Am186™CC/CH/CU Microcontrollers

User's Manual

Order #21914B

AMD

IF YOU HAVE QUESTIONS, WE'RE HERE TO HELP YOU.

The AMD customer service network includes U.S. offices, international offices, and a customer training center. Expert technical assistance is available from the AMD worldwide staff of field application engineers and factory support staff to answer E86™ and Comm86™ family hardware and software development questions.

Frequently accessed numbers are listed below. Additional contact information is listed on the back of this manual. AMD's WWW site lists the latest phone numbers.

Technical Support

Answers to technical questions are available online, through e-mail, and by telephone.

Go to AMD's home page at **www.amd.com** and follow the **Service** link for the latest AMD technical support phone numbers, software, and Frequently Asked Questions.

For technical support questions on all E86 and Comm86 products, send e-mail to **epd.support@amd.com** (in the US and Canada) or **euro.tech@amd.com** (in Europe and the UK).

You can also call the AMD Corporate Applications Hotline at:

(800) 222-9323 Toll-free for U.S. and Canada

44-(0) 1276-803-299 U.K. and Europe hotline

WWW Support

For specific information on E86 and Comm86 products, access the AMD home page at **www.amd.com** and follow the **Embedded Processors** link. These pages provide information on upcoming product releases, overviews of existing products, information on product support and tools, and a list of technical documentation. Support tools include online benchmarking tools and CodeKit™ software—tested source code example applications. Many of the technical documents are available online in PDF form.

Questions, requests, and input concerning AMD's WWW pages can be sent via e-mail to **webmaster@amd.com**.

Documentation and Literature Support

Data books, user's manuals, data sheets, application notes, and product CDs are free with a simple phone call. Internationally, contact your local AMD sales office for product literature.

To order literature, call:

(800) 222-9323 Toll-free for U.S. and Canada

(512) 602-5651 Direct dial worldwide

(512) 602-7639 Fax

Third-Party Support

AMD FusionE86℠ partners provide an array of products designed to meet critical time-to-market needs. Products and solutions available include emulators, hardware and software debuggers, board-level products, and software development tools, among others. The WWW site and the *E86 Family Products Development Tools CD, order# 21058*, describe these solutions. In addition, mature development tools and applications for the x86 platform are widely available in the general marketplace.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

COMM86 FAMILY

The Am186™CC communications controller, Am186CH HDLC microcontroller, and Am186CU USB microcontroller, the first members of the AMD Comm86™ family, are cost-effective, high-performance embedded microcontroller solutions for communications applications. These highly integrated microcontrollers enable customers to save system costs and increase performance over 8-bit microcontrollers and other 16-bit microcontrollers.

All of these microcontrollers offer the advantages of the x86 development environment's widely available native development tools, applications, and system software. Additionally, the microcontrollers use the industry-standard 186 instruction set that is part of the AMD E86™ family, which continually offers instruction-set-compatible upgrades. Built into each of the microcontrollers is a wide range of communications features required in many communications applications.

PURPOSE OF THIS MANUAL

This manual describes the technical features and programming interface of the Am186CC, Am186CH, and Am186CU microcontrollers.

Intended Audience

The *Am186CC/CH/CU Microcontrollers User's Manual*, order #21914, is intended for computer software and hardware engineers and system architects who are designing or are considering designing systems based on one of these controllers.

Overview of this Manual

This manual is organized into the following chapters:

- Chapter 1, "Architectural Overview," provides an overview of the features of the microcontrollers, including a block diagram and sample application diagrams.
- Chapter 2, "Configuration Basics," provides basic information about configuring the microcontrollers, including discussions of the registers, memory organization, address generation, I/O space, peripheral control block, instruction set, segments, data types, and addressing modes.
- Chapter 3, "System Overview," contains descriptions of the microcontrollers' system configuration registers, initialization and processor reset, signals, bus interface, and clock control.
- Chapter 4, "Emulator Support," describes the various features available in the microcontrollers to facilitate the design and operation of In-Circuit Emulators, and discusses common concerns shared among emulator developers.
- Chapter 5, "Chip Selects," describes the six chip selects provided for use with memory devices and the eight provided for use with peripherals in either memory or I/O space.
- Chapter 6, "DRAM Controller," discusses the fully integrated DRAM controller that provides a glueless interface to 40-, 50-, 60-, and 70-ns Extended Data Out (EDO) DRAM.

- Chapter 7, “Interrupts,” describes the microcontrollers’ support for interrupts, both maskable and nonmaskable. It discusses interrupt sequence and priority as well as how to configure the maskable interrupt sources through the interrupt channels. It also describes the nonmaskable interrupts.
- Chapter 8, “DMA Controller,” describes how to use the DMA channels (general-purpose and SmartDMA channels) to transfer data between memory and internal and external peripherals.
- Chapter 9, “Programmable I/O Signals,” discusses the user programmable input/output signals (PIOs).
- Chapter 10, “Programmable Timers,” tells how to use the programmable timers for the following tasks: counting or timing external events, generating nonrepetitive or variable-duty-cycle waveforms, generating interrupts, supporting real-time coding and time-delay applications through polling, prescaling the other timer, requesting DMA, or measuring pulse widths (as a PWD).
- Chapter 11, “Watchdog Timer,” describes how to use the watchdog timer to generate nonmaskable interrupts (NMIs), microcontroller resets, and system resets when the programmable time-out value is reached.
- Chapter 12, “Serial Communications Overview,” discusses the serial communications features of the microcontrollers and their trade-offs, and provides a brief overview of serial communications.
- Chapter 13, “Asynchronous Serial Ports (UARTs),” describes how to use the UART and High-Speed UART for asynchronous serial data transfer.
- Chapter 14, “Synchronous Serial Port (SSI),” discusses how to use the SSI synchronous serial port to provide half-duplex, bidirectional communications between the microcontrollers and other system components

CC CH
- Chapter 15, “High-Level Data Link Control (HDLC),” provides a brief overview of HDLC and describes how to configure the HDLC channels to support data movement in a variety of applications.

CC CH
- Chapter 16, “HDLC External Serial Interface Configuration (TSAs),” describes how to use the time-division multiplex features to configure the HDLC external serial interfaces. Each Time-Slot Assigner (TSA) can be programmed to select between raw DCE and dedicated PCM Highway external interfaces, as well as to multiplex GCI/PCM Highway data.

CC
- Chapter 17, “General Circuit Interface (GCI),” discusses how to configure the GCI controller for a GCI interface on HDLC Channel A, or multiplexed GCI/PCM Highway interfaces to the other channels

CC CU
- Chapter 18, “Universal Serial Bus (USB),” covers the highly flexible integrated USB peripheral controller and how to implement a variety of microcontroller-based USB peripheral devices for telephony, audio, or other high-end applications.
- Appendix A, “Register Summary,” provides a summary of all the microcontroller peripheral control block (PCB) registers, listed in offset order.
- The Glossary provides definitions of significant terms used in this manual.
- The Index contains extensive references to make it easier to find specific information.

RELATED DOCUMENTS

The following documents contain additional information that will be useful in designing an embedded application based on the Am186CC/CH/CU microcontrollers.

AMD Documentation

In addition to this manual, the documentation set for the Am186CC/CH/CU microcontrollers includes the following documents:

- The *Am186™CC Communications Controller Data Sheet*, order #21915, the *Am186™CH HDLC Microcontroller Data Sheet*, order #22024, and the *Am186™CU USB Microcontroller Data Sheet*, order #22025, include complete pin lists, pin state tables, timing and thermal characteristics, and package dimensions for their particular microcontroller.
- The *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916, fully describes all the registers required to program the microcontrollers.
- The *Am186 and Am188 Family Instruction Set Manual*, order #21267, provides a detailed description and examples for each instruction included in the Am186 and Am188 family instruction set.
- *Interfacing an Am186™CC Communications Controller to an AMD SLAC™ Device Using the Enhanced SSI*, order #21921, application note describes how to connect these two devices. The same techniques can be used to connect the Am186CC microcontroller to any SLAC device.
- *Am186™CC/CH/CU Communications Controller Customer Development Platform Board Manual*, order #22002, which describes a platform for silicon evaluation and software development, as well as a router/ISDN terminal adapter module.

Other information of interest includes:

- *E86™ Family Products and Development Tools CD*, order #21058, provides a single-source multimedia tool for customer evaluation of AMD products, as well as FusionE86^{SM} partner tools and technologies that support the E86 and Comm86 families. Technical documentation is included on the CD in PDF format.
- *IOM-2 Interface Reference Guide*, order #12576, describes the terminal version of the IOM-2/GCI interface.

To order literature, contact the nearest AMD sales office or call the literature center at one of the numbers on the back cover of this manual. In addition, many documents are available in PDF form on the AMD web site. To access the AMD home page, go to **www.amd.com**. Then follow the Embedded Processors link for information about E86 and Comm86 products.

Additional Information

CC CU

The following non-AMD documents and sources provide additional information that may be of interest to Am186CC and Am186CU microcontroller users:

- *Universal Serial Bus Specification, Revision 1.0*, available from the USB web site at **http://www.usb.org**.
- *Universal Serial Bus System Architecture*, by Don Anderson, Mindshare, Inc., Addison Wesley Developers Press, 1997.

DOCUMENTATION CONVENTIONS

Table 0-1 lists the documentation conventions used throughout this manual.

Table 0-1 Documentation Conventions

Notation	Meaning
General	
bit	A single bit in a register
bit field	Two or more consecutive and related bits
set the EN bit	Write a 1 to the EN bit
clear the EN bit	Write a 0 to the EN bit
external reset	A reset caused by asserting the $\overline{\text{RES}}$ input signal
internal reset	A reset initiated by the watchdog timer (see Chapter 11, “Watchdog Timer”)
system reset	Assertion of the RESOUT signal to reset external peripherals. An external reset always causes a system reset; an internal reset can optionally cause a system reset.
offset 000h	A register offset, relative to the base of the current PCB space defined in the Relocation (RELOC) register
Pin Naming	
{ }	Pin function during hardware reset (pinstrap)
[ ]	Alternative pin function
pin	Refers to the physical wire.
signal	Refers to the electrical signal that flows across a pin.
$\overline{\text{SIGNAL}}$	A line over a signal name indicates that the signal is active Low; a signal name without a line is active High.
$\overline{\text{MCS3}}$–$\overline{\text{MCS0}}$	All four signals (or registers, or fields)
$\overline{\text{MCSx}}$	Any of the four signals (or registers, or fields)
Numbers	
b	Binary number
d	Decimal number Decimal is the default radix
h	Hexadecimal number
x in a number	Any of several values is legal; for example, 0x01b can be either 0001b or 0101b
[X–Y]	The bit field that consists of bits X through Y

Table 0-1 Documentation Conventions (Continued)

Notation	Meaning
Microcontroller-Specific Information Icons	
CC	Information specific to the Am186CC Communications Controller
CH	Information specific to the Am186CH HDLC Microcontroller
CU	Information specific to the Am186CU USB Microcontroller

MICROCONTROLLER-SPECIFIC INFORMATION

This manual provides information that applies to all three of the Am186CC/CH/CU microcontrollers as well as information that is specific to each controller. To help identify controller-specific information, this manual uses icons in the margin and within tables and figures. Table 0-1 illustrates these icons.

Some chapters apply only to one or two of the controllers. These chapters have a note at the beginning of the chapter with the relevant icons next to the note.

One or more icons immediately following a heading indicates that the information under that heading (up to the next heading) applies only to the indicated controllers.

Icons that appear other than at the beginning of the chapter or immediately following a heading apply to the specific paragraph, list, figure, portion of figure, table, or table cell indicated by the icon. If a paragraph, list, figure, or table does not have any accompanying icons, the information applies to all of the microcontrollers covered by the chapter.

CHAPTER

1 ARCHITECTURAL OVERVIEW

1.1 FEATURES

The Am186CC, Am186CH, and Am186CU microcontrollers, the first members of the AMD Comm86™ family, are cost-effective, high-performance microcontroller solutions for communications applications. These highly integrated microcontrollers enable customers to save system costs and increase performance over 8-bit microcontrollers and other 16-bit microcontrollers.

These microcontrollers offer the advantages of the x86 development environment's widely available native development tools, applications, and system software. Additionally, these microcontrollers use the industry-standard 186 instruction set that is part of the AMD E86 family, which continually offers instruction-set-compatible upgrades. Use of this instruction set ensures both backward and upward software compatibility.

AMD offers complete solutions with these microcontrollers. A customer development platform board for silicon evaluation and software development is available. Reference designs under development include a low-end router with Integrated Services Digital Network (ISDN), Ethernet, USB, and Plain Old Telephone Service (POTS), as well as an ISDN terminal adapter featuring USB. AMD and its FusionE86^{SM} Partners offer boards, schematics, drivers, protocol stacks, and routing software for these reference designs to enable fast time to market.

1.2 Am186CC COMMUNICATIONS CONTROLLER

CC

Built into the Am186CC microcontroller is a wide range of communications features required in many communications applications, including High-level Data Link Control (HDLC) and the Universal Serial Bus (USB). It includes the following distinctive characteristics:

- Serial communications peripherals
  - Four High-level Data Link Control (HDLC) channels
  - USB peripheral controller
  - Eight SmartDMA™ channels to support HDLC and USB
  - Four independent Time Slot Assigners (TSAs)
  - Physical interface for HDLC channels can be raw DCE, PCM Highway, or GCI (IOM-2)
  - High-speed UART with autobaud
  - UART
  - Synchronous serial interface (SSI)

- System peripherals
  - Interrupt controller (36 maskable interrupts)
  - Four general-purpose DMA channels
  - 48 programmable I/O signals
  - Three programmable 16-bit timers
  - Hardware watchdog timer
- Memory and Peripheral Interface
  - Integrated DRAM controller
  - Glueless interface to RAM/ROM/Flash memory (40-ns Flash memory required for zero-wait-state operation at 50 MHz)
  - Fourteen chip selects (6 memory, 8 peripheral)
  - External bus mastering support
  - Multiplexed and nonmultiplexed address/data bus
  - Programmable bus sizing
  - 8-bit boot option

1.2.1 Am186CH HDLC Microcontroller

CH

The Am186CH HDLC microcontroller is a cost-reduced derivative of the Am186CC microcontroller that is targeted towards cost-sensitive applications such as linecards and digital phones. The Am186CH HDLC microcontroller is pin-compatible with the Am186CC microcontroller and offers many of the same features, yet the Am186CH HDLC microcontroller provides a cost-effective solution for communications devices that require fewer HDLC channels and do not need GCI or USB. It includes the following distinctive characteristics:

- Serial communications peripherals
  - Two High-level Data Link Control (HDLC) channels
  - Four SmartDMA channels to support HDLC
  - Two independent Time Slot Assigners (TSAs)
  - Physical interface for HDLC channels can be raw DCE or PCM Highway
  - High-speed UART with autobaud
  - UART
  - Synchronous serial interface (SSI)
- System peripherals
  - Interrupt controller (31 maskable interrupts)
  - Four general-purpose DMA channels
  - 48 programmable I/O signals
  - Three programmable 16-bit timers
  - Hardware watchdog timer

- Memory and Peripheral Interface
  - Integrated DRAM controller
  - Glueless interface to RAM/ROM/Flash memory (40-ns Flash memory required for zero-wait-state operation at 50 MHz)
  - Fourteen chip selects (6 memory, 8 peripheral)
  - External bus mastering support
  - Multiplexed and nonmultiplexed address/data bus
  - Programmable bus sizing
  - 8-bit boot option

1.2.2 Am186CU USB Microcontroller

CU

The Am186CU USB microcontroller is a cost-reduced derivative of the Am186CC microcontroller that is targeted towards cost-sensitive applications such as USB peripherals and digital-subscriber- line (DSL) modems. The Am186CU USB microcontroller is pin-compatible with the Am186CC microcontroller and offers many of the same features, yet the Am186CU USB microcontroller provides a cost-effective solution for USB devices that do not need GCI or HDLC. It includes the following distinctive characteristics:

- Serial communications peripherals
  - USB peripheral controller
  - Four SmartDMA channels to support USB
  - High-speed UART with autobaud
  - UART
  - Synchronous serial interface (SSI)
- System peripherals
  - Interrupt controller (30 maskable interrupts)
  - Four general-purpose DMA channels
  - 48 programmable I/O signals
  - Three programmable 16-bit timers
  - Hardware watchdog timer
- Memory and Peripheral Interface
  - Integrated DRAM controller
  - Glueless interface to RAM/ROM/Flash memory (40-ns Flash memory required for zero-wait-state operation at 50 MHz)
  - Fourteen chip selects (6 memory, 8 peripheral)
  - External bus mastering support
  - Multiplexed and nonmultiplexed address/data bus
  - Programmable bus sizing
  - 8-bit boot option

1.2.3 Feature Comparison

Table 1-1 summarizes and compares the features of each of the microcontrollers.

Table 1-1 Feature Comparison

Feature	CC	CH	CU
HDLC Channels	4	2	–
Time Slot Assigners (TSAs)	4	2	–
Raw DCE Interface	✔	✔	–
PCM Highway Interface	✔	✔	–
GCI (IOM-2) Interface	✔	–	–
USB Peripheral Controller	✔	–	✔
SmartDMA Channels	8 (4 pair)	4 (2 pair)	4 (2 pair)
General-Purpose DMA Channels	4	4	4
High-Speed UART	✔	✔	✔
UART	✔	✔	✔
Synchronous Serial Interface (SSI)	✔	✔	✔
Internal Maskable Interrupts	19	14	13
External Maskable Interrupts	17	17	17
Programmable I/O Signals (PIOs)	48	48	48
16-Bit Timers	3	3	3
Hardware Watchdog Timer	✔	✔	✔
Integrated DRAM Controller	✔	✔	✔
Glueless Interface to RAM/ROM/Flash Memory	✔	✔	✔
Memory Chip Selects	6	6	6
Peripheral Chip Selects	8	8	8
External Bus Mastering Support	✔	✔	✔
Multiplexed and Nonmultiplexed Address/Data Bus	✔	✔	✔
Programmable Bus Sizing	✔	✔	✔
8-Bit Boot Option	✔	✔	✔

1.3 BLOCK DIAGRAMS

Figure 1-1, Figure 1-2, and Figure 1-3 show the block diagrams for the Am186CC/CH/CU microcontrollers, respectively.

