORT1; 10=/RxC pin; 11=/TxC pin		4: Tx and Rx Clocking: The Baud Rate Generators
CMCR9-8	BRG0Src		00=BRG0 input is CTR0 output or PORT0; 01=CTR1 output or PORT1; 10=/RxC pin; 11=/TxC pin		
CMCR7-6	DPLLSrc		00=DPLL input is BRG0 output; 01=BRG1 output; 10=/RxC pin; 11=/TxC pin		4: Tx and Rx Clocking: Introduction to the DPLL
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 or PORT0; 111=TxCLK is CTR1 output or PORT1		4: Tx and Rx Clocking: TxCLK and RxCLK Selection
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 or PORT0; 111=RxCLK is CTR1 output or PORT1		

Daisy Chain Control Register (DCCR)

Register Address 1 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	7: Serial IP and IUS Bits

DCCR13	RS IUS	Read	1=Receive Status interrupt under service	RO	7: Serial IP and IUS Bits 7: Receive Status Interrupt Sources and IA Bits
		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	7: Serial IP and IUS Bits 7: Rx Data Interrupts
		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	7: Serial IP and IUS Bits 7: Tx Data Status Interrupt Sources and IA Bits
		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	7: Serial IP and IUS Bits 7: Transmit Data Interrupts
		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	7: Serial IP and IUS Bits 7: I/O Pin Interrupt Sources and IA Bits
		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	7: Serial IP and IUS Bits 7: Miscellaneous Interrupt Sources and IA Bits
		Write	1=set or clear Miscellaneous 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	7: Serial IP and IUS Bits

DCCR5	RS IP	Read	1=Receive Status interrupt pending	RO	7: Serial IP and IUS Bits 7: Rx Status Interrupt Sources and IA Bits
		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	7: Serial IP and IUS Bits 7: Rx Data Interrupts
		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	7: Serial IP and IUS Bits 7: Tx Status Interrupt Sources and IA Bits
		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	7: Serial IP and IUS Bits 7: Transmit Data Interrupts
		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	7: Serial IP and IUS Bits 7: I/O Pin Interrupt Sources and IA Bits
		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	7: Serial IP and IUS Bits 7: Miscellaneous Interrupt Sources and IA Bits
		Write	1=set or clear Miscellaneous IP/IUS per IP Op; 0=no change	WO	

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

DMA Array Count Register (DACR) **Register Address 0 x b 00100**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved (0)								RALCnt				TALCnt			

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
DACR7-4	RALCnt	Array or Linked List	Reflects the Rx DMA channel's progress while fetching array or list information. **See Ref text.**	RO	6: Array and Linked-List Fetching Status
DACR3-0	TALCnt		Reflects the Tx DMA channel's progress while fetching array or list information. **See Ref text.**		

DMA Command/Address Register (DCAR) **Register Address 0 x b 00000**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DCmd				Reserved (0)		Rx/Tx Cmd	MBRE	Rx/Tx Reg	B/W	RegAddr					U/L

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section

DCAR15-12	DCmd		Treating all of DCAR15-9 as a single field:
			0000000=Null (no operation);
			0001000=Reset Tx Channel;
			0001001=Reset Rx Channel;
			0010000=Start Tx Channel;
			0010001=Start Rx Channel;
			0011000=Start/Continue Tx Channel;
			0011001=Start/Continue Rx Channel;
			0100000=Pause Tx Channel;
			0100001=Pause Rx Channel;
			0101000=Abort Tx Channel;
			0101001=Abort Rx Channel;
			0111000=Start/Init Tx Channel;
			0111001=Start/Init Rx Channel;
			1000000=Reset Highest DMA IUS;
			1001000=Reset All Channels;
			1010000=Start All Channels;
			1011000=Start/Continue All Channels;
			1100000=Pause All Channels;
			1101000=Abort All Channels;
			1111000=Start/Init All Channels
				WO	6: Commands and /BUSREQ Enable

DCAR9	Rx/Tx Cmd				
DCAR8	MBRE		1=enable Bus Requests by the DMA channels; 0=block Bus Requests by the DMA channels	RW	
DCAR7	Rx/Tx Reg		1=select Rx DMA channel register for next access to DCAR; 0=Tx DMA register	WOC	2: Register Addressing
DCAR6	B/W	16-bit bus 16-bit bus	0=next access to DCAR will be 16 bits; 1=access MS or LS byte of register	WOC	
DCAR5-1	RegAddr		DMA register address for next access to DCAR	WOC	
DCAR0	U/L		1=next access to DCAR will be to MSByte of register selected by RegAddr; 0=LSByte or whole 16-bits	WOC	

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

DMA Control Register (DCR) **Register Address 0 x b 00011**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ChanPri		Pre Empt	ALBVO	ReArbTime		Reserved (0)		Reserved (0)		Min Off39	DCSD Out	1Wait	UAS All	AddrSeg	