Figure 1-1 Am186CC Communications Controller Block Diagram

CC

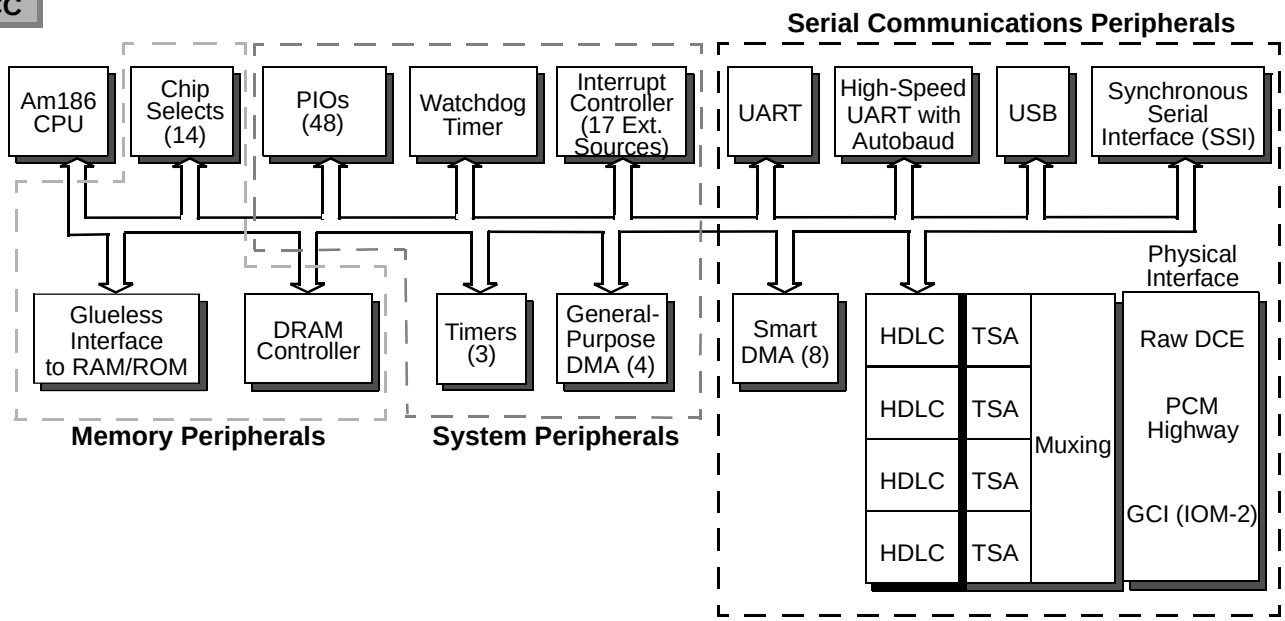


Figure 1-2 Am186CH HDLC Microcontroller Block Diagram

CH

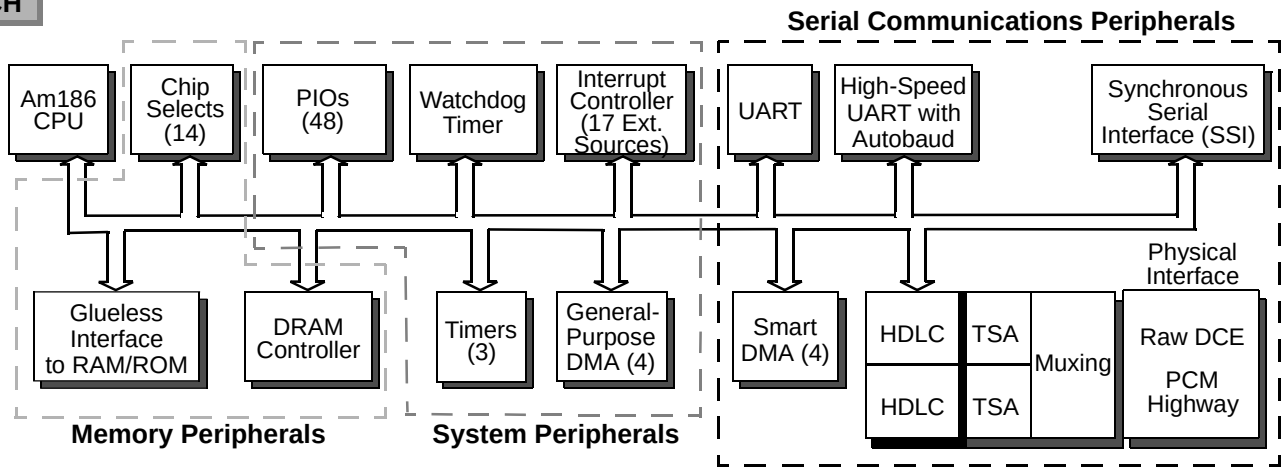
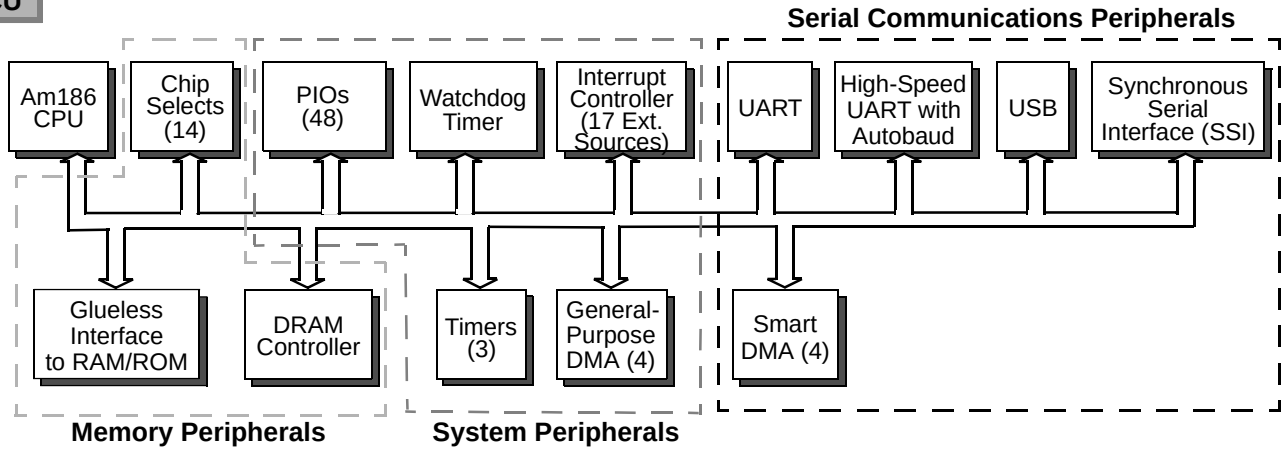


Figure 1-3 Am186CU USB Microcontroller Block Diagram

CU



1.4 ARCHITECTURAL OVERVIEW

The architectural goal of the Am186CC/CH/CU microcontrollers is to provide comprehensive communications features on a processor running the widely-known x86 instruction set. These microcontrollers combine communications peripherals with the Am186 embedded CPU, resulting in highly integrated microcontrollers that provide system-cost and performance advantages for a wide range of communications applications.

The following sections provide an overview of the features of the microcontrollers. The chapter number in parenthesis indicates where that feature is discussed in detail.

1.4.1 Am186 Embedded CPU (Chapter 2)

All members of the Am186 family, including the Am186CC/CH/CU microcontrollers, are compatible with the original industry-standard 186 parts, and build on the same core set of internal processor registers, instructions, and addressing modes. This chapter also describes the memory organization, address generation, I/O space, peripheral control block, segments, and data types.

1.4.2 Serial Communications Support (Chapter 12)

CC The Am186CC microcontroller supports eight serial interfaces: a USB peripheral controller, four HDLC channels, two UARTs, and an SSI. In addition, it supports the use of GCI and SmartDMA with the serial interfaces.

CH The Am186CH HDLC microcontroller supports five serial interfaces: two HDLC channels, two UARTs, and an SSI. In addition, it supports the use of SmartDMA with the serial interfaces.

CU The Am186CU USB microcontroller supports four serial interfaces: a USB peripheral controller, two UARTs, and an SSI. In addition, it supports the use of SmartDMA with the serial interfaces.

For an overview of the serial communications features, see Chapter 12, “Serial Communications Overview.”

1.4.2.1 Universal Serial Bus (Chapter 18)

CC **CU** The Am186CC and Am186CU microcontrollers each include a highly flexible integrated USB peripheral controller that lets designers implement a variety of microcontroller-based USB peripheral devices for telephony, audio, and other high-end applications. This integrated USB peripheral controller can provide a significant system-cost reduction compared to platforms that require a separate USB peripheral controller.

The Am186CC and Am186CU microcontrollers act as USB peripheral devices. The USB is a half-duplex, master/slave, polled bus. In other words, the microcontroller speaks on the USB only in response to a request from the USB host, usually a personal computer. There can be only one speaker on the USB at a time.

The USB controller does not support USB host or hub functions. However, the Am186CC and Am186CU microcontrollers can be used to implement USB peripheral functions in a device that also contains separate USB hub circuitry.

Use these microcontrollers in self-powered USB peripherals that use the full-speed signalling rate of 12 Mbit/s; they do not support the USB low-speed rate of 1.5 Mbit/s. Each microcontroller includes an integrated USB transceiver to minimize system device count and cost, but an external transceiver can be used instead, if necessary.

In addition, the USB peripheral controller supports the following:

- An unlimited number of device descriptors.
- A total of six endpoints: one control endpoint; one interrupt endpoint; and four data endpoints that can be configured as control, interrupt, bulk, or isochronous. The interrupt, bulk, and isochronous endpoints can be configured for the IN or OUT direction.
- Two of the data endpoints have 16-byte FIFOs and two have 64-byte FIFOs.
- Fully integrated differential driver, which directly supports the USB interface.
- Specialized hardware, which supports adaptive isochronous data streams and automatically synchronizes with HDLC data streams.
- General-purpose DMA and SmartDMA channels.

1.4.2.2 HDLC Channels (Chapter 15) and TSAs (Chapter 16)

CC **CH** The Am186CC microcontroller provides four HDLC channels and the Am186CH HDLC microcontroller provides two HDLC channels. These channels support the HDLC, SDLC, LAP-B, LAP-D, PPP, and V.120 protocols. The HDLC channels can also be used in transparent mode to support V.110. Each HDLC channel can connect to an external serial interface directly (non-multiplexed mode), or can pass through a TSA (multiplexed mode). The flexible interface multiplexing arrangement allows each HDLC channel to have its own external interface, to share a common PCM highway or other time division multiplexed (TDM) bus with the other channels, or to work in some combination.

CC The Am186CC microcontroller supports raw DCE, PCM highway, and GCI interfaces.

CH The Am186CH HDLC microcontroller supports raw DCE and PCM highway interfaces.

Each HDLC channel's independent TSA allows it to extract a subset of data from a TDM bus. It can isolate the entire frame or as little as one bit per frame. The channel's 12-bit counter defines the start/stop bit times as the number of bits after frame synchronization. The time slot can be an arbitrary number of bits up to 4096 bits. Start bit and stop bit times identify the isolated portion of the TDM frame. Support of less than eight bits per time slot, or *bit slotting*, allows isolation of from one to eight bits in a single time slot, providing a convenient way to work with D-channel data. Each TDM bus can have up to 512 8-bit time slots. Support of these features allows interoperation with PCM highway, E1, IOM-2, T1, and other TDM buses.

To make the Am186CC and Am186CH microcontrollers attractive devices for use where general HDLC capability is required, the HDLC channels support the following features:

- Clear-to-Send (CTS) and Ready-to-Receive (RTR) hardware handshaking and auto-enable operation
- Collision detection for multidrop applications
- Transparency mode
- Address comparison on receive
- Flag or mark idle operation
- Two dedicated buffer descriptor ring SmartDMA channels per HDLC channel
- Transmit and receive FIFOs
- Full-duplex data transfer

CC CH Each TSA channel can support a burst data rate to or from the HDLC of up to 10 Mbit/s in both raw DCE and PCM highway modes.

CC In addition to raw DCE and PCM highway, the Am186CC microcontroller can share its GCI interface with up to two other channels. In GCI mode, the Am186CC microcontroller's TSA channels can support a burst data rate to or from the HDLC of up to 768 Kbit/s.

Total system data throughput is highly dependent on the amount of per-packet and per-byte CPU processing, the rate at which packets are being sent, and other CPU activity.

When combined with the TSAs, the HDLC channels are suitable for use in a wide variety of applications such as ISDN basic rate interface (BRI) and primary rate interface (PRI) B and D channels, PCM highway, X.25, Frame Relay, and other proprietary Wide Area Network (WAN) connections.

1.4.2.3 General Circuit Interface (Chapter 17)

CC The GCI is an interface specification developed jointly by Alcatel, Italtel, GPT, and Siemens. This specification defines an industry-standard serial bus for interconnecting telecommunications integrated circuits. The standard covers linecard, NT1, and terminal architectures for ISDN applications. The Am186CC microcontroller supports the terminal version of GCI.

The GCI interface provides a glueless connection between the Am186CC microcontroller and GCI/IOM-2 based ISDN transceiver devices, such as the AMD Am79C30 or Am79C32. The GCI interface provides a 4-pin connection to the transceiver device. The Am186CC microcontroller also allows conversion of the GCI clock and GCI frame sync into a format usable by PCM codecs, allowing the use of PCM codecs directly with GCI/IOM-2 transceivers. Additional GCI features include slave mode with pin reversal, Terminal Interchip Communication (TIC) bus support for D channel arbitration and collision detection, and support for one Monitor and two Command/Indicate channels.

1.4.2.4 SmartDMA Channels (Chapter 8)

Each of the Am186CC/CH/CU microcontrollers contain both SmartDMA channels and general-purpose DMA channels (see "General-Purpose DMA Channels (Chapter 8)" on page 1-10). The SmartDMA channels provide a faster method for moving data between peripherals and memory with lower CPU utilization. SmartDMA transmits and receives data across multiple memory buffers and a sophisticated buffer-chaining mechanism. These channels work in pairs: transmitter and receiver. The transmit channels can transfer data only from memory to a peripheral; the receive channels can transfer data only from a peripheral to memory.

CC The Am186CC microcontroller provides a total of 12 DMA channels: eight SmartDMA channels and four general-purpose DMA channels. Four of the SmartDMA channels (two pairs) are dedicated for use with two of the on-board HDLC channels. The remaining four SmartDMA channels (two pairs) can support either the third or fourth HDLC channel or Universal Serial Bus (USB) endpoints A, B, C, or D.

CH The Am186CH HDLC microcontroller provides a total of eight DMA channels: four SmartDMA channels to support the two HDLC channels and four general-purpose DMA channels.

CU The Am186CU USB microcontroller provides a total of eight DMA channels: four SmartDMA channels to support USB endpoints A–D and four general-purpose DMA channels.

1.4.2.5 Asynchronous Serial Ports (Chapter 13)

The Am186CC/CH/CU microcontrollers each have two asynchronous serial ports that provide full-duplex, bidirectional data transfer with speeds up to 460 Kbaud. One port is a high-speed UART with transmit and receive FIFOs, special character matching, and automatic baud rate detection, suitable for implementation of a Hayes-compatible modem interface to a host PC. There is also a lower speed UART, which typically is used for a low baud rate system configuration port or debug port. Each of these UARTs can derive its baud rate from the CPU clock or from a separate baud rate generator clock input. Both UARTs support 7-, 8-, or 9-bit data transfers; address bit generation and detection in 7- or 8-bit frames; one or two stop bits; even, odd, or no parity; break generation and detection; hardware flow control; and DMA to and from the serial ports using the general-purpose DMA channels (see “General-Purpose DMA Channels (Chapter 8)” on page 1-10).

1.4.2.6 Synchronous Serial Port (Chapter 14)

The Am186CC/CH/CU microcontrollers each include one SSI port, which provides a half-duplex, bidirectional communications interface between the microcontroller and other system components. Typical applications use this interface to monitor the status of other system devices and to configure these devices under software control. In a communications application, these devices could be system components such as audio coder-decoders (codecs), line interface units, and transceivers. The SSI supports data transfer speeds of up to 25 Mbit/s with a 50-MHz CPU clock.

The SSI port operates as an interface master with the other attached devices acting as slave devices. Using this protocol, the microcontroller sends a command byte to the attached device, and then follows that byte with either a read or write of a byte of data.

The SSI port consists of three I/O pins: an enable (SDEN), a clock (SCLK), and a bidirectional data pin (SDATA). SDEN can be used directly as an enable for a single attached device. When more than one device requires control through the SSI, use PIOs to provide enable pins for those devices.

The SSI port is, in general, software compatible with the Am186EM SSI port. Some additional features have been added to the Am186CC/CH/CU microcontrollers’ SSI implementation. In addition, the microcontroller can select the polarity of the SCLK and SDEN pins, as well as the shift order of bits on the SDATA pin (least-significant-bit first versus most-significant-bit first). The SSI port also offers a programmable clock divisor (dividing the clock from 2 to 256 in power of 2 increments), a bidirectional transmit/receive shift register, and direct connection to AMD Subscriber Line Audio-processing Circuit (SLAC™) devices.

1.4.3 System Peripherals

The Am186CC/CH/CU microcontrollers provide several additional system peripherals to simplify incorporation of the microcontroller into an embedded application.

1.4.3.1 Interrupt Controller (Chapter 7)

The Am186CC/CH/CU microcontrollers each feature an interrupt controller, which arranges up to 36 maskable interrupt requests by priority and presents them one at a time to the CPU.

The interrupt controller supports the maskable interrupt sources through the use of 15 channels. To make this possible, most interrupt channels support multiple interrupt sources. These channels are programmable to support the external interrupt pins or various peripheral devices that can be configured to generate interrupts. The maskable interrupt sources include 17 external sources plus a number of internal sources.

CC The Am186CC microcontroller has 19 internal maskable interrupt sources.

CH The Am186CH HDLC microcontroller has 14 internal maskable interrupt sources.

CU The Am186CU USB microcontroller has 13 internal maskable interrupt sources.

In addition to interrupts managed by the interrupt controller, each microcontroller supports eight nonmaskable interrupts—an external or internal nonmaskable interrupt (NMI), a trace interrupt, and software interrupts and exceptions.

1.4.3.2 General-Purpose DMA Channels (Chapter 8)

Four of the DMA channels in each of the Am186CC/CH/CU microcontrollers are general purpose. The general-purpose DMA channels support data transfer between memory and I/O spaces (i.e., memory-to-I/O or I/O-to-memory) or within the same space (i.e., memory-to-memory or I/O-to-I/O). In addition, the microcontrollers support data transfer between peripherals and memory or I/O. Internal peripherals that support general-purpose DMA are Timer 2, which can provide a periodic internal DMA request, and the two asynchronous serial ports (UART and High-Speed UART).

External peripherals support DMA transfers through the external DMA request pins (DRQ1–DRQ0). Each general-purpose channel accepts a DMA request from one of three sources: the DMA request signals (DRQ1–DRQ0), Timer 2, or the UARTs. (Note that Timer 2 acts only as a DMA request source; no data is transferred to or from Timer 2.) In addition to the general-purpose channels, the microcontrollers provide SmartDMA channels (see “SmartDMA Channels (Chapter 8)” on page 1-8).

CC CU The USB peripheral controller in the Am186CC and Am186CU microcontrollers can also request a general-purpose DMA transfer.

1.4.3.3 Programmable I/O Signals (Chapter 9)

Each of the Am186CC/CH/CU microcontrollers provides 48 user-programmable input/output signals (PIOs). In the Am186CC microcontroller, each of these signals shares a pin with at least one alternate function. In the Am186CH and Am186CU microcontrollers, most but not all of the PIOs share a pin with alternate functions. If an application does not need the alternate function, the associated PIO can be used by programming the PIO registers.

If a pin is enabled to function as a PIO signal, the alternate function is disabled and does not affect the pin. A PIO signal can operate as an input or output, with or without internal pullup or pulldown resistors (whether the resistors are pullup or pulldown depends on the pin configuration and is not user-configurable), or as an open-drain output. In addition to the three PIOs multiplexed with interrupt signals, eight other PIOs can be configured as external interrupt sources. For more information about PIOs as interrupt sources, see Chapter 7, “Interrupts.”

1.4.3.4 Programmable Timers (Chapter 10)

Each of the Am186CC/CH/CU microcontrollers has three 16-bit programmable timers. Timers 0 and 1 are highly versatile and are each connected to two external pins (each one has an input and an output). These two timers can count or time external events that drive the timer input pins. Timers 0 and 1 can also generate nonrepetitive or variable-duty-cycle waveforms on the timer output pins.

Timer 2 is not connected to any external pins. Software can use it to generate interrupts, or poll it for real-time coding and time-delay applications. Software can also use Timer 2 as a prescaler to Timer 0 and Timer 1, or as a DMA request source (see Chapter 8, “DMA Controller”).

The source clock for Timer 2 is one-fourth of the CPU clock frequency. Timers 0 and 1 can use every fourth cycle of the CPU clock as a source, or they can be driven from the timer

input pins. When driven from a timer input pin, the timer is counting the “event” of an input transition.

The microcontroller also provides a pulse width demodulation (PWD) option so that a toggling input signal’s Low state and High state durations can be measured.

1.4.3.5 Hardware Watchdog Timer (Chapter 11)

Each of the Am186CC/CH/CU microcontrollers provides a full-featured watchdog timer, which includes the ability to generate NMIs, reset the microcontroller (except for pinstraps), and reset the system (assert $\overline{\text{RESOUT}}$) when the time-out value is reached. The time-out value is programmable and ranges from 2^{10} to 2^{26} processor clocks.

The watchdog timer is used to regain control when a system has failed due to a software error or the failure of an external device to respond in the expected way. Software errors can sometimes be resolved by recapturing control of the execution sequence through a watchdog-timer-generated NMI. When an external device fails to respond, or responds incorrectly, it may be necessary to reset the microcontroller or the entire system, including external devices. The watchdog timer provides the flexibility to support both NMI and reset generation.

1.4.4 Memory and Peripheral Interface

Each of the Am186CC/CH/CU microcontrollers includes the following memory and peripheral interfaces.

1.4.4.1 System Interfaces and Clock Control (Chapter 3)

The microcontroller includes a bus interface to control all accesses to the peripheral control block (PCB), memory-mapped and I/O-mapped external peripherals, and memory devices. The bus interface accesses the internal peripherals through the PCB. The bus interface features programmable bus sizing, separate byte/write enables, and the option to boot from an 8-bit or 16-bit device.

The industry-standard 80C186 and 80C188 microcontrollers use a multiplexed address and data (AD) bus. The address is present on the AD bus only during the t_1 clock phase. The microcontrollers also provide the multiplexed AD bus and in addition, provide a nonmultiplexed address (A) bus. The A bus provides an address to the system for the complete bus cycle (t_1–t_4).

The microcontroller operates with a V_{CC} of 3.3 ± 0.3 V. All the digital signals are capable of 5-V-tolerant I/O operation.

The processor supports clock rates from 25 MHz to 50 MHz. Commercial and industrial temperature ratings are available. The CPU can run in 1x, 2x, or 4x mode.

CC **CU** The Am186CC and Am186CU microcontrollers provide separate crystal oscillator inputs for the USB peripheral controller and the CPU. Flexibility is provided to run the entire device from a 12- or 24-MHz crystal when the USB is in use. The CPU can run in 1x, 2x, or 4x mode; the USB can run in 2x or 4x mode.