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
DCR15-14	ChanPri		00=Tx DMA has priority for bus access; 01=Rx DMA has priority; 10=alternating priority	RW	6: Inter-Channel Operation and Priority
DCR13	PreEmpt		1=higher-priority channel can seize bus control		
DCR12	ALBVO	Array and Linked List	0=addresses/counts are Little-Endian (Z80®/Intel) 1=Big-Endian (Z8000®/680x0)		6: Format of Binary Values in Array/Lists
DCR11-10	ReArbTime		00=select channel at start of each grant, both channels can use the bus in one grant; 01=channel keeps selection until its request is gone; then the other channel can use the bus in the same grant; 1x=Reserved, do not program		6: Inter-Channel Operation and Priority
DCR5	MinOff39		1=minimum bus re-request time is 39 CLKs; 0=7 CLKs		6: Bus Occupancy Throttling
DCR4	DCSDOut		1=drive D//C pin low for Tx DMA, high for Rx DMA and drive S//D low for array/list access, high for data; 0=do not drive D//C, S//D pins		6: DMA Cycle Options 6: Bus Cycle Options
DCR3	1Wait		1=add one Wait state to all DMA cycles		
DCR2	UASAll		1=present /UAS and MS16 of address in every cycle; 0=only when necessary		
DCR1-0	AddrSeg		00=32-bit address incrementing/decrementing; 10=incr/decr affects only LS 16 address bits; 11=incr/decr affects only LS 24 address bits		6: Address Sequencing

DMA Interrupt Control Register (DICR) **Register Address 0 x b 01100**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MIE	DLC	NV	VIS	Reserved (0)										RxDMA IE	TxDMA IE

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
DICR15	MIE		1=enable interrupts from DMA channels	RW	7: DMA-Controller-Level Interrupt Options
DICR14	DLC		1=disable IEO from IUSC		
DICR13	NV		1=do not provide a vector during IAck cycles		
DICR12	VIS		1=include TypeCode in DMA interrupt vectors; 0=return vector as software wrote it to DIVR7-0		
DICR1	RxDMA IE		1=Rx DMA interrupt enable(d)		
DICR0	TxDMA IE		1=Tx DMA interrupt enable(d)		

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

DMA Interrupt Vector Register (DIVR) **Register Address 0 x b 01010**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Interrupt Vector 7-3 (RO)					Type Code (RO)		IV0 (RO)	Interrupt Vector (RW)							

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
DIVR15-11		Read DIVR15-8, or IAck w/ VIS=1 (DICR12)	as software wrote DIVR7-3	RO	7: DMA Interrupt Vectors

DIVR10-9	TypeCode		highest pending interrupt type:
			00=no DMA type pending;
			10=Tx DMA (no Rx DMA);
			11=Rx DMA
				RO	

DIVR8			as software wrote DIVR0	RO	
DIVR7-0		Read/Write DIVR7-0, or IAck w/ VIS=0 (DICR12)	basic 8-bit DMA interrupt vector	RW	

Hardware Configuration Register (HCR) **Register Address 1 0 b 01001**

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

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	4: Tx and Rx Clocking CTR0 and CTR1
HCR13	CTR1DSel		0=CTR0Div determines CTR1 divisor; 1=DPLLDiv determines CTR1 divisor		
HCR12	CVOK	Biphase	1=do not report single code violations		4: More About the DPLL
HCR11-10	DPLLDiv		00=DPLL divides by 32; 01=/16; 10=/8; 11=do not use for DPLL (/4 for CTR1)		4: Tx and Rx Clocking: Introduction to the DPLL

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
					4: More About the DPLL

HCR5	BRG1S		1=BRG1 single cycle mode; 0=continuous		4: Tx and Rx Clocking The Baud Rate Generators
HCR4	BRG1E		1=enable BRG1		
HCR1	BRG0S		1=BRG0 single cycle mode; 0=continuous		
HCR0	BRG0E		1=enable BRG0		

Input/Output Control Register (IOCR) Register Address 1 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	4: The /CTS Pin
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		4: The /DCD Pin
IOCR11-10	TxRMode		00=/TxREQ pin is an input; 01=drive /TxREQ with Transmit DMA Request; 10=drive /TxREQ Low; 11=drive /TxREQ High		4: The /RxREQ and /TxREQ Pins
IOCR9-8	RxRMode		00=/RxREQ pin is an input; 01=drive /RxREQ with Receive DMA Request; 10=drive /RxREQ Low; 11=drive /RxREQ High		
IOCR7-6	TxDMode		00=drive /TxD with Transmitter output; 01=release /TxD to high impedance; 10=drive /TxD Low; 11=drive /TxD High		4: The /RxD and /TxD Pins
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		4: The /RxC and /TxC Pins
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 = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Interrupt Control Register (ICR) **Register Address 0 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 serial controller	RW	7: Serial Interrupt Options
ICR14	DLC		1=disable Interrupt Enable Out (IEO)	RW	
ICR13	NV		1=do not 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	7: Serial Interrupt Enable Bits