1.4.4.2 Dynamic Random Access Memory Support (Chapter 6)

To support DRAM, the microcontroller has a fully integrated DRAM controller that provides a glueless interface to 25-ns to 70-ns Extended Data Out (EDO) DRAM (EDO DRAM is sometimes called Hyper-Page Mode DRAM). The microcontroller can access up to two banks of 4-Mbit (256 Kbit x 16 bit) DRAM. The microcontroller does not support Page Mode DRAM, Fast Page Mode DRAM, Asymmetrical DRAM, or 8-bit wide DRAM. The microcontroller provides zero-wait state operation at up to 50 MHz with 40-ns DRAM. This

capability allows designs requiring larger amounts of memory to save system cost over SRAM designs by taking advantage of low DRAM costs.

The DRAM interface uses various chip select pins to implement the $\overline{RAS}/\overline{CAS}$ interface required by DRAMs. The microcontroller's DRAM controller drives the $\overline{RAS}/\overline{CAS}$ interface appropriately during both normal memory accesses and refresh. The microcontroller generates all required signals and does not require external logic.

The DRAM multiplexed address pins connect to the microcontroller's odd address pins, starting with A1 on the microcontroller connecting to MA0 on the DRAM. The correct row and column address are generated on these odd address pins during a DRAM access.

The RAS pins are multiplexed with $\overline{LCS}$ or $\overline{MCS3}$, allowing a DRAM bank to be present in either high or low memory space. $\overline{MCS2}$ and $\overline{MCS1}$ function as the lower and upper $\overline{CAS}$ pins, respectively, and define which byte of data in a 16-bit DRAM is being accessed.

The microcontroller supports the most common DRAM refresh option, CAS-Before-RAS. All refresh cycles contain three wait states to support the DRAMs at various frequencies. The DRAM controller never performs a burst access. All accesses are single accesses to DRAM. If the $\overline{PCS}$ chip selects are decoded to be in the DRAM address range, $\overline{PCS}$ accesses take precedence over the DRAM.

1.4.4.3 Chip Selects (Chapter 5)

The microcontroller provides six chip select outputs for use with memory devices and eight more chip selects for use with peripherals in either memory or I/O space. The six memory chip selects can address three memory ranges. Each peripheral chip select addresses a 256-byte block offset from a programmable base address.

The microcontroller can be programmed to sense a ready signal for each of the peripheral or memory chip select lines. A bit in each chip select control register determines whether the external ready signal is required or ignored.

In addition, the chip selects can control the number of wait states inserted in the bus cycle. Although most memory and peripheral devices can be accessed with three or less wait states, some slower devices cannot. This feature allows devices to use wait states to slow down the bus.

The chip select lines are active for all memory and I/O cycles in their programmed areas, whether the cycles are generated by the CPU or by the integrated DMA unit.

General enhancements over the original 80C186 include bus mastering (three-state) support for all chip selects, and activation only when the associated register is written (not when it is read).

1.4.5 In-Circuit Emulator Support (Chapter 4)

Because pins are an expensive resource, many play a dual role, and the programmer selects PIO operation or an alternate function. However, a pin configured to be a PIO may also be required for emulation support. Therefore, it is important that before a design is committed to hardware, a user contact potential emulator suppliers for a list of emulator pin requirements.

The Am186CC/CH/CU microcontrollers are designed to minimize conflicts. In most cases, pin conflict is avoided. For example, if the ALE signal is required for multiplex bus support, then it would not be programmed as PIO33. If the multiplexed AD bus is not used for address determination, then ALE can be programmed as a PIO pin.

1.5 APPLICATIONS

The Am186CC/CH/CU microcontrollers, with their integrated communications features, provide highly integrated, cost-effective solutions for a wide range of telecommunications and networking applications.

CC

- **ISDN Modems and Terminal Adapters:** Next-generation ISDN equipment requires USB (or High-speed UART capability), in addition to three channels of HDLC.

CC

- **Low-End Routers:** ISDN to Ethernet-based personal routers, often used for connections in Small Office/Home Office (SOHO) environments, require three channels of HDLC, as well as the high performance of a 16-bit controller.

CC CH

- **Linecard Applications:** Typically, linecards used in Central Offices (COs), PABX equipment, and other telephony applications require one or two channels of HDLC. Linecard manufacturers are moving to more lines per card for analog POTS as a means of cost reduction. This, and digital linecards for support of ISDN, often requires higher performance than existing 8-bit devices can offer. The Am186CC and Am186CH microcontrollers are ideal solutions for these applications because they integrate much of the necessary glue logic while providing higher performance.

CC CU

- **xDSL Applications:** Today's xDSL applications, such as high-speed ADSL modems, require data handling of 2 Mbit/s or greater and can take advantage of the USB interface for easy connectivity to the PC.

CC

- **Digital Corded Phones:** Typical digital telephone applications use up to three channels of HDLC and may use USB for merged PC telephony applications.

CC CH

- **Industrial Control:** Embedded x86 processors have long been used in the industrial control market. These applications often require a robust, high-performance processor solution with the capability to easily communicate with other parts of a system. The Am186CC and Am186CH microcontrollers provide numerous interfaces to achieve this communication, including the SSI interface, high-speed UART, and the HDLC channels, which also can be used to create a multidrop backplane.

CC CU

- **USB Peripheral Devices:** These devices will become more common as the PC market embraces the USB protocol specified by the Microsoft™ Windows 98 operating system. In addition to implementing communications device class systems such as an ISDN terminal adapter, the USB peripheral controller makes the Am186CC and Am186CU USB microcontrollers suitable for certain PC desktop applications such as a USB camera interface, ink-jet printers, and scanners.

- **General Communications Applications:** The Am186CC/CH/CU microcontrollers will also find a home in general embedded applications, because many devices will incorporate communications capability in the future. These microcontrollers are especially attractive for 186 designs adding HDLC, USB, or both.

The block diagrams beginning on page 1-5 show some typical designs. Figure 1-4 shows an ISDN terminal adapter. Figure 1-5 shows a low-end router. Figure 1-6 shows a 32-channel linecard.

The ISDN terminal adapter features an S/T or U interface and either a high-speed UART or USB connection for attaching the modem to the PC.

The ISDN-to-Ethernet low-end router features an S/T or U interface, two POTS lines, and a 10-Mbit/s connection to the PC.

The 32-channel linecard design demonstrates a linecard application where 32 lines terminate on the linecard.

Figure 1-4 ISDN Terminal Adapter

CC

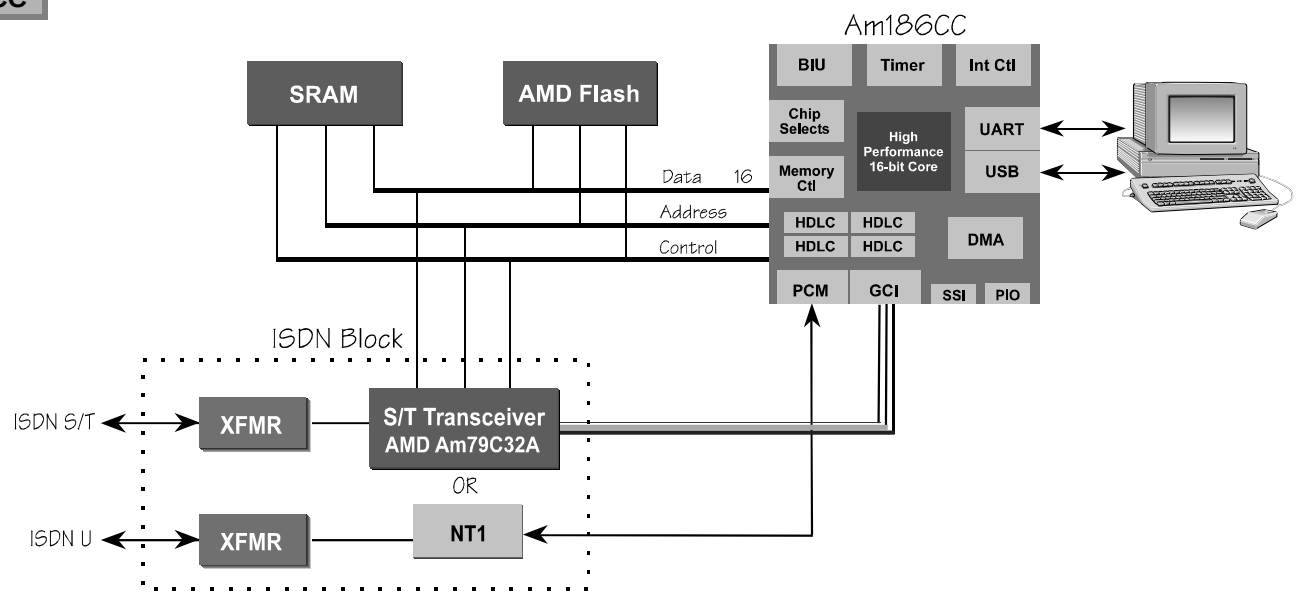


Figure 1-5 ISDN-to-Ethernet Low-End Router

CC

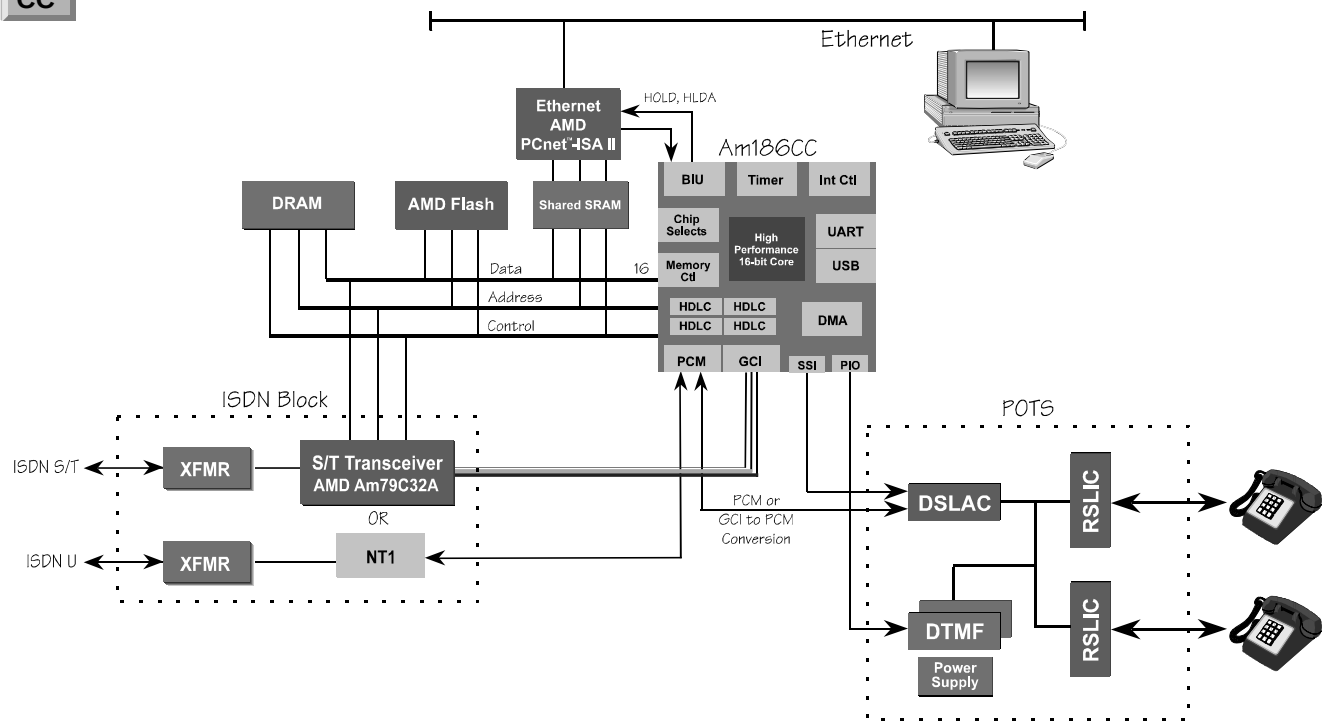
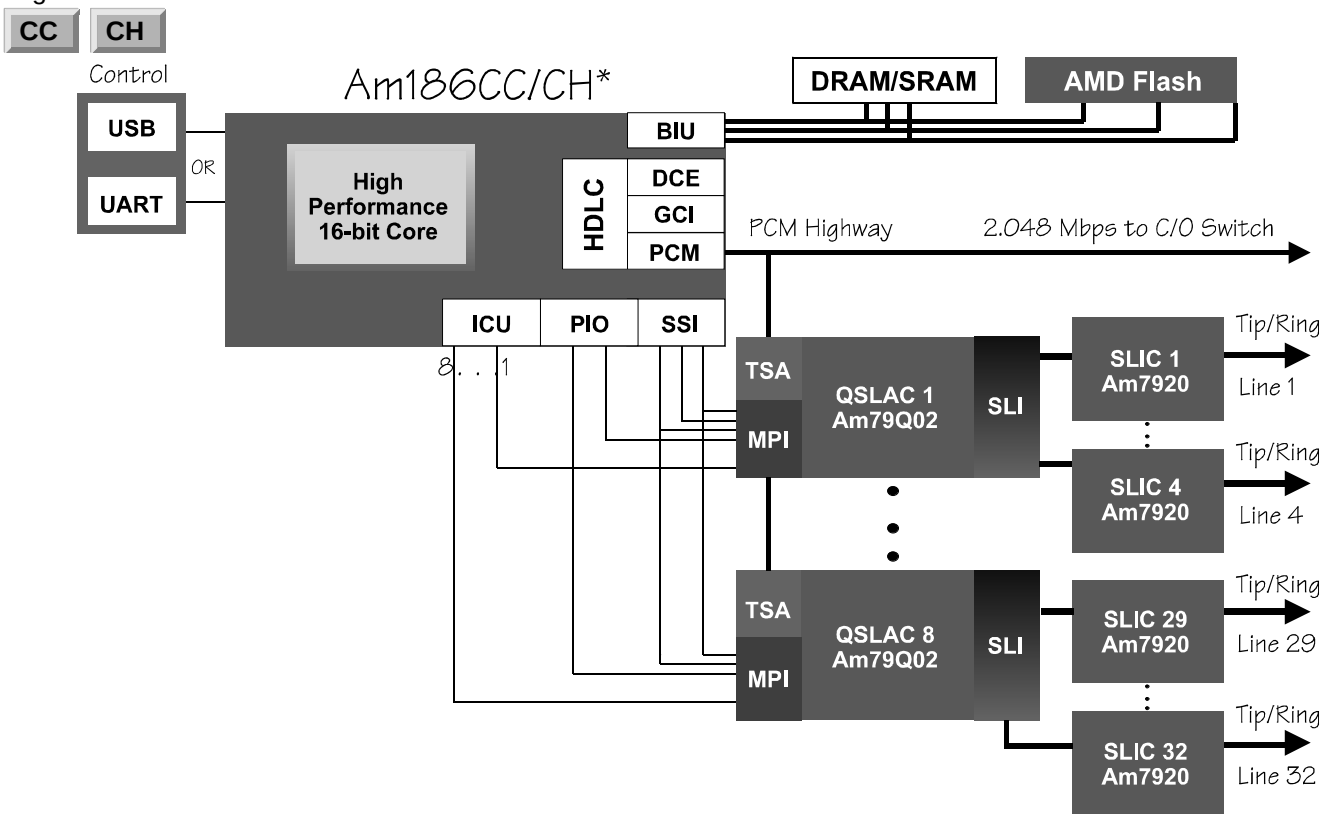


Figure 1-6 32-Channel Linecard



* Am186CH controller does not have USB peripheral controller or a GCI interface.

CHAPTER

2 CONFIGURATION BASICS

2.1 OVERVIEW

All members of the Am186 family, including the Am186CC/CH/CU microcontrollers, build on the same core set of internal processor registers, instructions, and addressing modes. All members are compatible with the original industry-standard 186 parts.

This chapter provides basic information about configuring the microcontrollers, including discussions of the registers, memory organization, address generation, I/O space, peripheral control block, instruction set, segments, data types, and addressing modes.

2.2 REGISTER SET

The microcontroller contains hundreds of configuration and control registers, both internal and external to the processor. The instruction set contains instructions to access the internal processor registers directly. Peripheral registers are external to the processor. However, because the processor treats these peripheral registers either like memory or like I/O, instructions with memory or I/O operands can access peripheral registers. This section briefly describes these processor and peripheral registers. For detailed information on the microcontroller peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

2.2.1 Processor Registers

The base architecture of the Am186CC/CH/CU microcontrollers has 14 registers, like all members of the Am186 family. Table 2-1 lists these registers.

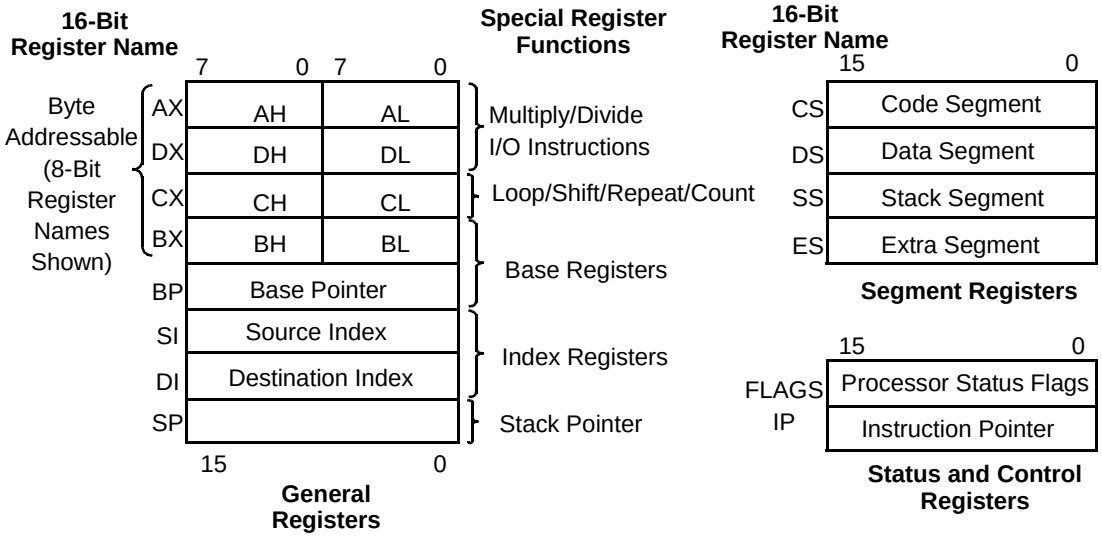
Table 2-1 Internal Processor Registers

Register Name	Register Mnemonic	Function
General-Purpose Arithmetic and Logical Operand	AX	Accumulator
	BX	Base
	CX	Count
	DX	Data
Base Pointer	BP	Stack segment, points to bottom of the stack frame
Source Index	SI	Data movement and string instructions
Destination Index	DI	
Stack Pointer	SP	Stack segment, points to top of stack
Code Segment	CS	Points to the current code segment, which contains instructions to be fetched
Data Segment	DS	Selects memory segment addressable for data
Stack Segment	SS	Selects memory segment addressable for the stack
Extra Segment	ES	Selects memory segment addressable for data
Processor Status Flags	FLAGS	Contains status and control flag bits
Instruction Pointer	IP	Contains offset address of next instruction to be executed

These registers are grouped into the following categories:

- **General-Purpose registers:** Eight 16-bit general-purpose registers support arithmetic and logical operands. Four of these (AX, BX, CX, and DX) also operate as pairs of separate 8-bit registers (AH, AL, BH, BL, CH, CL, DH, and DL). The Destination Index (DI) and Source Index (SI) general-purpose registers support data movement and string instructions. The Base Pointer (BP) and Stack Pointer (SP) general-purpose registers point to the bottom and to the top of the stack frame (in the stack segment), respectively.
  - **Base and Index registers:** Four of the general-purpose registers (BP, BX, DI, and SI) also support the determination offset addresses of operands in memory. These registers can contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.
  - **Stack Pointer register:** All stack operations (POP, POPA, POPF, PUSH, PUSHA, PUSHF) utilize the stack pointer. The Stack Pointer (SP) register is always offset from the Stack Segment (SS) register, and no segment override is allowed.
- **Segment registers:** Four 16-bit special-purpose registers (CS, DS, ES, and SS) select, at any given time, the segments of memory that are immediately addressable for code (CS), data (DS and ES), and stack (SS) memory.
- **Status and Control registers:** Two 16-bit special-purpose registers record or alter certain aspects of the processor state—the Instruction Pointer (IP) register contains the offset address of the next sequential instruction to be executed, and the Processor Status Flags (FLAGS) register contains status and control flag bits (see Figure 2-1 and Figure 2-2).

Figure 2-1 Register Set

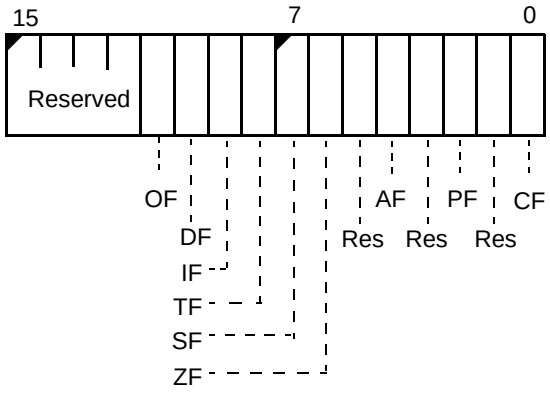


2.2.2 Processor Status Flags Register

The 16-bit Processor Status Flags register, illustrated in Figure 2-2, records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the microcontroller within a given operating mode (bits 8, 9, and 10).