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	

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Interrupt Vector Register (IVR) **Register Address 1 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	7: Serial Interrupt Vectors
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; 101=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 1 0 b 01110**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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

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	4: The /RxC and /TxC Pins
		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	4: The /RxREQ and /TxREQ pins
		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	4: The /DCD Pin
		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	4: The /CTS Pin
		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	5: DMA Support Features: The RCC FIFO
MISR2	DPLLDSync L/U		1=DPLL has lost sync	R,W1U	4: More About the DPLL 7: Miscellaneous Interrupt Sources and IA Bits
MISR1	BRG1 L/U		1=BRG1 has counted down to 0	R,W1U	4: Tx and Rx Clocking: The Baud Rate Generators
MISR0	BRG0 L/U		1=BRG0 has counted down to 0	R,W1U	

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Next Receive Address Register Lower (NRARL) — Register Address 0 1 b 11110

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LS 16 bits of "next receive address" (see below)															

Next Receive Address Register Upper (NRARU) — Register Address 0 1 b 11111

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MS 16 bits of "next receive address" (see below)															

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
NRARU15-0 NRARL15-0		Pipelined	32-bit address of next Rx DMA buffer	RW	6: Pipelined Mode 6: Array Mode 6: Linked List Mode
		Array or Linked List	32-bit address in Array or Linked List (used to fetch address and count of next Rx DMA buffer)		

Next Receive Byte Count Register (NRBCR) — Register Address 0 1 b 11101

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Length of next Rx DMA buffer															

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
NRBCR15-0		Pipelined	length of next Rx DMA buffer, in bytes	RW	6: Pipelined Mode

Next Transmit Address Register Lower (NTARL) **Register Address 0 0 b 11110**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LS 16 bits of "next transmit address" (see below)															

Next Transmit Address Register Upper (NTARU) **Register Address 0 0 b 11111**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MS 16 bits of "next transmit address" (see below)															

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
NTARU15-0 NTARL15-0		Pipelined	32-bit address of next Tx DMA buffer	RW	6: Pipelined Mode 6: Array Mode 6: Linked List Mode
		Array or Linked List	32-bit address in Array or Linked List (used to fetch address and count of next Tx DMA buffer)		

Next Transmit Byte Count Register (NTBCR) **Register Address 0 0 b 11101**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Length of next Rx DMA buffer															

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
NTBCR15-0		Pipelined	number of bytes in next Tx DMA buffer	RW	6: Pipelined Mode

Port Control Register (PCR) **Register Address 1 0 b 00101**

P7Mode		P6Mode		P5Mode		P4Mode		P3Mode		P2Mode		P1Mode		P0Mode	
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
PCR15-14	P7Mode		00=PORT7 pin is an input; 01=drive PORT7 with TxComplete; 10=drive PORT7 low; 11=drive PORT7 high	RW	4: The Port Pins

PCRR13-12	P6Mode		00=PORT6 pin is a GP input;
			01=PORT6 pin is /FSYNC input;
			10=drive PORT6 low; 11=drive PORT6 high
					4: The Port Pins
					4: The Time Slot Assigners

PCR11-10	P5Mode		00=PORT5 pin is an input; 01=drive PORT5 with /RxSYNC; 10=drive PORT5 low; 11=drive PORT5 high		4: The Port Pins
PCRR9-8	P4Mode		00=PORT4 pin is an input; 01=drive PORT4 with Tx TSA Gate; 10=drive PORT4 low; 11=drive PORT4 high		4: The Port Pins 4: The Time Slot Assigners
PCR7-6	P3Mode		00=PORT3 pin is an input; 01=drive PORT3 with Rx TSA Gate; 10=drive PORT3 low; 11=drive PORT3 high		
PCR5-4	P2Mode		00=PORT2 pin is an input; 01=drive PORT2 with LocalTalk driver enable signal; 10=drive PORT2 low; 11=drive PORT2 high		4: The Port Pins 4: LocalTalk (AppleTalk) Interface

PCR3-2	P1Mode		00=PORT1 pin is a GP input;
			01=PORT1 pin is CLK1 input;
			10=drive PORT1 low; 11=drive PORT1 high
					4: The Port Pins
					4: Tx and Rx Clocking: CTR0 and CTR1

PCR1-0	P0Mode		00=PORT0 pin is a GP input;
			01=PORT0 pin is CLK0 input;
			10=drive PORT0 low; 11=drive PORT0 high		

Port Status Register (PSR) **Register Address 1 0 b 00100**

P7L/U	/P7	P6L/U	/P6	P5L/U	/P5	P4L/U	/P4	P3L/U	/P3	P2L/U	/P2	P1L/U	/P1	P0L/U	/P0
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
PSR15	P7L/U		1=transition detected on PORT7 pin	R,W1U	4: The Port Pins
PSR14	/P7	P7L/U=1	1=rising edge on PORT7; 0=falling edge	RO	
		P7L/U=0	1=PORT7 was low last time P7L/U:=1; 0=PORT7 was high		
PSR13	P6L/U		1=transition detected on PORT6 pin	R,W1U	
PSR12	/P6	P6L/U=1	1=rising edge on PORT6; 0=falling edge	RO	
		P6L/U=0	1=PORT6 was low last time P6L/U:=1; 0=PORT6 was high		
PSR11	P5L/U		1=transition detected on PORT5 pin	R,W1U	
PSR10	/P5	P5L/U=1	1=rising edge on PORT5; 0=falling edge	RO	
		P5L/U=0	1=PORT5 was low last time P5L/U:=1; 0=PORT5 was high		
PSR9	P4L/U		1=transition detected on PORT4 pin	R,W1U	
PSR8	/P4	P4L/U=1	1=rising edge on PORT4; 0=falling edge	RO	
		P4L/U=0	1=PORT4 was low last time P4L/U:=1; 0=PORT4 was high		
PSR7	P3L/U		1=transition detected on PORT3 pin	R,W1U	
PSR6	/P3	P3L/U=1	1=rising edge on PORT3; 0=falling edge	RO	
		P3L/U=0	1=PORT3 was low last time P3L/U:=1; 0=PORT3 was high		
PSR5	P2L/U		1=transition detected on PORT2 pin	R,W1U	
PSR4	/P2	P2L/U=1	1=rising edge on PORT2; 0=falling edge	RO	
		P2L/U=0	1=PORT2 was low last time P2L/U:=1; 0=PORT2 was high		
PSR3	P1L/U		1=transition detected on PORT1 pin	R,W1U	
PSR2	/P1	P1L/U=1	1=rising edge on PORT1; 0=falling edge	RO	
		P1L/U=0	1=PORT1 was low last time P1L/U:=1; 0=PORT1 was high		
PSR1	P0L/U		1=transition detected on PORT0 pin	R,W1U	
PSR0	/P0	P0L/U=1	1=rising edge on PORT0; 0=falling edge	RO	
		P0L/U=0	1=PORT0 was low last time P0L/U:=1; 0=PORT0 was high		

Receive Address Register Lower (RARL) **Register Address 0 1 b 10110**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LS 16 bits of current Rx DMA buffer address															

Receive Address Register Upper (RARU) **Register Address 0 1 b 10111**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MS 16 bits of current Rx DMA buffer address															

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
RARU15-0 RARL15-0			32-bit address of next Rx DMA buffer	RW	6: DMA Fundamentals: Addresses and Byte Counts

Receive Byte Count Register (RBCR) **Register Address 0 1 b 10101**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Number of bytes left in current Rx DMA buffer															

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
RBCR15-0			Number of byte locations (left) in Rx DMA buffer	RW	6: DMA Fundamentals: Addresses and Byte Counts

Receive Character Count Register (RCCR) **Register Address 1 0 b 10110**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ending count of oldest received frame/message in RCC FIFO															

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	5: DMA Support Features: The RCC FIFO

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Receive Command/Status Register (RCSR) **Register Address 1 0 b 10010**

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

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=Select RICRHi=RTSA Data 0101=Select RICRHi=RxFIFO Status 0110=Select RICRHi=/INT Level 0111=Select RICRHi=/RxREQ Level 1xxx=Reserved	WO	5: Commands
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	5: Status Reporting: Detailed Status in the RCSR
RCSR14	1stBE		1=oldest byte in RxFIFO had RxBound, PE, or RxOver when RDR was last read	RO	
RCSR11-19	RxResidue	H/SDLC	000=frame ended at character boundary 001-111=number of extra bits at end	RO	5: HDLC/SDLC Mode: Frame Length Residuals
RCSR8	ShortF/ CVType	H/SDLC, CMR7-4 not xx00	1=received frame ended before Address/Control fields (see Note 1)	R,W1U or RO	5: Status Reporting: Detailed Status in the RCSR
		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 (global/real-time flag)		
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)		
		802.3	1=end of frame (see Note 2)		
		HDLC/ SDLC	1=Flag or Abort followed this character (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	Abort/PE	QAbort (RMR8)=0	1=parity error (see Note 2)	R,W1C or RO	
		H/SDLC, QAbort=1	1=Abort followed this character (see Note 2)		
RCSR1	RxOver		1=RxFIFO overflow (see Note 2)		
RCSR0	RxAvail		1=RxFIFO is not empty	RO	

Note 1: The IUSC 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 IUSC 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 (if interrupts are armed for this bit), as described in the referenced Chapter/Section.