After the processor executes an instruction, the value of the flags can be set to 1, cleared or reset to 0, unchanged, or undefined. The term *undefined* means that the flag value prior to the execution of the instruction is not preserved, and that after the instruction is executed, the value of the flag cannot be predicted.

Figure 2-2 Processor Status Flags Register



Bits 15–12, Reserved

Bit 11, Overflow Flag (OF): Set if the signed result cannot be expressed within the number of bits in the destination operand; otherwise cleared.

Bit 10, Direction Flag (DF): When set, causes string instructions to auto-decrement the appropriate index registers. When cleared, causes auto-increment.

Bit 9, Interrupt-Enable Flag (IF): When set, enables maskable interrupts to cause the CPU to transfer control to a location specified by an interrupt vector. This flag is cleared when the processor takes a hardware interrupt, or a trace interrupt, by using the CLI instruction. For more information about hardware and software interrupts, see Chapter 7, “Interrupts.”

Bit 8, Trace Flag (TF): When set, a trace interrupt occurs after instructions execute. TF is cleared by the trace interrupt after the processor status flags are pushed onto the stack. The trace service routine can continue tracing by popping the flags back with an interrupt return (IRET) instruction.

Bit 7, Sign Flag (SF): Set equal to high-order bit of result (0 if 0 or positive, 1 if negative).

Bit 6, Zero Flag (ZF): Set if result is 0; otherwise cleared.

Bit 5, Reserved

Bit 4, Auxiliary Carry (AF): Set on carry from or borrow to the low-order four bits of the AL general-purpose register; otherwise cleared.

Bit 3, Reserved

Bit 2, Parity Flag (PF): Set if low-order eight bits of result contain an even number of bits set to 1; otherwise cleared.

Bit 1, Reserved

Bit 0, Carry Flag (CF): Set on high-order bit carry or borrow; otherwise cleared.

2.2.3 Peripheral Registers

While the 186-legacy registers can be accessed directly through the 186 instructions, the peripheral registers must be accessed by using instruction operands that access memory or I/O space.

The address of each 16-bit read/write peripheral register is in the internal 1-Kbyte peripheral control block (PCB). Registers physically reside in the peripheral devices they control, but they are addressed through the PCB. This block resides either in memory or I/O space, at the location pointed to by the Peripheral Control Block Relocation (RELOC) register (see Table 2-2). Because the base address of the block can change, the address of each register is specified as an offset from the location pointed to by the RELOC register, rather than as an absolute address. To determine the absolute address of the register in memory or I/O space, add the offset to the base address. For a discussion of memory versus I/O space, see “Memory Organization and Address Generation” on page 2-5.

Note: *Accesses to the PCB should be performed by direct processor actions. The use of DMA to write or read from the PCB results in unpredictable behavior, except where explicit exception is made to support a peripheral function, such as the High-Speed UART transmit and receive data registers.*

Table 2-2 Configuration Register Summary

Offset	Register Mnemonic	Register Name	Description
3FEh	RELOC	Peripheral Control Block Relocation	Allows software to relocate the peripheral control block to start at any even 1024-byte location in either memory or I/O space.

The PCB base address can be set to any even 1-Kbyte boundary in memory or I/O space (i.e., the lower 10 bits of the base address must be 0). RELOC resides in the last register address of the PCB, at offset 03FEh. On an external or watchdog timer reset, the RELOC register value is set to 20FCh, which maps the PCB to start at FC00h in I/O space. This places the RELOC register at FFFEh. Appendix A, “Register Summary,” provides a summary of PCB registers in offset order, including default address locations. For a complete description of the RELOC register, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

A legacy feature on the Am186CC/CH/CU microcontrollers allows logical word writes to the PCB registers to be performed as byte writes on the external bus. These writes transfer 16 bits of data to the PCB register even if an 8-bit register is named in the instruction. For example, “out dx, al” writes the value of AX to the port address in DX. Reads to the PCB registers should always be done as word reads. This feature eliminates the need for an additional bus cycle when the same code is executed on an 8-bit Am188 device or when the PCB overlaps an 8-bit address space. Unaligned reads and writes to the PCB result in unpredictable behavior on the Am186CC/CH/CU microcontrollers.

Internal logic recognizes control block addresses and responds to bus cycles. During bus cycles to internal registers, the bus controller signals the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, and data lines are driven as in a normal bus cycle), but the data bus, SRDY, and ARDY are ignored.

Table 2-3 lists the peripheral registers by functional groupings, along with the address offset where the group begins. For detailed information about the peripheral registers, refer to the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 2-3 Peripheral Register Summary

Offset Range	Functional Block	User's Manual Chapter[1]
000h–0F0h[2] CC CH	High-level Data Link Control (HDLC)	Chapter 15
100h–13Ch	General-Purpose DMA	Chapter 8
140h–198h	SmartDMA	
1E0h–25Eh[3] CC CU	Universal Serial Bus (USB)	Chapter 18
260h–27Ch	High-Speed Asynchronous Serial Port (High-Speed UART)	Chapter 13
280h–28Eh	Asynchronous Serial Port (UART)	
2A0h–2BEh[4] CC	General Circuit Interface (GCI)	Chapter 17
2C0h–2DCh[2] CC CH	Time Slot Assigner (TSA)	Chapter 16
2F0h–2F8h	Synchronous Serial Interface (SSI)	Chapter 14
300h–338h	Interrupt Controller	Chapter 7
340h–354h	Programmable Timers	Chapter 10
3A0h–3A8h	Chip Selects	Chapter 5
3AAh–3ACh	DRAM	Chapter 6
3C0h–3DCh	Programmable I/O (PIO)	Chapter 9
3DEh	Reset Configuration	Chapter 3
3E0h	Watchdog Timer	Chapter 11
3F0h	System Configuration	Chapter 3
3F4h	Processor Release Level	
3FEh	Relocation	Chapter 2

Notes:

1. For a list of registers with their bit field names and offset addresses, see Appendix A, "Register Summary." The Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, describes these registers in detail. That manual presents the registers in order of offset, from lowest to highest.

2. Reserved in the Am186CU USB microcontroller.

3. Reserved in the Am186CH HDLC microcontroller.

4. Reserved in the Am186CH and Am186CU microcontrollers.

2.3 MEMORY ORGANIZATION AND ADDRESS GENERATION

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of 64K (2^{16}) 8-bit bytes and must begin on a 16-byte boundary. Memory is addressed using a two-component address that consists of a 16-bit segment value and a 16-bit offset. The *offset* is the number of bytes from the beginning of the *segment* (the segment address) to the data or instruction that is being accessed. This segment value and offset form the *logical* address, which is used by code.

The processor forms the *physical* address of the target location by taking the segment address, shifting it to the left 4 bits (multiplying by 16), and adding the result to the 16-bit offset. The resulting sum is the 20-bit address of the target data or instruction. This technique allows for a 1-Mbyte physical address size.

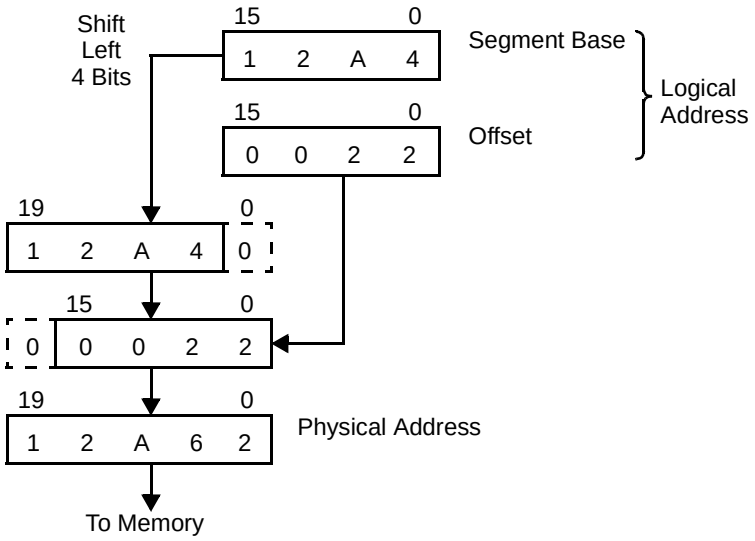
For example, if the segment register is loaded with 12A4h and the offset is 0022h, the resultant address is 12A62h, as illustrated in Figure 2-3. To find the result:

1. The segment register contains 12A4h.
2. Shift the segment register left 4 places to produce 12A40h.
3. The offset is 0022h.
4. Add the shifted segment address (12A40h) to the offset (00022h). The result is 12A62h.
5. This address is placed on the pins of the microcontroller.

All instructions that address operands in memory must specify (implicitly or explicitly) a 16-bit segment value and a 16-bit offset value. The 16-bit segment values are contained in one of the four internal segment registers (CS, DS, ES, and SS). For more information about calculating the offset value, see “Addressing Modes” on page 2-9. For more information about CS, DS, ES, and SS, see “Segments” on page 2-7.

In addition to 1 Mbyte of memory space, the Am186CC microcontroller provides 64K of I/O space (see Figure 2-4). Note that the processor reserves 00000h to 003FFh in memory for the interrupt vector table.

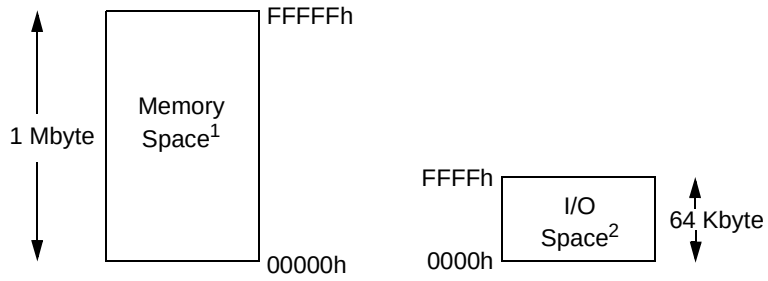
Figure 2-3 Physical Address Generation



2.4 I/O SPACE

The I/O space consists of 64K 8-bit or 32K 16-bit ports. The IN, INS, OUT, and OUTS instructions address the I/O space with either an 8-bit port address specified in the instruction, or a 16-bit port address in the DX register. Eight-bit port addresses are zero-extended so that A15–A8 are Low. Note the processor reserves I/O port addresses 00F8h through 00FFh. The microcontroller provides specific instructions for addressing I/O space.

Figure 2-4 Memory and I/O Space



Notes:

1. 00000h–003FFh are reserved for the interrupt vector table.

2. 00F8h–00FFh are reserved.

2.5 INSTRUCTION SET

The instruction set for the Am186CC/CH/CU microcontrollers is identical to the 80C186/188 instruction set. An instruction can reference from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed on page 2-9. For instruction set details, see the *Am186 and Am188 Family Instruction Set Manual*, order #21267.

2.6 SEGMENTS

The microcontroller uses four segment registers:

1. **Data Segment (DS):** The processor assumes that all accesses to the program's variables are from the 64K space pointed to by the DS register. The data segment holds data, operands, and so on.
2. **Code Segment (CS):** This 64K space is the default location for all instructions. All code must be executed from the code segment.
3. **Stack Segment (SS):** The processor uses the SS register to perform operations that involve the stack, such as pushes and pops. The stack segment provides temporary storage space.
4. **Extra Segment (ES):** Typically, this segment supports large string operations and large data structures. Certain string instructions assume the extra segment as the segment portion of the address. By using a segment override, the extra segment can also support a spare data segment.

When a data movement instruction does not define a segment, the processor assumes a data segment. An instruction prefix can override the segment register. For speed and compact instruction encoding, the addressing mode implies the segment register used for physical address generation (see Table 2-4).

Table 2-4 Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Local Data	Data (DS)	All data references
Instructions	Code (CS)	Instructions (including immediate data)
Stack	Stack (SS)	All stack pushes and pops Any memory references that use the BP register
External Data (Global)	Extra (ES)	All string instruction references that use the DI register as an index

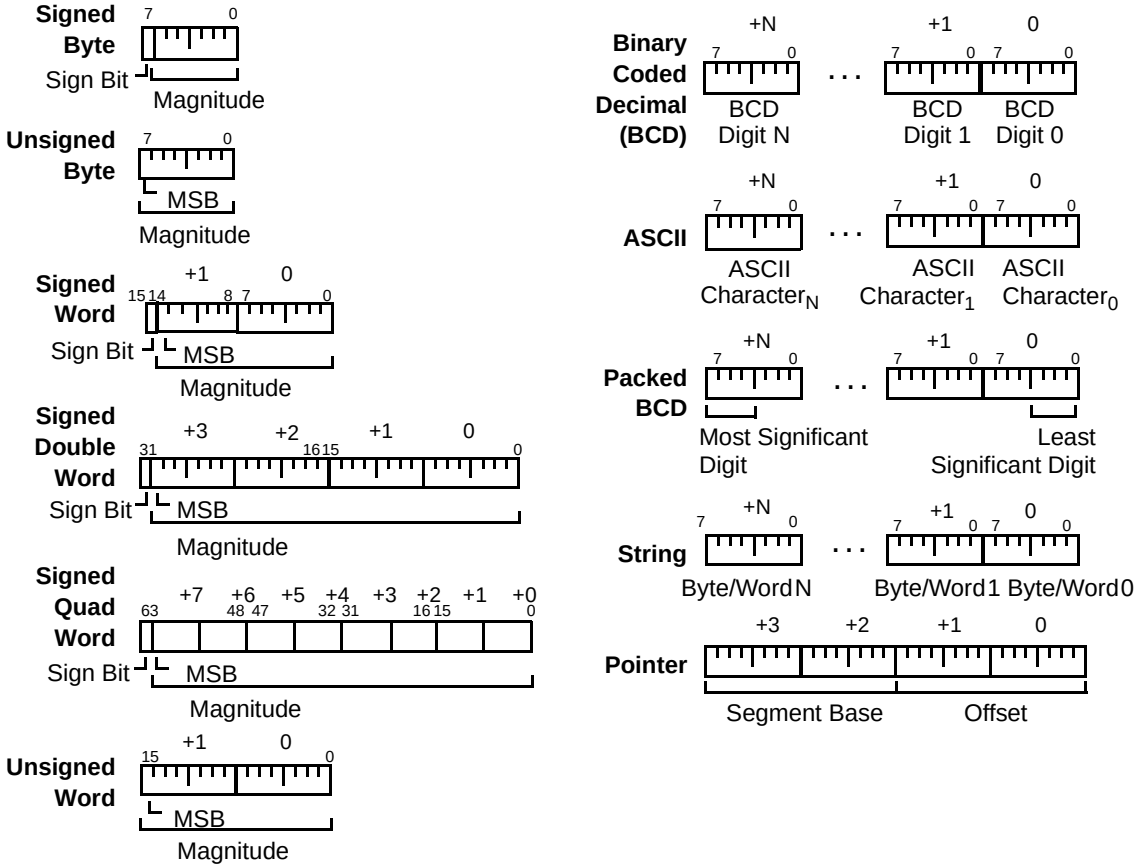
2.7 DATA TYPES

The Am186CC/CH/CU microcontrollers directly support the following data types:

- **Integer:** A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a two's complement representation.
- **Ordinal:** An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- **Double Word:** A signed binary numeric value contained in two sequential 16-bit addresses, or in a DX::AX register pair.
- **Quad Word:** A signed binary numeric value contained in four sequential 16-bit addresses.
- **BCD:** An unpacked byte representation of the decimal digits 0–9.
- **ASCII:** A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- **Packed BCD:** A packed byte representation of two decimal digits (0–9). Each nibble (four bits) of the byte contains one digit.
- **Pointer:** A 16-bit or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- **String:** A contiguous sequence of bytes or words. A string can contain from 1 byte up to 64 Kbytes.

In general, individual data elements must fit within defined segment limits. Figure 2-5 graphically represents the data types supported by the Am186CC/CH/CU microcontrollers.

Figure 2-5 Supported Data Types



2.8 ADDRESSING MODES

The Am186CC/CH/CU microcontrollers use eight categories of addressing modes to specify operands: two addressing modes for instructions that operate on register or immediate operands, and six modes that specify the location of an operand in a memory segment.

2.8.1 Register and Immediate Operands

- **Register Operand Mode:** The operand is in one of the 8-bit or 16-bit registers.
- **Immediate Operand Mode:** The operand is constant data included in the instruction.

2.8.2 Memory Operands

A memory-operand address consists of two 16-bit components: a segment value and an offset. The segment value is supplied by a 16-bit segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the *effective address*, is calculated by summing any combination of the following three address elements:

- **Displacement:** An 8-bit or 16-bit immediate value contained in the instruction.
- **Base:** The contents of either the BX or BP base register.
- **Index:** The contents of either the SI or DI index register.

Any carry from the 16-bit addition is ignored. Eight-bit displacements are sign-extended to 16-bit values.

Combinations of the above three address elements define the following six memory addressing modes (see Table 2-5 for examples).

- **Direct Mode:** The instruction contains the operand offset as an 8-bit or 16-bit displacement element.
- **Register Indirect Mode:** The operand offset is in one of the following registers: SI, DI, BX, or BP.
- **Based Mode:** The operand offset is the sum of an 8-bit or 16-bit displacement and the contents of a base register (BX or BP).
- **Indexed Mode:** The operand offset is the sum of an 8-bit or 16-bit displacement and the contents of an index register (SI or DI).
- **Based Indexed Mode:** The operand offset is the sum of the contents of a base register and an index register.
- **Based Indexed Mode with Displacement:** The operand offset is the sum of a base register's contents, an index register's contents, and an 8-bit or 16-bit displacement.

Table 2-5 Memory Addressing Mode Examples

Addressing Mode	Example
Direct	`mov ax, ds:4`
Register Indirect	`mov ax, [si]`
Based	`mov ax, [bx]4`
Indexed	`mov ax, [si]4`
Based Indexed	`mov ax, [si][bx]`
Based Indexed with Displacement	`mov ax, [si][bx]4`

CHAPTER

3 SYSTEM OVERVIEW

3.1 OVERVIEW

This chapter contains descriptions of the Am186CC/CH/CU microcontrollers' system configuration registers, initialization and processor reset, signals, bus interface, and clock control.

3.2 SYSTEM DESIGN

Table 3-1 shows the multiplexed signals and the trade-offs when selecting various functions. Table 3-2 on page 3-3 shows the multiplexed signal information ordered by PIO signal.