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Receive Count Limit Register (RCLR) **Register Address 1 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	5: DMA Support Features: The Character Counters

Receive Data Register (RDR) **Register Address 1 0 b 1x000 or 1 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	5: The Data Registers and the FIFOs
RDR7-0			Received character		

Receive DMA Interrupt Arm Register (RDIAR) **Register Address 0 1 b 01111**

Reserved (0)	EOA/ EOL IA	EOB IA	HAbort IA	SAbort 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
RDIAR3	EOA/EOL IA	Array, Linked List	1=arm interrupt on End of Array/End of List (RMCR3)	RW	7: DMA Interrupt Sources and IA Bits
RDIAR2	EOB IA		1=arm interrupt on End of Buffer (RDMR2)		
RDIAR1	HAbort IA		1=arm interrupt on Hardware Abort (RDMR1)		
RDIAR0	SAbort IA		1=arm interrupt on Software Abort (RDMR0)		

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Receive DMA Mode Register (RDMR)

Register Address 0 1 b 00001

DMAMode		RSB InA/L	Clear Count	AddrMode		TermE	8/16	CONT	GLink	BUSY	INITG	EOA/EOL	EOB	HAbort	SAbort
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
RDMR15-14	DMAMode		00=Single Buffer; 01=Pipelined; 10=Array; 11=Linked List	RW	6: DMA Fundamentals
RDMR13	RSBinA/L	Array or Linked List	00=store Receive Status Blocks in data buffers after frames/messages; 1=store RSBs in Array/List entries;	RW	6: Storing Receive Status Blocks
RDMR12	ClearCount	Array or Linked List	1=clear Byte Count fields in Array/List entries to zero after fetching them	RW	6: Array Mode 6: Linked List Mode
RDMR11-10	AddrMode		00=increment addresses; 01=decrement addresses; 10=fixed address	RW	6: Address Sequencing
RDMR9	TermE		1=terminate buffer on RxBound	RW	6: DMA Fundamentals: Buffer Termination
RDMR8	8/16	16-bit bus	1=8-bit transfers; 0=16-bit transfers	RW	6: DMA Fundamentals: Data Width, Byte Ordering
RDMR7	CONT	Pipelined	1=software has issued a Start/Continue command after loading Next Address and Count	RO	6: Channel Status
RDMR6	GLink	Linked List	1=the channel is reading the Link address from a list entry, or it stopped while doing so	RO	
RDMR5	BUSY		1=the channel is operating per a Start command; 0=the channel is stopped	RO	
RDMR4	INITG	Array or Linked List	1=the channel is fetching information from the array or linked list, or it stopped while doing so	RO	
RDMR3	EOA/EOL	Array or Linked List	1=the channel has reached the end of the array or list, signified by a zero Byte Count field	ROC	
RDMR2	EOB		1=the channel has reached the end of the buffer	ROC	
RDMR1	HAbort		1=the channel has stopped because the /ABORT pin went low while it was doing a memory cycle	ROC	
RDMR0	SAbort		1=software stopped the channel via an Abort command	ROC	

Receive Interrupt Control Register (RICR) **Register Address 1 0 b 10011**

15–8	7	6	5	4	3	2	1	0
"RTSA data" if last RCSR15-12 command 4-7 was 4 "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	Abort /PE IA	RxOver IA	TCOR Sel

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
RICR15-9	RTSASlot	4 written to RCmd since 5-7 written there, read or write w/RICR8=0	"slot number" (number of bytes from frame sync) at which to activate Rx in each frame	RW	4: The Time Slot Assigners
RICR15-13	RTSAOffset	4 written to RCmd since 5-7 written there, write w/RICR8=1	"offset" (number of bits delay) at which to activate Rx in each frame	WO	
RICR12-9	RTSACount	4 written to RCmd since 5-7 written there, write w/RICR8=1	0000=disable Rx Time Slot Assigner 0001-1111=number of consecutive bytes/ octets/time slots to receive in each frame	WO	
RICR15-8		5 written to RCmd, or Reset, since 4, 6, or 7 written there	the number of characters/bytes/octets currently in the RxFIFO	RO	5: The Data Registers and the FIFOs
RICR15-8		6 written to RCmd since 4, 5, or 7 written there	number of characters/bytes/octets in the RxFIFO, above which to request a Receive Data interrupt	RW	7: Receive Data interrupts
RICR15-8		7 written to RCmd since 4-6 written there	number of characters/bytes/octets in the RxFIFO, above which to request a Receive DMA transfer ***This value must be at least 1 for 16-bit Rx DMA operation***	RW	6: DMA Requests by the Receiver and Transmitter
RICR7	ExitedHunt IA		1=arm interrupts on ExitedHunt (RCSR7)	RW	7: Receive Status Interrupt Sources and IA Bits
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	5: Status Reporting: Detailed Status in the RCSR
RICR2	Abort/PE IA		1=arm interrupts on Abort/PE (RCSR2)	RW	7: Receive Status Interrupt Sources and IA bits
RICR1	RxOver IA		1=arm interrupts on RxOver (RCSR1)	RW	
RICR0	TCOR Sel		0=select Time Constant value for reading TCOR 1=capture current count for reading TCOR	RW	4: Tx and Rx Clocking: The Baud Rate Generators

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Receive Mode Register (RMR)

Register Address 1 0 b 10001

RxDecode			RxCRCType		RxCRC Start	RxCRC Enab	QAbort	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	4: Data Formats and Encoding
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		5: Cyclic Redundancy Checking
RMR10	RxCRCStart	Sync	0=start Receive CRC generator as all-zeros; 1=all ones		
RMR9	RxCRCEnab	Sync	1=include Receive characters in CRC		
RMR8	QAbort	HDLC/ SDLC	1=use Abort/PE bit in RxFIFO and RCSR2 for Abort indication		5: Status Reporting: Detailed Status in the RCSR 5: HDLC/SDLC: Handling a Received Abort
			0=use Abort/PE for Parity Error indication		
RMR7-6	RxParType		00=Receive Parity Even; 01=Odd; 10=Zero (Space); 11=One (Mark)		5: Parity Checking
RMR5	RxParEnab		1=accumulate and check Parity bits		
RMR4-2	RxLength		000=receive eight bit characters; 001-111=receive 1-7 bit characters		5: The Mode Registers: Character Length
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		

Receive Sync Register (RSR)

Register Address 1 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	RW	5: Monosync and Bisync Modes 5: 802.3 (Ethernet) Mode
		Bisync	second half of Receive sync match (SYN1)		
		802.3	match against second-received 8 bits of address		
RSR7-0		Bisync	first half of Receive sync match (SYN0)		5: Monosync and Bisync Modes 5: HDLC/SDLC Mode 5: 802.3 (Ethernet) Mode
		H/SDLC, (CMR7-4) not xx00, 802.3	match against first-received 8 bits of address		

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Set DMA Interrupt Register (SDIR) **Register Address 0 x b 01110**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved (0)						RxDMA IUS	TxDMA IUS	Reserved (0)						RxDMA IP	TxDMA IP

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
SDIR9	RxDMA IUS	Read	1=Rx DMA interrupt under service	RO	7: DMA IP and IUS Bits
		Write	1=set Rx DMA IUS bit; 0=no change	WO	
SDIR8	TxDMA IUS	Read	1=Tx DMA interrupt under service	RO	
		Write	1=set Tx DMA IUS bit; 0=no change	WO	
SDIR1	RxDMA IP	Read	1=Rx DMA interrupt pending	RO	
		Write	1=set Rx DMA IP bit; 0=no change	WO	
SDIR0	TxDMA IP	Read	1=Tx DMA interrupt pending	RO	
		Write	1=set Tx DMA IP bit; 0=no change	WO	

Status Interrupt Control Register (SICR) **Register Address 1 0 b 01111**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	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

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	4: The /RxC and /TxC Pins
SICR14	RxCUp IA		1=set MISR15/interrupt on rise of /RxC		
SICR13	TxCDn IA		1=set MISR13/interrupt on fall of /TxC		
SICR12	TxCUp IA		1=set MISR13/interrupt on rise of /TxC		
SICR11	RxRDn IA		1=set MISR11/interrupt on fall of /RxREQ		4: /RxREQ and /TxREQ Pins
SICR10	RxRUp IA		1=set MISR11/interrupt on rise of /RxREQ		
SICR9	TxRDn IA		1=set MISR9/interrupt on fall of /TxREQ		
SICR8	TxRUp IA		1=set MISR9/interrupt on rise of /TxREQ		
SICR7	DCDDn IA		1=set MISR7/interrupt on fall of /DCD		4: The /DCD Pin
SICR6	DCDUp IA		1=set MISR7/interrupt on rise of /DCD		
SICR5	CTSDn IA		1=set MISR5/interrupt on fall of /CTS		4: The /CTS Pin
SICR4	CTSUp IA		1=set MISR5/interrupt on rise of /CTS		
SICR3	RCC Under IA	RCC used	1=interrupt on RCC underflow (Receive frame/message longer than max allowed)		5: DMA Support Features: The RCC FIFO
SICR2	DPLLDSync IA	Biphase	1=interrupt on DPLL sync loss		4: More About the DPLL 7: Miscellaneous Interrupt Sources and IA Bits
SICR1	BRG1 IA		1=interrupt on BRG1 zero		4: Tx and Rx Clocking: The Baud Rate Generators
SICR0	BRG0 IA		1=interrupt on BRG0 zero		

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Test Mode Control Register (TMCR)

Register Address 1 0 b 00111

15–5	4–0
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 or write in TMDR	RW	8: Serial Controller Test Modes

Test Mode Data Register (TMDR)

Register Address 1 0 b 00110

15–0
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
TMDR15-0			Test register selected by TMCR4-0	Varies	8: Serial Controller Test Modes

Time Constant 0 Register (TC0R)

Register Address 1 0 b 10111

15–0
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	5: DMA Support Features: The Character Counters
		Read w/ TC0RSel (RICR0)=1	Value of BRG0 counter last time TC0RSel:=1	RO	

Time Constant 1 Register (TC1R)