Table 3-1 Multiplexed Signal Trade-Offs

Desired Function			**Unavailable Functions**							
Inter-face	**Signal**	**Pin**	**Inter-face**	**Signal**	**Inter-face**	**Signal**	**Inter-face**	**Signal**	**Inter-face**	**Signal**
Memory										
SRAM	$\overline{LCS}$	131	**DRAM**	$\overline{RAS0}$	—	—	—	—	**PIO**	—
	$\overline{MCS1}$	127		$\overline{CAS1}$	—	—	—	—		—
	$\overline{MCS2}$	128		$\overline{CAS0}$	—	—	—	—		—
	$\overline{MCS3}$	129		$\overline{RAS1}$	—	—	—	—		PIO5
DRAM	$\overline{CAS0}$	128	**SRAM**	$\overline{MCS2}$	—	—	—	—	**PIO**	—
	$\overline{CAS1}$	127		$\overline{MCS1}$	—	—	—	—		—
	$\overline{RAS0}$	131		$\overline{LCS}$	—	—	—	—		—
	$\overline{RAS1}$	129		$\overline{MCS3}$	—	—	—	—		PIO5
Synchronous Communications Interfaces										
DCE Channel A CC CH	DCE_RXD_A	118	**PCM Channel A** CC CH	PCM_RXD_A	—	—	**GCI Channel A** CC	GCI_DD_A	**PIO**	—
	DCE_TXD_A	119		PCM_TXD_A	—	—		GCI_DU_A		—
	DCE_RCLK_A	117		PCM_CLK_A	—	—		GCI_DCL_A		—
	DCE_TCLK_A	116		PCM_FSC_A	—	—		GCI_FSC_A		—
	$\overline{DCE\_CTS\_A}$	123		$\overline{PCM\_TSC\_A}$	—	—		—		PIO17
	$\overline{DCE\_RTR\_A}$	122		—	—	—		—		PIO18
DCE Channel B CC CH	DCE_RXD_B	138	**PCM Channel B** CC CH	PCM_RXD_B	—	—	—	—	**PIO**	PIO36
	DCE_TXD_B	139		PCM_TXD_B	—	—	—	—		PIO37
	DCE_RCLK_B	135		PCM_CLK_B	—	—	—	—		PIO40
	DCE_TCLK_B	134		PCM_FSC_B	—	—	—	—		PIO41
	$\overline{DCE\_CTS\_B}$	137		$\overline{PCM\_TSC\_B}$	—	—	—	—		PIO38
	$\overline{DCE\_RTR\_B}$	136		—	—	—	—	—		PIO39
DCE Channel C CC	DCE_RXD_C	153	**PCM Channel C** CC	PCM_RXD_C	—	—	**GCI to PCM Con-version** CC	—	**PIO**	PIO42
	DCE_TXD_C	154		PCM_TXD_C	—	—		—		PIO43
	DCE_RCLK_C	150		PCM_CLK_C	—	—		PCM_CLK_C		PIO22
	DCE_TCLK_C	149		PCM_FSC_C	—	—		PCM_FSC_C		PIO23
	$\overline{DCE\_CTS\_C}$	152		$\overline{PCM\_TSC\_C}$	—	—		—		PIO44
	$\overline{DCE\_RTR\_C}$	151		—	—	—		—		PIO45

Table 3-1 Multiplexed Signal Trade-Offs (Continued)

Desired Function			Unavailable Functions							
Inter-face	Signal	Pin	Inter-face	Signal	Inter-face	Signal	Inter-face	Signal	Inter-face	Signal
DCE Channel D CC	DCE_RXD_D	158	PCM Channel D CC	PCM_RXD_D	Low-Speed UART	RXD_U	High-Speed UART (Flow Control)		PIO	PIO26
	DCE_TXD_D	159		PCM_TXD_D		TXD_U				PIO20
	DCE_RCLK_D	156		PCM_CLK_D		$\overline{\text{RTR\_U}}$				PIO25
	DCE_TCLK_D	157		PCM_FSC_D		$\overline{\text{CTS\_U}}$				PIO24
	$\overline{\text{DCE\_CTS\_D}}$	24		$\overline{\text{PCM\_TSC\_D}}$		—		$\overline{\text{CTS\_HU}}$		PIO46
	$\overline{\text{DCE\_RTR\_D}}$	23		—		—		$\overline{\text{RTR\_HU}}$		PIO47
PCM Channel A CC CH	PCM_RXD_A	118	DCE Channel A CC CH	DCE_RXD_A	—	—	GCI Channel A CC	GCI_DD_A	PIO	—
	PCM_TXD_A	119		DCE_TXD_A	—	—		GCI_DU_A		—
	PCM_CLK_A	117		DCE_RCLK_A	—	—		GCI_DCL_A		—
	PCM_FSC_A	116		DCE_TCLK_A	—	—		GCI_FSC_A		—
	$\overline{\text{PCM\_TSC\_A}}$	123		$\overline{\text{DCE\_CTS\_A}}$	—	—		—		PIO17
PCM Channel B CC CH	PCM_RXD_B	138	DCE Channel B CC CH	DCE_RXD_B	—	—	—	—	PIO	PIO36
	PCM_TXD_B	139		DCE_TXD_B	—	—	—	—		PIO37
	PCM_CLK_B	135		DCE_RCLK_B	—	—	—	—		PIO40
	PCM_FSC_B	134		DCE_TCLK_B	—	—	—	—		PIO41
	$\overline{\text{PCM\_TSC\_B}}$	137		$\overline{\text{DCE\_CTS\_B}}$	—	—	—	—		PIO38
PCM Channel C CC	PCM_RXD_C	153	DCE Channel C CC	DCE_RXD_C	—	—	GCI to PCM Conversion CC	—	PIO	PIO42
	PCM_TXD_C	154		DCE_TXD_C	—	—		—		PIO43
	PCM_CLK_C	150		DCE_RCLK_C	—	—		PCM_CLK_C		PIO22
	PCM_FSC_C	149		DCE_TCLK_C	—	—		PCM_FSC_C		PIO23
	$\overline{\text{PCM\_TSC\_C}}$	152		$\overline{\text{DCE\_CTS\_C}}$	—	—		—		PIO44
PCM Channel D CC	PCM_RXD_D	158	DCE Channel D CC	DCE_RXD_D	Low-Speed UART	RXD_U	High-Speed UART	—	PIO	PIO26
	PCM_TXD_D	159		DCE_TXD_D		TXD_U		—		PIO20
	PCM_CLK_D	156		DCE_RCLK_D		$\overline{\text{RTR\_U}}$		—		PIO25
	PCM_FSC_D	157		DCE_TCLK_D		$\overline{\text{CTS\_U}}$		—		PIO24
	$\overline{\text{PCM\_TSC\_D}}$	24		$\overline{\text{DCE\_CTS\_D}}$		—		$\overline{\text{CTS\_HU}}$		PIO46
Low-Speed UART	RXD_U	158	DCE Channel D CC	DCE_RXD_D	PCM Channel D CC	PCM_RXD_D	—	—	PIO	PIO26
	TXD_U	159		DCE_TXD_D		PCM_TXD_D	—	—		PIO20
	$\overline{\text{RTR\_U}}$	156		DCE_RCLK_D		PCM_CLK_D	—	—		PIO25
	$\overline{\text{CTS\_U}}$	157		DCE_TCLK_D		PCM_FSC_D	—	—		PIO24
High-Speed UART	RXD_HU	25	DCE Channel D CC	—	PCM Channel D CC	—	—	—	PIO	PIO16
	TXD_HU	26		—		—	—	—		—
	$\overline{\text{RTR\_HU}}$	23		$\overline{\text{DCE\_RTR\_D}}$		—	—	—		PIO47
	$\overline{\text{CTS\_HU}}$	24		$\overline{\text{DCE\_CTS\_D}}$		$\overline{\text{PCM\_TSC\_D}}$	—	—		PIO46
GCI Channel A CC	GCI_DD_A	118	DCE Channel A CC CH	DCE_RXD_A	PCM Channel A CC CH	PCM_RXD_A	—	—	PIO	—
	GCI_DU_A	119		DCE_TXD_A		PCM_TXD_A	—	—		—
	GCI_DCL_A	117		DCE_RCLK_A		PCM_CLK_A	—	—		—
	GCI_FSC_A	116		DCE_TCLK_A		PCM_FSC_A	—	—		—
GCI to PCM Conversion CC	PCM_CLK_C	150	DCE Channel C CC	DCE_RCLK_C	PCM Channel C CC	PCM_CLK_C	—	—	PIO	PIO22
	PCM_FSC_C	149		DCE_TCLK_C		PCM_FSC_C	—	—		PIO23

Table 3-1 Multiplexed Signal Trade-Offs (Continued)

Desired Function			Unavailable Functions							
Inter-face	Signal	Pin	Inter-face	Signal	Inter-face	Signal	Inter-face	Signal	Inter-face	Signal
Miscellaneous										
Bus Interface	$\overline{DEN}$	18	**Bus Interface**	$\overline{DS}$	—	—	—	—	**PIO**	PIO30
	$\overline{DS}$	18		$\overline{DEN}$	—	—	—	—		PIO30
Clocks	UCLK	22	**Clocks**	USBSOF CC CU	**Clocks**	USBSCI CC CU	—	—	**PIO**	PIO21
	USBSOF CC CU	22		UCLK		USBSCI CC CU	—	—		PIO21
	USBSCI CC CU	22		UCLK		USBSOF CC CU	—	—		PIO21

Table 3-2 Multiplexed Signal Trade-Offs Ordered by PIO

Desired Function		Unavailable Functions		
Signal	Pin	Signal	Signal	Signal
PIO0	144	TMRIN1	—	—
PIO1	143	TMROUT1	—	—
PIO2	10	$\overline{PCS5}$	—	—
PIO3	9	$\overline{PCS4}$	—	—
PIO4	126	$\overline{MCS0}$	—	—
PIO5	129	$\overline{MCS3}$	$\overline{RAS1}$	—
PIO6	147	INT8	PWD	—
PIO7	146	INT7	—	—
PIO8	14	ARDY	—	—
PIO9	124	DRQ0	—	—
PIO10	2	SDEN	—	—
PIO11	3	SCLK	—	—
PIO12	4	SDATA	—	—
PIO13	5	$\overline{PCS0}$	—	—
PIO14	6	$\overline{PCS1}$	—	—
PIO15	16	$\overline{WR}$	—	—
PIO16	25	RXD_HU	—	—
PIO17	123	$\overline{DCE\_CTS\_A}$ CC CH	$\overline{PCM\_TSC\_A}$ CC CH	—
PIO18	122	$\overline{DCE\_RTR\_A}$ CC CH	—	—
PIO19	145	INT6	—	—
PIO20	159	TXD_U	DCE_TXD_D CC	PCM_TXD_D CC
PIO21	22	UCLK	USBSOF CC CU	USBSCI CC CU
PIO22	150	DCE_RCLK_C CC	PCM_CLK_C CC	—
PIO23	149	DCE_TCLK_C CC	PCM_FSC_C CC	—
PIO24	157	$\overline{CTS\_U}$	DCE_TCLK_D CC	PCM_FSC_D CC
PIO25	156	$\overline{RTR\_U}$	DCE_RCLK_D CC	PCM_CLK_D CC
PIO26	158	RXD_U	DCE_RXD_D CC	PCM_RXD_D CC
PIO27	142	TMRIN0	—	—
PIO28	141	TMROUT0	—	—
PIO29	17	DT/$\overline{R}$	—	—

Table 3-2 Multiplexed Signal Trade-Offs Ordered by PIO (Continued)

Desired Function		Unavailable Functions		
Signal	**Pin**	**Signal**	**Signal**	**Signal**
PIO30	18	$\overline{\text{DEN}}$	$\overline{\text{DS}}$	—
PIO31	13	$\overline{\text{PCS7}}$	—	—
PIO32	11	$\overline{\text{PCS6}}$	—	—
PIO33	19	ALE	—	—
PIO34	20	$\overline{\text{BHE}}$	—	—
PIO35	15	SRDY	—	—
PIO36	138	DCE_RXD_B CC CH	PCM_RXD_B CC CH	—
PIO37	139	DCE_TXD_B CC CH	PCM_TXD_B CC CH	—
PIO38	137	$\overline{\text{DCE\_CTS\_B}}$ CC CH	$\overline{\text{PCM\_TSC\_B}}$ CC CH	—
PIO39	136	$\overline{\text{DCE\_RTR\_B}}$ CC CH	—	—
PIO40	135	DCE_RCLK_B CC CH	PCM_CLK_B CC CH	—
PIO41	134	DCE_TCLK_B CC CH	PCM_FSC_B CC CH	—
PIO42	153	DCE_RXD_C CC	PCM_RXD_C CC	—
PIO43	154	DCE_TXD_C CC	PCM_TXD_C CC	—
PIO44	152	$\overline{\text{DCE\_CTS\_C}}$ CC	$\overline{\text{PCM\_TSC\_C}}$ CC	—
PIO45	151	$\overline{\text{DCE\_RTR\_C}}$ CC	—	—
PIO46	24	$\overline{\text{CTS\_HU}}$	$\overline{\text{DCE\_CTS\_D}}$ CC	$\overline{\text{PCM\_TSC\_D}}$ CC
PIO47	23	$\overline{\text{RTR\_HU}}$	$\overline{\text{DCE\_RTR\_D}}$ CC	—

3.3 SYSTEM CONFIGURATION

Table 3-3 lists the registers used by the Am186CC/CH/CU microcontrollers for system configuration. Appendix A summarizes the bits in all of the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 3-3 System Configuration Register Summary

Offset	Register Mnemonic	Register Name	Description
3DEh	RESCON	Reset Configuration	Provides a way to make design-specific hardware configuration information available to software.
3F0h	SYSCON	System Configuration	Contains system-wide configuration bits which affect the operation on a global basis.
3F4h	PRL	Processor Revision Level	Contains the specific release level of the processor.

When $\overline{\text{RES}}$ is asserted, the Reset Configuration (RESCON) register is set to the value found on AD15–AD0. There is a one-to-one correspondence between address/data bus signals and the RESCON register's bits during reset (AD15 corresponds to bit 15 of the RESCON register, and so on). When $\overline{\text{RES}}$ is deasserted, the RESCON register holds its value. Software can read this value to determine the configuration information. For more information, see "Initialization and Reset" on page 3-5.

The System Configuration (SYSCON) register is typically written once to establish the proper modes of operation based on the system in which the part is operating. This register performs the following functions:

- Enables the data strobe timings on the $\overline{\text{DEN}}$ pin. When the DSDEN bit (bit 13) is set to 1, data strobe bus mode is enabled, and the $\overline{\text{DS}}$ timing for reads and writes is identical to the normal read cycle $\overline{\text{DEN}}$ timing. When the DSDEN bit is cleared to 0, the $\overline{\text{DEN}}$ timing for both reads and writes is normal (i.e., like the original 80C186). The $\overline{\text{DEN}}$ pin is renamed $\overline{\text{DS}}$ in data strobe bus mode. For more information, see “Bus Interface” on page 3-28.
- Enables Pulse Width Demodulation (PWD) mode. For more information about PWD mode, see Chapter 10, “Programmable Timers.”
- Disables memory and I/O addresses on the AD15–AD0 bus. For more information, see “Bus Interface” on page 3-28.
- Configures HDLC Channel C and D external interfaces. For more information, see Chapter 15, “High-Level Data Link Control (HDLC).”
- Disables CLKOUT and forces the pin to drive a zero externally. For more information, see “Clock Control” on page 3-32.

The Processor Revision Level (PRL) register contains the processor revision level for the device. Use this information when requesting support.

3.4 INITIALIZATION AND RESET

This document uses the following terms throughout:

- **External or power-on reset:** A reset caused by asserting $\overline{\text{RES}}$.
- **Internal reset:** A reset initiated by the watchdog timer.
- **System reset:** Resets the microcontroller (the CPU plus the internal peripherals) as well as any external peripherals connected to RESOUT. An external reset always causes a system reset; an internal reset can optionally cause a system reset.

Processor initialization or startup is accomplished by either an external reset or by an internal reset initiated by the watchdog timer. Resets force the microcontroller to terminate all execution and local bus activity. No instruction or bus activity occurs as long as the processor is in reset.

In all resets, the multiplexed pins default to the signal as shown in Table 3-7 on page 3-10 (the signal name without brackets). Pins are latched on the deassertion of $\overline{\text{RES}}$, and therefore are not affected by an internal watchdog-timer-generated reset. These latched pins include the reset configuration pins (pinstraps) shown in Table 3-5 on page 3-7 and the RESCON register inputs.

After an external or internal reset has completed and an internal processing interval elapses, the microcontroller begins execution with the instruction at physical location FFFF0h and the watchdog timer starts counting (reset enables the watchdog timer). $\overline{\text{RES}}$ must be asserted for at least 1 ms during power-up to allow the internal circuits to stabilize. If the $\overline{\text{RES}}$ signal is asserted while the watchdog timer is performing a watchdog timer reset, the external reset takes precedence.

The Am186CC/CH/CU microcontrollers also feature a Reset Out (RESOUT) signal, which indicates that the microcontroller is being reset (either externally or internally) and can be used as a system reset to reset any external peripherals connected to RESOUT.

During an external reset, RESOUT remains active (High) for two clocks after $\overline{RES}$ is deasserted. The microcontroller exits reset and begins the first valid bus cycle approximately 4.5 clocks after RES is deasserted.

With an internal reset, the watchdog timer reset duration, and therefore the duration of the RESOUT signal, is 2^{16} processor clocks. This duration allows sufficient time for external devices to reach their reset state. For more information about internal resets, see Chapter 11, “Watchdog Timer.”

Both external and internal resets set the registers to predefined values as shown in Appendix A, “Register Summary,” with the exception of the RESCON and WDTCON registers whose default values depend on the type of reset.

The Reset Configuration (RESCON) register latches system-configuration information that is presented to the processor on the address/data bus (AD15–AD0) at the deassertion of $\overline{RES}$. The interpretation of this information is system-specific. The processor does not impose any predetermined interpretation, but simply provides a means for communicating this information to software. When the $\overline{RES}$ input is asserted, the contents of the AD bus are written into the RESCON register. Note that the RESCON value is only sampled on an external reset. The system can place configuration information on the AD bus using weak external pullup or pulldown resistors, or using an external driver that is enabled during reset. The processor does not drive the AD bus during reset. For example, the RESCON register could be used to provide the software with the position of a configuration switch in the system. By using weak external pullup and pulldown resistors on the AD bus, the system could provide the microcontroller with a value corresponding to the position of a jumper during reset.

For compatibility with future devices, always write reserved bits in registers with their reset default values. The *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916, defines the bits for all the registers.

Table 3-4 CPU and Internal Peripheral States Immediately Following Power-On Reset

CPU/Peripheral	State
Am186 CPU	Enabled, executes at address FFFF0h
Chip Selects	$\overline{UCS}$ active, all other chip selects inactive
DRAM Controller	Disabled
Interrupt Controller	Disabled—only nonmaskable interrupts and traps can be taken
General-Purpose DMA and SmartDMA Channels	Disabled
Programmable I/Os	See Chapter 9, “Programmable I/O Signals”
Programmable Timers	Disabled
Watchdog Timer	Enabled with maximum time-out value (2^{16} clocks)
UART and High-Speed UART	Disabled
Synchronous Serial Interface (SSI)	Disabled
High-level Data Link Control (HDLC) Channels CC CH	Disabled
Time Slot Assigners (TSAs) CC CH	Disabled
General Circuit Interface (GCI) CC	Disabled
Universal Serial Bus (USB) Peripheral Controller CC CU	Disabled

Table 3-5 Reset Configuration Pins (Pinstraps)[1]

Signal Name	Multiplexed Signal(s)	Description
{ADEN}	BHE PIO34	**Address Enable**: If {ADEN} is held High or left floating during power-on reset, the address portion of the AD bus (AD15–AD0) is enabled or disabled during LCS, UCS, or other memory bus cycles based on how the software configures the DA bit in the UMCS or LMCS registers. In this case, the memory address is accessed on the A19–A0 pins. There is a weak internal pullup resistor on {ADEN} so no external pullup is required. This mode of operation reduces power consumption. If {ADEN} is held Low on power-on reset, the AD bus drives both addresses and data, regardless of how software configures the DA bit in the UMCS or LMCS registers.
{CLKSEL1} {CLKSEL2}	HLDA [PCS4] PIO3	**CPU PLL Mode Select 1** determines the PLL mode for the CPU clock source. **CPU PLL Mode Select 2** is sampled on the rising edge of reset and determines the PLL mode for the CPU clock source. This pin has an internal pullup resistor that is active only during reset. There are four CPU PLL modes that are selected by the values of {CLKSEL1} and {CLKSEL2} as shown below. **CPU PLL Modes** {CLKSEL1} / {CLKSEL2} / CPU PLL Mode 1 / 1 / 2X, CPU PLL enabled (default) 1 / 0 / 4X, CPU PLL enabled 0 / 1 / 1X, CPU PLL enabled 0 / 0 / PLL Bypass
{ONCE}	UCS	**ONCE Mode Request** asserted Low places the Am186CC/CH/CU microcontroller into ONCE mode. Otherwise, the controller operates normally. In ONCE mode, all pins are three-stated and remain in that state until a subsequent reset occurs. To guarantee that the controller does not inadvertently enter ONCE mode, {ONCE} has a weak internal pullup resistor that is active only during a reset. A reset ending ONCE mode should be as long as a power-on reset for the PLL to stabilize.
{UCSX8}	[MCS0] PIO4	**Upper Memory Chip Select, 8-Bit Bus** asserted Low configures the upper chip select region for an 8-bit bus size. This pin has a pullup resistor that is active only during reset, so no external pullup is required to set the bus to 16-bit mode.
{USBSEL2} CC CU {USBSEL1} CC CU	PCS1 PIO14 PCS0 PIO13	**USB Clock Mode Selects 1–2** select the USB PLL operating mode. The pins have internal pullups that are active only during reset. The USB PLL can operate in one of three modes. With a crystal and the internal USB oscillator or an external oscillator, the USB PLL can output 4X or 2X the input frequency. The USB PLL can also be disabled and the USB peripheral controller can receive its clock from the CPU PLL, which is the default mode. The pins are encoded as shown below. **USB PLL Modes** {USBSEL1} / {USBSEL2} / USB PLL Mode 1 / 1 / Use CPU clock (after CPU PLL mode select), USB PLL disabled (default) 1 / 0 / 4X, USB PLL enabled 0 / 1 / 2X, USB PLL enabled 0 / 0 / Reserved
{USBXCVR} CC CU	S0	**USB External Transceiver Enable** asserted Low disables the internal USB transceiver and enables the pins needed to hook up an external transceiver. This pin has a pullup resistor that is active only during reset, so no external pullup is required as long as the user ensures that this input is not driven Low during a power-on reset.

Notes:

1. A pinstrap is used to enable or disable features based on the state of the pin during an external reset. The pinstrap must be held in its desired state for at least 4.5 clock cycles after the deassertion of RES. The pinstraps are sampled in an external reset only (when RES is asserted), not during an internal watchdog timer-generated reset.