Register Address 1 0 b 11111

15–0
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 (RICR0)=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
		Read w/ TC1RSel (RICR0)=1	Value of BRG1 counter last time TC1RSel:=1	RO	

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Transmit Address Register Lower (TARL) **Register Address 0 0 b 10110**

LS 16 bits of current Tx DMA buffer address
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Transmit Address Register Upper (TARU) **Register Address 0 0 b 10111**

MS 16 bits of current Tx DMA buffer 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
TARU15-0 TARL15-0			32-bit current address in Tx DMA buffer	RW	6: DMA Fundamentals: Addresses and Byte Counts

Transmit Byte Count Register (TBCR) **Register Address 0 0 b 10101**

Number of bytes left to send in current Tx DMA buffer
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
TBCR15-0			Number of bytes left to send in Tx DMA buffer	RW	6: DMA Fundamentals: Addresses and Byte Counts

Transmit Character Count Register (TCCR) **Register Address 1 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	5: DMA Support Features: The Character Counters

Transmit Command/Status Register (TCSR) **Register Address 1 0 b 11010**

TCmd				Under Wait	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	5: Commands
		Sync	0010=Clear Tx CRC Generator		
			0011=reserved 0100=Select TICRHi=TTSA Data 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		
TCSR11	UnderWait	Sync	1=interlock Transmitter from Tx underrun until Send Frame command. Also, if TxCtrlBlk (CCR15-14) is 10 = 32-bit TCBs, delay start of frame transmission until TxFIFO is full or complete frame written to TxFIFO.	RW	5: Handling Overruns and Underruns: Tx Underruns
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	5: Between Messages, Frames, or Characters
TCSR7	PreSent	Sync	1=Transmitter has finished sending Preamble	R,W1U	5: Status Reporting: Detailed Status in the TCSR
TCSR6	IdleSent		1=Transmitter has sent Idle condition		
TCSR5	AbortSent	H/SDLC	1=Transmitter has sent Abort		
TCSR4	EOF/EOM Sent	Sync	1=Transmitter has sent End of Frame/End of Message		
TCSR3	CRCSent	Sync	1=Transmitter has sent a CRC code		
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 1 0 b 11101**

15–0
Starting value for Transmit Character Counter

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, in bytes	RW	5: DMA Support Features: The Character Counters

Transmit Data Register (TDR) **Register Address 1 0 b 1x000 or 1 1 b xxxxx**

15–8	7–0
Transmit character: write only using 16-bit operation	Transmit character: 8- or 16-bit write

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	5: The Data Registers and the FIFOs
TDR7-0			Transmit character		

Transmit DMA Interrupt Arm Register (TDIAR) **Register Address 0 0 b 01111**

15–4	3	2	1	0
Reserved (0)	EOA/ EOL IA	EOB IA	HAbort IA	SAbort IA

See the description of the Receive DMA Interrupt Arm Register (RDIAR). This one is identical except that it arms status bits in the TDMR rather than the RDMR.

Transmit DMA Mode Register (TDMR) **Register Address 0 0 b 00001**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAMode		TCB InA/L	Clear Count	AddrMode		TermE	8/16	CONT	GLink	BUSY	INITG	EOA/ EOL	EOB	HAbort	SAbort

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
TDMR15-14	DMAMode		00=Single Buffer; 01=Pipelined; 10=Array; 11=Linked List	RW	6: DMA Fundamentals
TDMR13	TCBinA/L	Array or Linked List with TCBs	0=fetch Transmit Control Blocks from data buffers before start of frames/messages; 1=fetch TCBs from Array/List entries		6: Fetching Transmit Control Blocks
TDMR12	ClearCount	Array or Linked List	1=clear Byte Count fields in Array/List entries to zero after fetching them		6: Array Mode 6: Linked List Mode
TDMR11-10	AddrMode		00=increment addresses; 01=decrement addresses; 10=fixed address		5: Address Sequencing
TDMR9	TermE		1=terminate buffer at end of Tx frame		6: DMA Fundamentals: Buffer Termination
TDMR8	8/16	16-bit bus	1=8-bit transfers 0=16-bit transfers		6: DMA Fundamentals: Data Width, Byte Ordering
TDMR7	CONT	Pipelined	1=software has issued a Start/Continue command after loading Next Address and Count	RO	6: Channel Status
TDMR6	GLink	Linked List	1=the channel is reading the Link address from a list entry, or it stopped while doing so		
TDMR5	BUSY		1=the channel is operating per a Start command; 0=the channel is stopped		
TDMR4	INITG	Array or Linked List	1=the channel is fetching information from the array or linked list, or it stopped while doing so		
TDMR3	EOA/EOL	Array or Linked List	1=the channel has reached the end of the array or list, as signified by a zero Byte Count field	ROC	
TDMR2	EOB		1=the channel has reached the end of a buffer		
TDMR1	HAbort		1=the channel stopped because the /ABORT pin went low while it was doing a memory cycle		
TDMR0	SAbort		1=software stopped the channel via an Abort command		