3.5 SIGNAL DESCRIPTIONS

Table 3-7 contains a description of the Am186CC/CH/CU microcontroller signals. Table 3-6 describes the terms used in Table 3-7. The signals are organized alphabetically within the following functional groups:

- Bus interface/general-purpose DMA request (page 3-10)
- Clocks/reset/watchdog timer (page 3-14)
- Reserved (page 3-16)
- Power and ground (page 3-16)
- Debug support (page 3-17)
- Chip selects (page 3-17)
- DRAM (page 3-19)
- Interrupts (page 3-19)
- Programmable I/O (PIOs) (page 3-21)
- Programmable timers (page 3-21)
- Asynchronous serial ports (UART and High-Speed UART) (page 3-22)
- Synchronous serial interface (SSI) (page 3-23)
- HDLC synchronous communications: channels A–D for Data Communications Equipment (DCE), Pulse Code Modulation (PCM), and General Circuit Interface (GCI) interfaces (page 3-23)
- Universal Serial Bus (USB) (page 3-27)

For pinstraps refer to Table 3-5 on page 3-7.

Table 3-6 Signal Descriptions Table Definitions

Term	Definition
General Terms	
[ ]	Indicates the pin alternate function; a pin defaults to the signal named without the brackets.
{ }	Indicates the reset configuration pin (pinstrap).
pin	Refers to the physical wire.
reset	An *external or power-on reset* is caused by asserting $\overline{\text{RES}}$. An *internal reset* is initiated by the watchdog timer. A *system reset* is one that resets the microcontroller (the CPU plus the internal peripherals) as well as any external peripherals connected to RESOUT. An external reset always causes a system reset; an internal reset can optionally cause a system reset.
signal	Refers to the electrical signal that flows across a pin.
$\overline{\text{SIGNAL}}$	A line over a signal name indicates that the signal is active Low; a signal name without a line is active High.
Signal Types	
B	Bidirectional
H	High
LS	Programmable to hold last state of pin
O	Totem pole output
OD	Open drain output
OD-O	Open drain output or totem pole output
PD	Internal pulldown resistor
PU	Internal pullup resistor
STI	Schmitt trigger input
STI-OD	Schmitt trigger input or open drain output
TS	Three-state output

Table 3-7 Signal Descriptions

Signal Name[1]	Multiplexed Signal(s)	Type	Description
BUS INTERFACE/GENERAL-PURPOSE DMA REQUEST			
A19–A0	—	O	**Address Bus** supplies nonmultiplexed memory or I/O addresses to the system one half of a CLKOUT period earlier than the multiplexed address and data bus (AD15–AD0). During bus-hold or reset conditions, the address bus is three-stated with pulldowns. When the lower or upper chip-select regions are configured for DRAM mode, the A19–A0 bus provides the row and column addresses at the appropriate times. The upper and lower memory chip-select ranges can be individually configured for DRAM mode.
AD15–AD0	—	B	**Address and Data Bus** time-multiplexed pins supply memory or I/O addresses and data to the system. This bus can supply an address to the system during the first period of a bus cycle (t_1). It transmits (write cycle) or receives (read cycle) data to or from the system during the remaining periods of that cycle (t2, t3, and t4). The address phase of these pins can be disabled—see the {$\overline{\text{ADEN}}$} pin description in Table 3-5 on page 3-7. During a reset condition, the address and data bus is three-stated with pulldowns, and during a bus hold it is three-stated. In addition, during a reset the state of the address and data bus pins (AD15–AD0) is latched into the Reset Configuration (RESCON) register. This feature can be used to provide software with information about the external system at reset time.
ALE	[PIO33]	O	**Address Latch Enable** indicates to the system that an address appears on the address and data bus (AD15–AD0). The address is guaranteed valid on the falling edge of ALE. ALE is three-stated and has a pulldown resistor during bus-hold or reset conditions.
ARDY	[PIO8]	STI	**Asynchronous Ready** is a true asynchronous ready that indicates to the microcontroller that the addressed memory space or I/O device will complete a data transfer. The ARDY pin is asynchronous to CLKOUT and is active High. To guarantee the number of wait states inserted, ARDY or SRDY must be synchronized to CLKOUT. If the falling edge of ARDY is not synchronized to CLKOUT as specified, an additional clock period can be added. To always assert the ready condition to the microcontroller, tie ARDY and SRDY High. If the system does not use ARDY, tie the pin Low to yield control to SRDY.

Table 3-7 Signal Descriptions (Continued)

<table>
<tr><th>Signal Name[1]</th><th>Multiplexed Signal(s)</th><th>Type</th><th>Description</th></tr>
<tr><td>$\overline{BHE}$</td><td>[PIO34]
{$\overline{ADEN}$}</td><td>O</td><td>Bus High Enable: During a memory access, $\overline{BHE}$ and the least-significant address bit (AD0) indicate to the system which bytes of the data bus (upper, lower, or both) participate in a bus cycle. The $\overline{BHE}$ and AD0 pins are encoded as follows:
Data Byte Encoding<table><tr><th>$\overline{BHE}$</th><th>AD0</th><th>Type of Bus Cycle</th></tr><tr><td>0</td><td>0</td><td>Word transfer</td></tr><tr><td>0</td><td>1</td><td>High byte transfer (bits 15–8)</td></tr><tr><td>1</td><td>0</td><td>Low byte transfer (bits 7–0)</td></tr><tr><td>1</td><td>1</td><td>Refresh</td></tr></table>$\overline{BHE}$ is asserted during t_1 and remains asserted through t_3 and t_W. $\overline{BHE}$ does not require latching. $\overline{BHE}$ is three-stated with a pullup during bus-hold and reset conditions.

$\overline{WLB}$ and $\overline{WHB}$ implement the functionality of $\overline{BHE}$ and AD0 for high and low byte write enables, and they have timing appropriate for use with the nonmultiplexed bus interface.

$\overline{BHE}$ also signals DRAM refresh cycles when using the multiplexed address and data (AD) bus. A refresh cycle is indicated when both $\overline{BHE}$ and AD0 are High. During refresh cycles, the AD bus is driven during the t_1 phase and three-stated during the t_2, t_3, and t_4 phases. The value driven on the A bus is undefined during a refresh cycle. For this reason, the A0 signal cannot be used in place of the AD0 signal to determine refresh cycles.</td></tr>
<tr><td>$\overline{BSIZE8}$</td><td>—</td><td>O</td><td>Bus Size 8 is asserted during t_1–t_4 to indicate an 8-bit cycle, or is deasserted to indicate a 16-bit cycle.</td></tr>
<tr><td>$\overline{DEN}$</td><td>[$\overline{DS}$]
[PIO30]</td><td>O</td><td>Data Enable supplies an output enable to an external data-bus transceiver. $\overline{DEN}$ is asserted during memory and I/O cycles. $\overline{DEN}$ is deasserted when DT/$\overline{R}$ changes state. $\overline{DEN}$ is three-stated with a pullup during bus-hold or reset conditions.</td></tr>
<tr><td>[DRQ0]

DRQ1</td><td>PIO9

—</td><td>STI

STI</td><td>DMA Requests 0 and 1 indicate to the microcontroller that an external device is ready for a DMA channel to perform a transfer. DRQ1–[DRQ0] are level-triggered and internally synchronized. DRQ1–[DRQ0] are not latched and must remain active until serviced.</td></tr>
</table>

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
[$\overline{DS}$]	$\overline{DEN}$ [PIO30]	O	**Data Strobe** provides a signal where the write cycle timing is identical to the read cycle timing. When used with other control signals, [$\overline{DS}$] provides an interface for 68K-type peripherals without the need for additional system interface logic. When [$\overline{DS}$] is asserted, addresses are valid. When [$\overline{DS}$] is asserted on writes, data is valid. When [$\overline{DS}$] is asserted on reads, data can be driven on the AD bus. Following a reset, this pin is configured as $\overline{DEN}$. The pin is then configured by software to operate as [$\overline{DS}$].
DT/$\overline{R}$	[PIO29]	O	**Data Transmit or Receive** indicates which direction data should flow through an external data-bus transceiver. When DT/$\overline{R}$ is asserted High, the microcontroller transmits data. When this pin is deasserted Low, the microcontroller receives data. DT/$\overline{R}$ is three-stated with a pullup during a bus-hold or reset condition.
HLDA	{CLKSEL1}	O	**Bus-Hold Acknowledge** is asserted to indicate to an external bus master that the microcontroller has relinquished control of the local bus. When an external bus master requests control of the local bus (by asserting HOLD), the microcontroller completes the bus cycle in progress, then relinquishes control of the bus to the external bus master by asserting HLDA and three-stating A19–A0, AD15–AD0, $\overline{S2}$–$\overline{S0}$, and S6. The following are also three-stated and have pullups: $\overline{BHE}$, $\overline{DEN}$, DT/$\overline{R}$, $\overline{LCS}$, $\overline{MCS3}$–$\overline{MCS0}$, $\overline{PCS7}$–$\overline{PCS0}$, $\overline{RD}$, $\overline{UCS}$, $\overline{WHB}$, $\overline{WLB}$, and $\overline{WR}$. ALE is three-stated and has a pulldown. When the external bus master has finished using the local bus, it indicates this to the microcontroller by deasserting HOLD. The microcontroller responds by deasserting HLDA. If the microcontroller requires access to the bus (for example, for refresh), the microcontroller deasserts HLDA before the external bus master deasserts HOLD. The external bus master must be able to deassert HOLD and allow the microcontroller access to the bus. See the timing diagrams for bus hold in the microcontroller data sheet.

Table 3-7 Signal Descriptions (Continued)

<table>
<tr><th>Signal Name[1]</th><th>Multiplexed Signal(s)</th><th>Type</th><th>Description</th></tr>
<tr><td>HOLD</td><td>—</td><td>STI</td><td>Bus-Hold Request indicates to the microcontroller that an external bus master needs control of the local bus.

The microcontroller HOLD latency time—the time between HOLD request and HOLD acknowledge—is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is second only to DRAM refresh requests in priority of activity requests received by the processor. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency can be as great as four bus cycles. This occurs if a DMA word transfer operation is taking place from an odd address to an odd address. This is a total of 16 clock cycles or more if wait states are required. In addition, if locked transfers are performed, the HOLD latency time is increased by the length of the locked transfer. HOLD latency is also potentially increased by DRAM refreshes.

The board designer is responsible for properly terminating the HOLD input.

For more information, see the HLDA pin description above.</td></tr>
<tr><td>$\overline{RD}$</td><td>—</td><td>O</td><td>Read Strobe indicates to the system that the microcontroller is performing a memory or I/O read cycle. $\overline{RD}$ is guaranteed not to be asserted before the address and data bus is three-stated during the address-to-data transition. $\overline{RD}$ is three-stated with a pullup during bus-hold or reset conditions.</td></tr>
<tr><td>$\overline{S2}$
$\overline{S1}$
$\overline{S0}$</td><td>—
—
{$\overline{USBXCVR}$}</td><td>O</td><td>Bus Cycle Status 2–0 indicate to the system the type of bus cycle in progress. $\overline{S2}$ can be used as a logical memory or I/O indicator, and $\overline{S1}$ can be used as a data transmit or receive indicator. $\overline{S2}$–$\overline{S0}$ are three-stated during bus hold and three-stated with a pullup during reset. The $\overline{S2}$–$\overline{S0}$ pins are encoded as follows:

Bus Status Pins
<table>
<tr><th>$\overline{S2}$</th><th>$\overline{S1}$</th><th>$\overline{S0}$</th><th>Bus Cycle</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Reserved</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Read data from I/O</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Write data to I/O</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Halt</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Instruction fetch</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Read data from memory</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Write data to memory</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>None (passive)</td></tr>
</table></td></tr>
<tr><td>S6</td><td>—</td><td>O</td><td>Bus Cycle Status Bit 6: This signal is asserted during t_1–t_4 to indicate a DMA-initiated bus cycle or a refresh cycle. S6 is three-stated during bus hold and three-stated with a pulldown during reset.</td></tr>
</table>

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
SRDY	[PIO35]	STI	**Synchronous Ready** indicates to the microcontroller that the addressed memory space or I/O device will complete a data transfer. The SRDY pin accepts an active High input synchronized to CLKOUT. Using SRDY instead of ARDY allows a relaxed system timing because of the elimination of the one-half clock period required to internally synchronize ARDY. To always assert the ready condition to the microcontroller, tie SRDY High. If the system does not use SRDY, tie the pin Low to yield control to ARDY.
$\overline{\text{WHB}}$ $\overline{\text{WLB}}$	— —	O O	**Write High Byte** and **Write Low Byte** indicate to the system which bytes of the data bus (upper, lower, or both) participate in a write cycle. In 80C186 microcontroller designs, this information is provided by $\overline{\text{BHE}}$, AD0, and $\overline{\text{WR}}$. However, by using $\overline{\text{WHB}}$ and $\overline{\text{WLB}}$, the standard system interface logic and external address latch that were required are eliminated. $\overline{\text{WHB}}$ is asserted with AD15–AD8. $\overline{\text{WHB}}$ is the logical AND of $\overline{\text{BHE}}$ and $\overline{\text{WR}}$. This pin is three-stated with a pullup during bus-hold or reset conditions. $\overline{\text{WLB}}$ is asserted with AD7–AD0. $\overline{\text{WLB}}$ is the logical AND of AD0 and $\overline{\text{WR}}$. This pin is three-stated with a pullup during bus-hold or reset conditions.
$\overline{\text{WR}}$	[PIO15]	O	**Write Strobe** indicates to the system that the data on the bus is to be written to a memory or I/O device. $\overline{\text{WR}}$ is three-stated with a pullup during bus-hold or reset conditions.
CLOCKS/RESET/WATCHDOG TIMER			
CLKOUT	—	O	**Clock Output** supplies the clock to the system. Depending on the values of the CPU mode select pinstraps, {CLKSEL1} and {CLKSEL2}, CLKOUT operates at either the PLL frequency or the source input frequency during PLL Bypass mode. (See Table 3-5 on page 3-7.) CLKOUT remains active during bus-hold or reset conditions. The DISCLK bit in the SYSCON register can be set to disable the CLKOUT signal. Refer to the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916. All synchronous AC timing specifications not associated with SSI, HDLCs, UARTs, and the USB are synchronous to CLKOUT.

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
$\overline{\text{RES}}$	—	STI	**Reset** requires the microcontroller to perform a reset. When $\overline{\text{RES}}$ is asserted, the microcontroller immediately terminates its present activity, clears its internal logic, and on the deassertion of $\overline{\text{RES}}$, transfers CPU control to the reset address FFFF0h. $\overline{\text{RES}}$ must be asserted for at least 1 ms to allow the internal circuits to stabilize. $\overline{\text{RES}}$ can be asserted asynchronously to CLKOUT because $\overline{\text{RES}}$ is synchronized internally. For proper initialization, V_{CC} must be within specifications, and CLKOUT must be stable for more than four CLKOUT periods during which $\overline{\text{RES}}$ is asserted. If $\overline{\text{RES}}$ is asserted while the watchdog timer is performing a watchdog-timer reset, the external reset takes precedence over the watchdog-timer reset. This means that the RESOUT signal asserts as with any external reset and the WDTCON register will not have the RSTFLAG bit set. In addition, the microcontroller will exit reset based on the external reset timing, i.e., 4.5 clocks after the deassertion of $\overline{\text{RES}}$ rather than 2^{16} clocks after the watchdog timer timeout occurred. The microcontroller begins fetching instructions approximately 6.5 CLKOUT periods after $\overline{\text{RES}}$ is deasserted. This input is provided with a Schmitt trigger to facilitate power-on $\overline{\text{RES}}$ generation via a resistor-capacitor (RC) network.
RESOUT	—	O	**Reset Out** indicates that the microcontroller is being reset (either externally or internally), and the signal can be used as a system reset to reset any external peripherals connected to RESOUT. During an external reset, RESOUT remains active (High) for two clocks after $\overline{\text{RES}}$ is deasserted. The microcontroller exits reset and begins the first valid bus cycle approximately 4.5 clocks after $\overline{\text{RES}}$ is deasserted.
[UCLK]	[USBSOF] [USBSCI] PIO21	STI	**UART Clock** can be used instead of the processor clock as the source clock for either the UART or the High-Speed UART. The source clock for the UART and the High-Speed UART are selected independently and both can use the same source.
USBX1 CC CU USBX2 CC CU	— —	STI O	**USB Controller Crystal Input** (USBX1) and **USB Controller Crystal Output** (USBX2) provide connections for a fundamental mode, parallel-resonant crystal used by the internal USB oscillator circuit. If the CPU crystal is used to generate the USB clock, USBX1 must be pulled down.
X1 X2	— —	O STI	**CPU Crystal Input** (X1) and **CPU Crystal Output** (X2) provide connections for a fundamental mode, parallel-resonant crystal used by the internal oscillator circuit. If an external oscillator is used, inject the signal directly into X1 and leave X2 floating.

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
PINSTRAPS (See Table 3-5 on page 3-7.)			
RESERVED			
RSVD_75 CH	—	—	On the Am186CH HDLC microcontroller, the RSVD_75 pin should be tied externally to V_{SS}. On the Am186CH HDLC microcontroller, pins RSVD_75, RSVD_76, RSVD_80, RSVD_81, and RSVD_101–RSVD_104 and are reserved. On the Am186CC and Am186CU microcontrollers, pins RSVD_101–RSVD_104 are reserved unless pinstrap {$\overline{\text{USBXCVR}}$} is sampled Low on the rising edge of RESET. On the Am186CU USB microcontroller, pins RSVD_119–RSVD_116 are reserved. All other reserved pins should not be connected.
RSVD_76 CH	—		
RSVD_80 CH	—		
RSVD_81 CH	—		
RSVD_101	UTXDPLS		
RSVD_102	UTXDMNS		
RSVD_103	$\overline{\text{UXVOE}}$		
RSVD_104	UXVRCV		
RSVD_116 CU	—		
RSVD_117 CU	—		
RSVD_118 CU	—		
RSVD_119 CU	—		
POWER AND GROUND			
V_{CC} (15) CC CU (16) CH	—	STI	**Digital Power Supply** pins supply power (+3.3 ± 0.3 V) to the microcontroller logic.
V_{CC} _A (1)	—	STI	**Analog Power Supply** pin supplies power (+3.3 ± 0.3 V) to the oscillators and PLLs.
V_{CC} _USB (1) CC CU	—	STI	**USB Power Supply** pin supplies power (+3.3 ± 0.3 V) to the USB block.
V_{SS} (15) CC CU (16) CH	—	STI	**Digital Ground** pins connect the microcontroller logic to the system ground.
V_{SS} _A (1)	—	STI	**Analog Ground** pin connects the oscillators and PLLs to the system ground.
V_{SS}_USB (1) CC CU	—	STI	**USB Ground** pin connects the USB block to the system ground.

Table 3-7 Signal Descriptions (Continued)

<table>
<tr><th>Signal Name[1]</th><th>Multiplexed Signal(s)</th><th>Type</th><th>Description</th></tr>
<tr><td colspan="4">DEBUG SUPPORT</td></tr>
<tr><td>QS1–QS0</td><td>—</td><td>O</td><td>Queue Status 1–0 values provide information to the system concerning the interaction of the CPU and the instruction queue. The pins have the following meanings:
Queue Status Pins
<table>
<tr><th>QS1</th><th>QS0</th><th>Queue Operation</th></tr>
<tr><td>0</td><td>0</td><td>None</td></tr>
<tr><td>0</td><td>1</td><td>First opcode byte fetched from queue</td></tr>
<tr><td>1</td><td>0</td><td>Queue was initialized</td></tr>
<tr><td>1</td><td>1</td><td>Subsequent byte fetched from queue</td></tr>
</table></td></tr>
<tr><td colspan="4">The following signals are also used by emulators: A19–A0, AD15–AD0, {ADEN}, ALE, ARDY, BHE, BSIZE8, CAS1–CAS0, CLKOUT, {CLKSEL2}–{CLKSEL1}, HLDA, HOLD, LCS, MCS3–MCS0, NMI, {ONCE}, QS1–QS0, RAS1–RAS0, RD, RES, RESOUT, S2–S0, S6, SRDY, UCS, {UCSX8}, WHB, WLB, WR. For more information, see Chapter 4, “Emulator Support.”</td></tr>
<tr><td colspan="4">CHIP SELECTS</td></tr>
<tr><td>LCS</td><td>[RAS0]</td><td>O</td><td>Lower Memory Chip Select indicates to the system that a memory access is in progress to the lower memory block. The base address and size of the lower memory block are programmable up to 512 Kbyte. LCS can be configured for 8-bit or 16-bit bus size. LCS is three-stated with a pullup resistor during bus-hold or reset conditions.</td></tr>
<tr><td>[MCS0]

MCS1

MCS2

[MCS3]</td><td>{UCSX8}
PIO4

[CAS1]

[CAS0]

[RAS1]
PIO5</td><td>O</td><td>Midrange Memory Chip Selects 0–3 indicate to the system that a memory access is in progress to the corresponding region of the midrange memory block. The base address and size of the midrange memory block are programmable. The midrange chip selects can be configured for 8-bit or 16-bit bus size. The midrange chip selects are three-stated with pullup resistors during bus-hold or reset conditions.

[MCS0] can be programmed as the chip select for the entire middle chip select address range.

Unlike the UCS and LCS chip selects that operate relative to the earlier timing of the nonmultiplexed A address bus, the MCS outputs assert with the multiplexed AD address and data bus timing.</td></tr>
</table>

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
$\overline{\text{PCS0}}$	[PIO13] {USBSEL1}	O	**Peripheral Chip Selects 0–7** indicate to the system that an access is in progress to the corresponding region of the peripheral address block (either I/O or memory address space). The base address of the peripheral address block is programmable. $\overline{\text{PCS7}}$–$\overline{\text{PCS0}}$ are three-stated with pullup resistors during bus-hold or reset conditions. Unlike the $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$ chip selects that operate relative to the earlier timing of the nonmultiplexed A address bus, the $\overline{\text{PCS}}$ outputs assert with the multiplexed AD address and data bus timing.
$\overline{\text{PCS1}}$	[PIO14] {USBSEL2}		
$\overline{\text{PCS2}}$	—		
$\overline{\text{PCS3}}$	—		
[$\overline{\text{PCS4}}$]	PIO3 [CLKSEL2]		
[$\overline{\text{PCS5}}$]	PIO2		
[$\overline{\text{PCS6}}$]	PIO32		
[$\overline{\text{PCS7}}$]	PIO31		
$\overline{\text{UCS}}$	{$\overline{\text{ONCE}}$}	O	**Upper Memory Chip Select** indicates to the system that a memory access is in progress to the upper memory block. The base address and size of the upper memory block are programmable up to 512 Kbytes. $\overline{\text{UCS}}$ is three-stated with a weak pullup during bus-hold or reset conditions. The $\overline{\text{UCS}}$ can be configured for an 8-bit or 16-bit bus size out of reset. For additional information, see the {$\overline{\text{UCSX8}}$} pin description in Table 3-5 on page 3-7. After reset, $\overline{\text{UCS}}$ is active for the 64-Kbyte memory range from F0000h to FFFFFh, including the reset address of FFFF0h.

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
DRAM			
[$\overline{\text{CAS0}}$] [$\overline{\text{CAS1}}$]	$\overline{\text{MCS2}}$ $\overline{\text{MCS1}}$	O	**Column Address Strobes 0–1**: When either the upper or lower chip select regions are configured for DRAM, these pins provide the column address strobe signals to the DRAM. The $\overline{\text{CAS}}$ signals can be used to perform byte writes in a manner similar to $\overline{\text{WLB}}$ and $\overline{\text{WHB}}$, respectively, i.e., [$\overline{\text{CAS0}}$] corresponds to the low byte ($\overline{\text{WLB}}$) and [$\overline{\text{CAS1}}$] corresponds to the high byte ($\overline{\text{WHB}}$).
[$\overline{\text{RAS0}}$]	$\overline{\text{LCS}}$	O	**Row Address Strobe 0**: When the lower chip select region is configured to DRAM, this pin provides the row address strobe signal to the lower DRAM bank.
[$\overline{\text{RAS1}}$]	[$\overline{\text{MCS3}}$] PIO5	O	**Row Address Strobe 1**: When the upper chip select region is configured to DRAM, this pin provides the row address strobe signal to the upper DRAM bank.
INTERRUPTS			
INT5–INT0 [INT6] [INT7] [INT8]	— PIO19 PIO7 [PWD] PIO6	STI	**Maskable Interrupt Requests 0–8** indicate to the microcontroller that an external interrupt request has occurred. If the individual pin is not masked, the microcontroller transfers program execution to the location specified by the associated interrupt vector in the microcontroller's interrupt vector table. Interrupt requests are synchronized internally and can be edge-triggered or level-triggered. The interrupt polarity is programmable.To guarantee interrupt recognition for edge-triggered interrupts, the user should hold the interrupt source for a minimum of five system clocks. A second interrupt from the same source is not recognized until after an acknowledge of the first. The board designer is responsible for properly terminating the INT8–INT0 inputs.

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
NMI	—	STI	**Nonmaskable Interrupt** indicates to the microcontroller that an interrupt request has occurred. The NMI signal is the highest priority hardware interrupt and cannot be masked. The microcontroller always transfers program execution to the location specified by the nonmaskable interrupt vector in the microcontroller's interrupt vector table when NMI is asserted. Although NMI is the highest priority hardware interrupt source, it does not participate in the priority resolution process of the maskable interrupts. There is no bit associated with NMI in the interrupt in-service or interrupt request registers. This means that a new NMI request can interrupt an executing NMI interrupt service routine. As with all hardware interrupts, the interrupt flag (IF) is cleared when the processor takes the interrupt, disabling the maskable interrupt sources. However, if maskable interrupts are re-enabled by software in the NMI interrupt service routine (for example, via the STI instruction), the fact that an NMI is currently in service does not have any effect on the priority resolution of maskable interrupt requests. For this reason, it is strongly advised that the interrupt service routine for NMI should not enable the maskable interrupts. An NMI transition from Low to High is latched and synchronized internally, and it initiates the interrupt at the next instruction boundary. To guarantee that the interrupt is recognized, the NMI pin must be asserted for at least one CLKOUT period. The board designer is responsible for properly terminating the NMI input.
Also configurable as interrupts are PIO5, PIO15, PIO27, PIO29, PIO30, PIO33, PIO34, and PIO35. For more information, see Chapter 9, "Programmable I/O Signals."			

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
PROGRAMMABLE I/O (PIOS)			
PIO47–PIO0	(For a list of the multiplexed signals ordered by PIO number, see Table 3-2.)	B	**Shared Programmable I/O** pins can be programmed with the following attributes: PIO function (enabled/disabled), direction (input/output), and weak pullup or pulldown. After a reset, the PIO pins default to various configurations. Most of the PIO pins are configured as PIO inputs with pullup after reset. The system initialization code must reconfigure any PIO pins as required. PIO5, PIO15, PIO27, PIO29, PIO30, and PIO33–PIO35 are capable of generating an interrupt on the shared interrupt channel 14. The multiplexed signals PIO8/ARDY, PIO13/$\overline{\text{PCS0}}$, PIO14/$\overline{\text{PCS1}}$, PIO15/$\overline{\text{WR}}$, PIO29/DT/$\overline{\text{R}}$, PIO30/$\overline{\text{DEN}}$, PIO33/ALE, PIO34/$\overline{\text{BHE}}$, and PIO35/SRDY default to non-PIO operation at reset. The following PIO signals are multiplexed with alternate signals that can be used by emulators: PIO8, PIO15, PIO33, PIO34, and PIO35. Consider any emulator requirements for the alternate signals before using these pins as PIOs.
PROGRAMMABLE TIMERS			
[PWD]	[INT8] PIO6	STI	**Pulse-Width Demodulator:** If pulse-width demodulation is enabled, [PWD] processes a signal through the Schmitt trigger input. [PWD] is used internally to drive [TMRIN0] and [INT8], and [PWD] is inverted internally to drive [TMRIN1] and an additional internal interrupt. If interrupts are enabled and Timer 0 and Timer 1 are properly configured, the pulse width of the alternating [PWD] signal can be calculated by comparing the values in Timer 0 and Timer 1. In PWD mode, the signals [TMRIN0]/PIO27 and [TMRIN1]/PIO0 can be used as PIOs. If they are not used as PIOs they are ignored internally. The additional internal interrupt used in PWD mode uses the same interrupt channel as [INT7]. If [INT7] is used, it must be assigned to the shared interrupt channel.

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
[TMRIN0] [TMRIN1]	PIO27 PIO0	STI STI	**Timer Inputs 0–1** supply a clock or control signal to the internal microcontroller timers. After internally synchronizing a Low-to-High transition on [TMRIN1]–[TMRIN0], the microcontroller increments the timer. [TMRIN1]–[TMRIN0] must be tied High if not being used. When PIO is enabled for one or both, the pin is pulled High internally. [TMRIN1]–[TMRIN0] are driven internally by [INT8]/[PWD] when pulse-width demodulation functionality is enabled. The [TMRIN1]–[TMRIN0] pins can be used as PIOs when pulse-width demodulation is enabled.
[TMROUT0] [TMROUT1]	PIO28 PIO1	O O	**Timer Outputs 0–1** supply the system with either a single pulse or a continuous waveform with a programmable duty cycle. [TMROUT1]–[TMROUT0] are three-stated during bus-hold or reset conditions.
ASYNCHRONOUS SERIAL PORTS (UART AND HIGH-SPEED UART)			
UART			
[RXD_U]	DCE_RXD_D [PCM_RXD_D] PIO26	STI	**Receive Data UART** is the asynchronous serial receive data signal that supplies data from the asynchronous serial port to the microcontroller.
[TXD_U]	[DCE_TXD_D] [PCM_TXD_D] PIO20	O	**Transmit Data UART** is the asynchronous serial transmit data signal that supplies data to the asynchronous serial port from the microcontroller.
[$\overline{\text{CTS\_U}}$]	[DCE_TCLK_D] [PCM_FSC_D] PIO24	STI	**Clear-To-Send UART** provides the Clear-to-Send signal from the asynchronous serial port when hardware flow control is enabled for the port. The [$\overline{\text{CTS\_U}}$] signal gates the transmission of data from the serial port transmit shift register. When [$\overline{\text{CTS\_U}}$] is asserted, the transmitter begins transmission of a frame of data, if any is available. If [$\overline{\text{CTS\_U}}$] is deasserted, the transmitter holds the data in the serial port transmit shift register. The value of [$\overline{\text{CTS\_U}}$] is checked only at the beginning of the transmission of the frame. [$\overline{\text{CTS\_U}}$] and [$\overline{\text{RTR\_U}}$] form the hardware handshaking interface for the UART.
[$\overline{\text{RTR\_U}}$]	DCE_RCLK_D [PCM_CLK_D] PIO25	O	**Ready-To-Receive UART** provides the Ready-to-Receive signal for the asynchronous serial port when hardware flow control is enabled for the port. The [$\overline{\text{RTR\_U}}$] signal is asserted when the associated serial port receive data register does not contain valid, unread data. [$\overline{\text{CTS\_U}}$] and [$\overline{\text{RTR\_U}}$] form the hardware handshaking interface for the UART.
HIGH-SPEED UART			
[RXD_HU]	PIO16	STI	**Receive Data High-Speed UART** is the asynchronous serial receive data signal that supplies data from the high-speed serial port to the microcontroller.
TXD_HU	—	O	**Transmit Data High-Speed UART** is the asynchronous serial transmit data signal that supplies data to the high-speed serial port from the microcontroller.

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
[$\overline{\text{CTS\_HU}}$]	[$\overline{\text{DCE\_CTS\_D}}$] [PCM_TSC_D] PIO46	STI	**Clear-To-Send High-Speed UART** provides the Clear-to-Send signal from the high-speed asynchronous serial port when hardware flow control is enabled for the port. The [$\overline{\text{CTS\_HU}}$] signal gates the transmission of data from the serial port transmit shift register. When [$\overline{\text{CTS\_HU}}$] is asserted, the transmitter begins transmission of a frame of data, if any is available. If [$\overline{\text{CTS\_HU}}$] is deasserted, the transmitter holds the data in the serial port transmit shift register. The value of [$\overline{\text{CTS\_HU}}$] is checked only at the beginning of the transmission of the frame. [$\overline{\text{CTS\_HU}}$] and [$\overline{\text{RTR\_HU}}$] form the hardware handshaking interface for the High-Speed UART.
[$\overline{\text{RTR\_HU}}$]	[$\overline{\text{DCE\_RTR\_D}}$] PIO47	O	**Ready-To-Receive High-Speed UART** provides the Ready-to-Receive signal to the high-speed asynchronous serial port when hardware flow control is enabled for the port. The [$\overline{\text{RTR\_HU}}$] signal is asserted when the associated serial port receive data register does not contain valid, unread data. [$\overline{\text{CTS\_HU}}$] and [$\overline{\text{RTR\_HU}}$] form the hardware handshaking interface for the High-Speed UART.
SYNCHRONOUS SERIAL INTERFACE (SSI)			
[SCLK]	PIO11	O	**Serial Clock** provides the clock for the synchronous serial interface to allow synchronous transfers between the microcontroller and a slave device.
[SDATA]	PIO12	B	**Serial Data** is used to transmit and receive data between the microcontroller and a slave device on the synchronous serial interface.
[SDEN]	PIO10	O	**Serial Data Enable** enables data transfers on the synchronous serial interface.
HIGH-LEVEL DATA LINK CONTROL SYNCHRONOUS COMMUNICATION INTERFACES			
HDLC Channel A (DCE) CC CH			
DCE_RXD_A CC CH	[GCI_DD_A] [PCM_RXD_A]	STI	**DCE Receive Data Channel A** is the serial data input pin for the channel A DCE interface.
DCE_TXD_A CC CH	[GCI_DU_A] [PCM_TXD_A]	OD-O	**DCE Transmit Data Channel A** is the serial data output pin for the channel A DCE interface.
DCE_RCLK_A CC CH	[GCI_DCL_A] [PCM_CLK_A]	STI	**DCE Receive Clock Channel A** provides the receive clock to the channel A DCE interface. If the same clock is to be used for both transmit and receive, then this pin should be tied to the DCE_TCLK_A pin externally. The DCE function is the default at reset, so the board designer is responsible for properly terminating the DCE_RCLK_A input.
DCE_TCLK_A CC CH	[GCI_FSC_A] [PCM_FSC_A]	STI	**DCE Transmit Clock Channel A** provides the transmit clock to the channel A DCE interface. If the same clock is to be used for both transmit and receive, then this pin should be tied to the DCE_RCLK_A pin externally. The DCE function is the default at reset, so the board designer is responsible for properly terminating the DCE_TCLK_A input.

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
[DCE_CTS_A] CC CH	[PCM_TSC_A] PIO17	STI	**DCE Clear-To-Send Channel A** indicates to the channel A DCE interface that an external serial interface is ready to receive data. [DCE_CTS_A] and [DCE_RTR_A] provide the handshaking for the channel A DCE interface.
[DCE_RTR_A] CC CH	PIO18	O	**DCE Ready-to-Receive Channel A** indicates to an external serial interface that the internal channel A DCE interface is ready to accept data. [DCE_CTS_A] and [DCE_RTR_A] provide the handshaking for the channel A DCE interface.
HDLC Channel B (DCE) CC CH			
[DCE_RXD_B] CC CH	[PCM_RXD_B] PIO36	STI	**DCE Receive Data Channel B** is the serial data input pin for the channel B DCE interface.
[DCE_TXD_B] CC CH	[PCM_TXD_B] PIO37	OD-O	**DCE Transmit Data Channel B** is the serial data output pin for the channel B DCE interface.
[DCE_RCLK_B] CC CH	[PCM_CLK_B] PIO40	STI	**DCE Receive Clock Channel B** provides the receive clock to the channel B DCE interface. If the same clock is to be used for both transmit and receive, this pin should be tied to the [DCE_TCLK_B] pin externally.
[DCE_TCLK_B] CC CH	[PCM_FSC_B] PIO41	STI	**DCE Transmit Clock Channel B** provides the transmit clock to the channel B DCE interface. If the same clock is to be used for both transmit and receive, this pin should be tied to the [DCE_RCLK_B] pin externally.
[DCE_CTS_B] CC CH	[PCM_TSC_B] PIO38	STI	**DCE Clear-To-Send Channel B** indicates to the channel B DCE interface that an external serial interface is ready to receive data. [DCE_CTS_B] and [DCE_RTR_B] provide the handshaking for the channel B DCE interface.
[DCE_RTR_B] CC CH	PIO39	O	**DCE Ready-to-Receive Channel B** indicates to an external serial interface that the internal channel B DCE interface is ready to accept data. [DCE_CTS_B] and [DCE_RTR_B] provide the handshaking for the channel B DCE interface.
HDLC Channel C (DCE) CC			
[DCE_RXD_C] CC	[PCM_RXD_C] PIO42	STI	**DCE Receive Data Channel C** is the serial data input pin for the channel C DCE interface.
[DCE_TXD_C] CC	[PCM_TXD_C] PIO43	OD-O	**DCE Transmit Data Channel C** is the serial data output pin for the channel C DCE interface.
[DCE_RCLK_C] CC	[PCM_CLK_C] PIO22	STI	**DCE Receive Clock Channel C** provides the receive clock to the channel C DCE interface. If the same clock is to be used for both transmit and receive, this pin should be tied to the [DCE_TCLK_C] pin externally.
[DCE_TCLK_C] CC	[PCM_FSC_C] PIO23	STI	**DCE Transmit Clock Channel C** provides the transmit clock to the channel C DCE interface. If the same clock is to be used for both transmit and receive, this pin should be tied to the [DCE_RCLK_C] pin externally.
[DCE_CTS_C] CC	[PCM_TSC_C] PIO44	STI	**DCE Clear-To-Send Channel C** indicates to the channel C DCE interface that an external serial interface is ready to receive data. [DCE_CTS_C] and [DCE_RTR_C] provide the handshaking for the channel C DCE interface.

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
[DCE_RTR_C] CC	PIO45	O	**DCE Ready-to-Receive Channel C** indicates to an external serial interface that the internal channel C DCE is ready to accept data. [DCE_CTS_C] and [DCE_RTR_C] provide the handshaking for the channel C DCE interface.
HDLC Channel D (DCE) CC			
DCE_RXD_D CC	[RXD_U] (UART) [PCM_RXD_D] PIO26	STI	**DCE Receive Data Channel D** is the serial data input pin for the channel D DCE interface.
[DCE_TXD_D] CC	[TXD_U] (UART) [PCM_TXD_D] PIO20	OD-O	**DCE Transmit Data Channel D** is the serial data output pin for the channel D DCE interface.
DCE_RCLK_D CC	[RTR_U] (UART) [PCM_CLK_D] PIO25	STI	**DCE Receive Clock Channel D** provides the receive clock to the channel D DCE interface. If the same clock is to be used for both transmit and receive, then this pin should be tied to the [DCE_TCLK_D] pin externally.
[DCE_TCLK_D] CC	[CTS_U] (UART) [PCM_FSC_D] PIO24	STI	**DCE Transmit Clock Channel D** provides the transmit clock to the channel D DCE interface. If the same clock is to be used for both transmit and receive, then this pin should be tied to the DCE_RCLK_D pin externally.
[DCE_CTS_D] CC	[CTS_HU] (High-Speed UART) [PCM_TSC_D] PIO46	STI	**DCE Clear-To-Send Channel D** indicates to the channel D DCE interface that an external serial interface is ready to receive data. [DCE_CTS_D] and [DCE_RTR_D] provide the handshaking for DCE Channel D.
[DCE_RTR_D] CC	[RTR_HU] (High-Speed UART) PIO47	O	**DCE Ready-To-Receive Channel D** indicates to an external serial interface that the internal channel D DCE interface is ready to accept data. [DCE_CTS_D] and [DCE_RTR_D] provide the handshaking for the channel D DCE interface.
HDLC Channel A (PCM) CC CH			
[PCM_RXD_A] CC CH	DCE_RXD_A [GCI_DD_A]	STI	**PCM Receive Data Channel A** is the serial data input pin for the channel A PCM Highway interface.
[PCM_TXD_A] CC CH	DCE_TXD_A [GCI_DU_A]	O-LS-OD	**PCM Transmit Data Channel A** is the serial data output pin for the channel A PCM Highway interface.
[PCM_CLK_A] CC CH	DCE_RCLK_A [GCI_DCL_A]	STI	**PCM Clock** is the single transmit and receive data clock pin for the channel A PCM Highway interface.
[PCM_FSC_A] CC CH	DCE_TCLK_A [GCI_FSC_A]	STI	**PCM Frame Synchronization Clock** provides the Frame Synchronization Clock input (usually 8 kHz) for the channel A PCM Highway interface.
[PCM_TSC_A] CC CH	[DCE_CTS_A] PIO17	OD	**PCM Time Slot Control A** enables an external buffer device when channel A PCM Highway data is present on the [PCM_TXD_A] output pin in PCM Highway mode.