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Transmit Interrupt Control Register (TICR) **Register Address 1 0 b 11011**

15–8	7	6	5	4	3	2	1	0
"TTSA data" if last TCSR15-12 command 4-7 was 4 "TxFIFO fill level" if last TCSR15-12 command 4-7 was 5 "Tx Int Req level" if last TCSR15-12 command 4-7 was 6 "Tx DMA Req 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

Bit(s)	Field/Bit Name	Conditions /Context	Description	RW Status	Ref Chapter: Section
TICR15-9	TTSASlot	4 written to TCmd since 5-7 written there, read or write w/TICR8=0	"slot number" (number of bytes from frame sync) at which to activate Tx in each frame	RW	4: The Time Slot Assigners
TICR15-13	TTSAOffset	4 written to TCmd since 5-7 written there, write w/TICR8=1	"offset" (number of bits delay) at which to activate Tx in each frame	WO	
TICR12-9	TTSACount	4 written to TCmd since 5-7 written there, write w/TICR8=1	0000=disable Tx Time Slot Assigner 0001-1111=number of consecutive bytes/ octets/time slots to receive in each frame	WO	
TICR15-8		5 written to TCmd, or Reset, since 4, 6, or 7 written there	the number of characters/bytes/octets currently in the TxFIFO	RO	5: The Data Registers and the FIFOs
TICR15-8		6 written to TCmd since 4, 5, or 7 written there	the number of empty characters/bytes/octets in the TxFIFO, above which to request a Transmit Data interrupt	RW	7: Transmit Data interrupts
TICR15-8		7 written to RCmd since 4-6 written there	the number of empty characters/bytes/octets in the TxFIFO, above which to request a Transmit DMA transfer ***This value must be at least 1 for 16-bit Tx DMA operation***	RW	6: DMA Requests by the Receiver and Transmitter
TICR7	PreSent IA	Sync	1=arm interrupts on Preamble Sent (TCSR7)	RW	7: Transmit Status Interrupt Sources and IA Bits
TICR6	IdleSent IA		1=arm interrupts on IdleSent (TCSR6)		
TICR5	AbortSent IA	H/SDLC	1=arm interrupts on AbortSent (TCSR5)		
TICR4	EOF/EOM Sent IA	Sync	1=arm interrupts on EOF/EOM Sent (TCSR4)		
TICR3	CRCSent IA	Sync	1=arm interrupts on CRCSent (TCSR3)		
TICR2	Wait2Send	Sync	1=hold Transmitter from sending each frame/message until software issues "Send Message/Frame" command	RW	5: Synchronizing Frames/ Messages with Software Response
TICR1	TxUnder IA		1=arm interrupts on TxUnder (TCSR1)	RW	7: Transmit Status Interrupt Sources and IA Bits
TICR0	TC1R Sel		0=select Time Constant value for reading TC1R 1=capture current count for reading TC1R	RW	4: Tx and Rx Clocking: The Baud Rate Generators

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

Transmit Mode Register (TMR) **Register Address 1 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=do not 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	4: Data Formats and Encoding
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		5: Cyclic Redundancy Checking
TMR10	TxCRCStart	Sync	0=start Transmit CRC generator as all-zeros; 1=all ones		
TMR9	TxCRCEnab	Sync	0=include Transmit characters in CRC		
TMR8	TxCRCat End	Sync	1=send accumulated CRC at EOF/EOM		
TMR7-6	TxParType		00=Transmit Parity Even; 01=Odd; 10=Zero (Space); 11=One (Mark)		5: Parity Checking
TMR5	TxParEnab		1=accumulate and send Parity bits		
TMR4-2	TxLength		000=send eight bit characters; 001-111=send 1-7 bit characters		5: The Mode Registers: Character Length
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		5: The Mode Registers: Enabling and Disabling

Transmit Sync Register (TSR) **Register Address 1 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)	RW	5: Monosync and Bisync Modes 5: Slaved Monosync Mode
RSR7-0		Monosync, Slaved Monosync	Transmit Sync character		
		Bisync	First half of Transmit sync (SYN0)		

RW = Read/Write, RO = Read Only, WO = Write Only. For other codes see page 8-11.

[illegible]

Appendix: Changes 9

Application Notes

Superintegration™ Products Guide

Zilog's General Terms and Conditions of Sale T

Zilog Sales Offices Representatives & Distributors Z

Zilog's Literature Guide Ordering Information L

CHAPTER 9
APPENDIX: CHANGES

9.1 INTRODUCTION

This section summarizes the changes in the names of registers and commands since the original USC Technical Manual.

9.2 BASIC TERMINOLOGY

Transmit Status Blocks—>Transmit Control Blocks
The names of registers and other USC features, in past documentation, maintained the distinction between "status" information 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.

9.3 COMMANDS

Reload RCC / TCC —> Load RCC/TCC
It was not 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...

9.4 BIT/FIELD NAMES

Bit and field names not really used in the early USC manuals — 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 is 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.