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
HDLC Channel B (PCM) CC CH			
[PCM_RXD_B] CC CH	[DCE_RXD_B] PIO36	STI	**PCM Receive Data Channel B** is the serial data input pin for the channel B PCM Highway interface.
[PCM_TXD_B] CC CH	[DCE_TXD_B] PIO37	O-LS-OD	**PCM Transmit Data Channel B** is the serial data output pin for the channel B PCM Highway interface.
[PCM_CLK_B] CC CH	[DCE_RCLK_B] PIO40	STI	**PCM Clock** is the single transmit and receive data clock pin for the channel B PCM Highway interface.
[PCM_FSC_B] CC CH	[DCE_TCLK_B] PIO41	STI	**PCM Frame Synchronization Clock** provides the Frame Synchronization Clock input (usually 8 kHz) for the channel B PCM Highway interface.
[$\overline{\text{PCM\_TSC\_B}}$] CC CH	[$\overline{\text{DCE\_CTS\_B}}$] PIO38	OD	**PCM Time Slot Control B** enables an external buffer device when channel B PCM Highway data is present on the [PCM_TXD_B] output pin in PCM Highway mode.
HDLC Channel C (PCM) CC			
[PCM_RXD_C] CC	[DCE_RXD_C] PIO42	STI	**PCM Receive Data Channel C** is the serial data input pin for the channel C PCM Highway interface.
[PCM_TXD_C] CC	[DCE_TXD_C] PIO43	O-LS-OD	**PCM Transmit Data Channel C** is the serial data output pin for the channel C PCM Highway interface.
[PCM_CLK_C] CC	[DCE_RCLK_C] PIO22	B	**PCM Clock:** For PCM Highway operation, [PCM_CLK_C] is the single transmit and receive data clock input pin for the channel C PCM Highway interface. [PCM_CLK_C] becomes a clock source output when the GCI to PCM Highway clock and frame synchronization conversion are enabled.
[PCM_FSC_C] CC	[DCE_TCLK_C] PIO23	B	**PCM Frame Synchronization Clock:** For PCM Highway operation, [PCM_FSC_C] provides the Frame Synchronization Clock input (usually 8 kHz) for the channel C PCM Highway interface. [PCM_FSC_C] becomes a frame synchronization source output when the GCI to PCM Highway clock and frame synchronization conversion are enabled.
[$\overline{\text{PCM\_TSC\_C}}$] CC	[$\overline{\text{DCE\_CTS\_C}}$] PIO44	OD	**PCM Time Slot Control C** enables an external buffer device when channel C PCM Highway data is present on the [PCM_TXD_C] output pin in PCM Highway mode.
HDLC Channel D (PCM) CC			
[PCM_RXD_D] CC	[RXD_U] (UART) DCE_RXD_D PIO26	STI	**PCM Receive Data Channel D** is the serial data input pin for the channel D PCM Highway interface.
[PCM_TXD_D] CC	[TXD_U] (UART) [DCE_TXD_D] PIO20	O-LS-OD	**PCM Transmit Data Channel D** is the serial data output pin for the channel D PCM Highway interface.
[PCM_CLK_D] CC	[$\overline{\text{RTR\_U}}$] (UART) DCE_RCLK_D PIO25	STI	**PCM Clock** is the single transmit and receive data clock pin for the channel D PCM Highway interface.

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
[PCM_FSC_D] CC	[$\overline{\text{CTS\_U}}$] (UART) [DCE_TCLK_D] PIO24	STI	**PCM Frame Synchronization Clock** provides the Frame Synchronization Clock input (usually 8 kHz) for the channel D PCM Highway interface.
[$\overline{\text{PCM\_TSC\_D}}$] CC	[$\overline{\text{CTS\_HU}}$] (High-Speed UART) [$\overline{\text{DCE\_CTS\_D}}$] PIO46	OD	**PCM Time Slot Control D** enables an external buffer device when channel D PCM Highway data is present on the [PCM_TXD_D] output pin in PCM Highway mode.
HDLC Channel A (GCI) CC			
[GCI_DD_A] CC	DCE_RXD_A [PCM_RXD_A]	BOD	**GCI Data Downstream** is the serial data input pin for the channel A GCI interface.
[GCI_DU_A] CC	DCE_TXD_A [PCM_TXD_A]	BOD	**GCI Data Upstream** is the serial data output pin for the channel A GCI interface.
[GCI_DCL_A] CC	DCE_RCLK_A [PCM_CLK_A]	STI	**GCI Data Clock** is the single transmit and receive channel A GCI data clock input generated by an upstream device. The data clock frequency must be twice the data rate.
[GCI_FSC_A] CC	DCE_TCLK_A [PCM_FSC_A]	STI	**GCI Frame Synchronization Clock** provides the 8-kHz Frame Synchronization Clock input for the channel A GCI interface generated by an upstream device.
UNIVERSAL SERIAL BUS (USB) CC CU			
[UDMNS] CC CU [UDPLS] CC CU	USBD– USBD+	STI STI	**USB External Transceiver Gated Differential Plus** and **USB External Transceiver Gated Differential Minus** are inputs from the external USB transceiver used to detect single-ended zero and error conditions. The signals have the following meanings: **USB External Transceiver Signals** UDPLS / UDMNS / Status: 0 / 0 / Single-Ended Zero (SE0) 0 / 1 / Full speed 1 / 0 / Reserved 1 / 1 / Error
USBD+ CC CU USBD– CC CU	[UDPLS] [UDMNS]	B B	**USB Differential Plus** and **USB Differential Minus** form the bidirectional electrical data interface for the USB port. The pins form a differential pair that can be connected to a physical USB connector without an external transceiver.
[USBSCI] CC CU	[UCLK] [USBSOF] PIO21	STI	**USB Sample Clock Input** is used to synchronize an external clock to the internal USB peripheral controller for isochronous transfers.
[USBSOF] CC CU	[UCLK] [USBSCI] PIO21	O	**USB Start of Frame** is a 1-kHz frame pulse used to synchronize USB isochronous transfers to an external device on a frame-by-frame basis.
UTXDMNS CC CU	RSVD_102	O	**USB External Transceiver Differential Minus** is an output that drives the external transceiver differential driver minus input.

Table 3-7 Signal Descriptions (Continued)

Signal Name[1]	Multiplexed Signal(s)	Type	Description
UTXDPLS CC CU	RSVD_101	O	**USB External Transceiver Differential Plus** is an output that drives the external transceiver differential driver plus input.
$\overline{\text{UXVOE}}$ CC CU	RSVD_103	O	**USB External Transceiver Transmit Output Enable** is an output that enables the external transceiver. $\overline{\text{UXVOE}}$ signals the external transceiver that USB data is being output by the microcontroller. When Low this pin enables the transceiver output, and when High this pin enables the receiver.
UXVRCV CC CU	RSVD_104	STI	**USB External Transceiver Differential Receiver** is a data input received from the external transceiver differential receiver.

Notes:
1. Icons indicating microcontroller specific signals are used only in the Signal Name column.

3.6 BUS INTERFACE

3.6.1 Overview

The Am186CC/CH/CU bus interface controls all accesses to the peripheral control block (PCB), memory-mapped and I/O-mapped external peripherals, and memory devices. The bus interface accesses internal peripherals through the PCB. The microcontroller provides an enhanced bus interface with the following features:

- Multiplexed address and data bus
- Nonmultiplexed address bus
- Option to disable the address phase of the address/data bus for accesses to the upper ($\overline{\text{UCS}}$) and/or lower ($\overline{\text{LCS}}$) memory regions
- Option to globally disable the address phase of the address/data bus for all memory or I/O accesses
- Programmable bus sizing, individually selectable for the upper ($\overline{\text{UCS}}$) memory space, lower ($\overline{\text{LCS}}$) memory space, all non-UCS and non-LCS memory space, and all I/O space
- Option to boot from an 8-bit device
- Separate byte write enables for high and low bytes
- $\overline{\text{RD}}$ signal can act as output enable
- Bus-mastering support of a PCnet-ISA interface by three-stating additional pins ($\overline{\text{UCS}}$, $\overline{\text{LCS}}$, $\overline{\text{MCS3}}$–$\overline{\text{MCS0}}$, $\overline{\text{PCS7}}$–$\overline{\text{PCS0}}$, and $\overline{\text{ALE}}$) during bus-hold
- Integrated DRAM controller
- Data Enable/Data Strobe ($\overline{\text{DEN}}$/$\overline{\text{DS}}$) and Data Transmit/Receive signal (DT/$\overline{\text{R}}$) provided to support an external data bus transceiver and to support a bus interface to 68xxx- style peripherals
- Support for the Reset Configuration (RESCON) register used to latch system configuration information from the AD bus during a power-on reset
- Peripheral Control Block Relocation (RELOC) register configurable to perform a “dual decode” of PCB addresses; one address is locked at the default reset location and the other address depends on how the RELOC register is programmed (default)

3.6.2 Block Diagrams

Figure 3-1 shows an Am186CC/CH/CU microcontroller system with DRAM; Figure 3-2, with SRAM.

Figure 3-1 Typical Microcontroller Memory System With DRAM

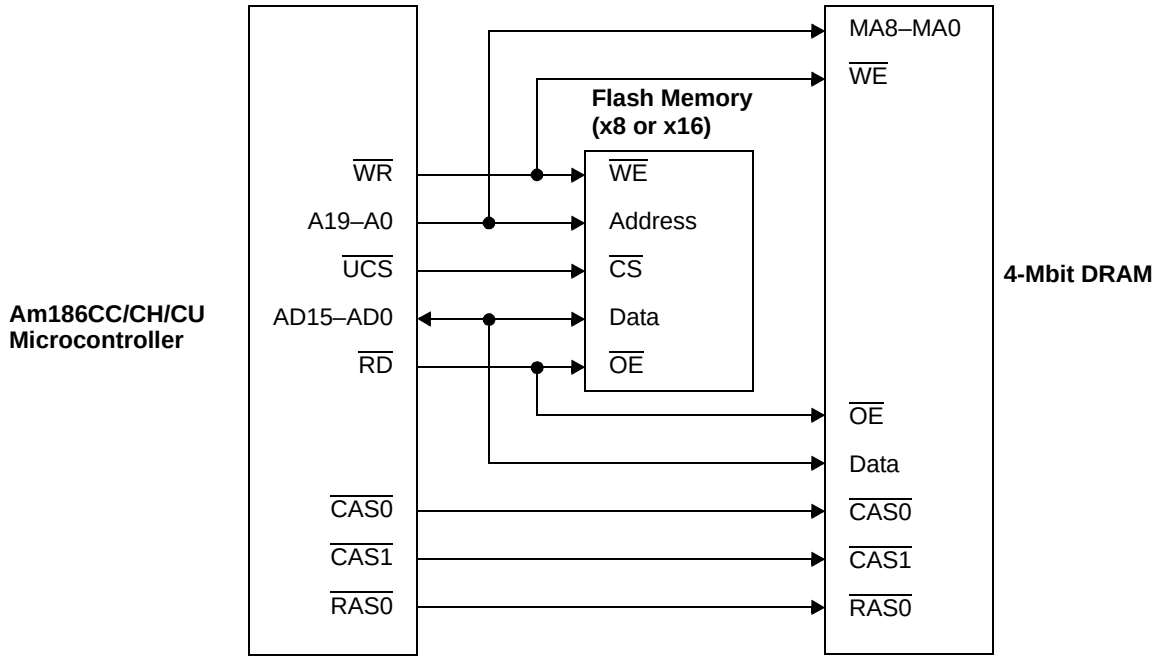
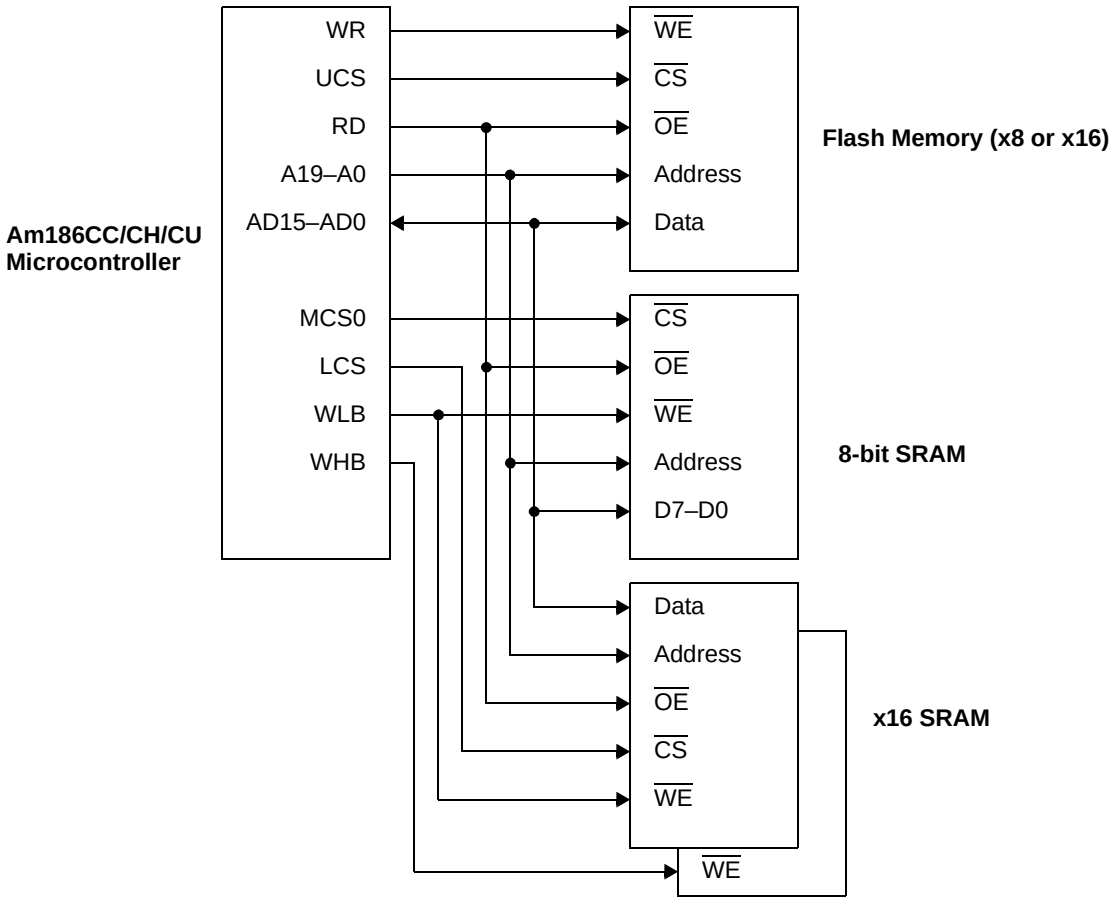


Figure 3-2 Typical Microcontroller Memory System With SRAM



3.6.3 Operation

3.6.3.1 Address and Data Buses

The 80C186 and 80C188 microcontrollers use a multiplexed address and data (AD) bus. The address is present on the AD bus only during the t_1 clock phase. The Am186CC/CH/CU microcontrollers provide the multiplexed AD bus and, in addition, provide a nonmultiplexed address (A) bus. The A bus provides an address to the system for the complete bus cycle. During refresh cycles, the AD bus is driven during the t_1 phase and the values are three-stated during the t_2, t_3, and t_4 phases. The value driven on the A bus is undefined during a refresh cycle.

The nonmultiplexed address bus (A19–A0) is valid one-half CLKOUT cycle in advance of the address on the AD bus. When used with the modified $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$ outputs and the byte write enable signals, the A19–A0 bus provides a seamless interface to external SRAM, DRAM, and Flash/EPROM memory systems.

For systems where power consumption is a concern, it is possible to disable the address from being driven on the AD bus on the microcontroller during the normal address portion of the bus cycle for accesses to $\overline{\text{RAS0}}$, $\overline{\text{RAS1}}$, upper ($\overline{\text{UCS}}$), and lower ($\overline{\text{LCS}}$) address spaces. In this mode, the affected bus is placed in a high-impedance state during the address portion of the bus cycle. This feature is enabled through the DA bits (bit 7) in the Upper Memory Chip Select (UMCS) and Lower Memory Chip Select (LMCS) registers. In addition, the DISMEM bit (bit 11, for memory addresses) and the DISIO bit (bit 10, for I/O addresses) in the SYSCON register serve as global address disables to prevent address bits from appearing on the AD15–AD0 bus. Setting the DISMEM bit overrides clearing the DA bits.

When address disable is in effect, the number of signals that assert on the bus during all normal bus cycles to the associated address space is reduced, thus decreasing power consumption, reducing processor switching noise, and preventing bus contention with memory devices and peripherals when operating at high clock rates. For more information about chip selects, see Chapter 5, “Chip Selects.”

If the $\overline{\text{ADEN}}$ pin is asserted during processor reset, the values of the DA, DISMEM, and DISIO bits are ignored and the address is driven on the AD bus for all accesses, thus preserving the industry-standard 80C186 and 80C188 microcontrollers’ multiplexed address bus and providing support for existing emulation tools.

For timing diagrams, see the data sheets for the Am186CC/CH/CU microcontrollers. For more information about the registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

3.6.3.2 Programmable Bus Sizing

The 80C186 microcontroller provided a 16-bit wide data bus over its entire memory and I/O address ranges, but did not allow accesses to an 8-bit wide bus. However, the data bus width on the Am186CC/CH/CU microcontrollers is programmable through the Upper Memory Chip Select (UMCS), Lower Memory Chip Select (LMCS), and PCS and MCS Auxiliary (MPCS) registers. The USIZ bit (bit 5) in the UMCS register determines the width of the data bus for memory accesses to the upper memory region and the LSIZ bit (bit 5) in the LMCS register determines the width for the lower memory region. The OMSIZ bit (bit 5) in the MPCS register specifies the width of the data bus for memory accesses to all non-upper and non-lower memory regions (i.e., $\overline{\text{MCS}}$ space, $\overline{\text{PCS}}$ space in memory, and the remaining memory space that does not reside in one of the enabled chip-select memory regions). The IOSIZ bit (bit 5) in the MPCS register specifies the width of the data bus for all I/O accesses. Table 3-8 shows how the bit settings affect bus size.

The width of the data access should not be modified while the processor is fetching instructions from the associated address space or while the peripheral control block is overlaid on the affected address space.

Table 3-8 Programming Am186CC/CH/CU Microcontrollers Bus Width

Space	Register	Bit	Value	Bus Width	Comments
$\overline{UCS}$	UMCS	USIZ	N/A	N/A	Dependent on boot option[1]
$\overline{LCS}$	LMCS	LSIZ	N/A	N/A	
$\overline{MCS}$	MPCS	OMSIZ	0	16 bits	Default
			1	8 bits	
$\overline{PCS}$[2]	MPCS	OMSIZ	0	16 bits	Default
			1	8 bits	
I/O[3]	MPCS	IOSIZ	0	16 bits	Default
			1	8 bits	
Other Memory[4]	MPCS	OMSIZ	0	16 bits	Default
			1	8 bits	

Notes:

1. $\overline{UCS}$ width on reset is determined by the $\overline{UCSX8}$ pin. If $\overline{UCSX8}$ is Low, the bus width is x8; if $\overline{UCSX8}$ is High, the bus is x16. If $\overline{UCS}$ boots as 8-bit space, it can be overridden by clearing the USIZ bit. If $\overline{UCS}$ boots as 16-bit space, it is not reconfigurable to 8-bit.

2. $\overline{PCS}$ space configured for memory only; not I/O.

3. If PCB space is mapped to I/O, its functions are not affected by this bit.

4. The remaining memory space that does not reside in one of the enabled, memory, chip-select regions. If PCB space is mapped to memory, its functions are not affected by this bit.

3.6.3.3 Byte Write Enables

The Am186CC/CH/CU microcontrollers provide two signals that act as byte write enables—$\overline{WHB}$ (Write High Byte, AD15–AD8) and $\overline{WLB}$ (Write Low Byte, AD7–AD0). $\overline{WHB}$ is the logical OR of $\overline{BHE}$ and $\overline{WR}$ ($\overline{WHB}$ is Low when both $\overline{BHE}$ and $\overline{WR}$ are Low). $\overline{WLB}$ is the logical OR of A0 and $\overline{WR}$ ($\overline{WLB}$ is Low when both A0 and $\overline{WR}$ are Low).

The byte write enables are driven with the nonmultiplexed address bus as required for the write timing requirements of common SRAMs.

3.6.3.4 Output Enable

The Am186CC/CH/CU microcontrollers provide the $\overline{RD}$ (Read) signal, which can act as an output enable for memory or peripheral devices. The $\overline{RD}$ signal is Low when the microcontroller reads a word or byte.

3.6.3.5 Bus Mastering

When an external bus master requests control of the local bus (by asserting HOLD), the microcontroller completes the bus cycle in progress. It then relinquishes control of the bus to the external bus master by asserting HLDA and floating $\overline{S2}$–$\overline{S0}$, AD15–AD0, S6, TMROUT1, and TMROUT0. During HOLD, internal pullups are active for $\overline{BHE}$, $\overline{DEN}$, DT/$\overline{R}$, $\overline{LCS}$, $\overline{MCS1}$/$\overline{CAS1}$, $\overline{MCS2}$/$\overline{CAS0}$, $\overline{MCS3}$/$\overline{RAS1}$, $\overline{PCS7}$–$\overline{PCS0}$, PIO4/$\overline{MCS0}$, $\overline{RD}$, $\overline{UCS}$, $\overline{WHB}$, $\overline{WLB}$, and $\overline{WR}$ and an internal pulldown is active for A19–A0 and ALE.

3.6.3.6 DRAM Controller

The microcontroller has a fully integrated DRAM controller that provides a glueless interface to 25-ns–70-ns EDO DRAM. The microcontroller provides zero-wait state operation at up to 50 MHz with 40-ns DRAM. The DRAM controller includes the following features:

- Multiplexed addresses for DRAM row and column accesses
- 8-bit and 16-bit boot mode for $\overline{\text{UCS}}$ accesses
- Two $\overline{\text{RAS}}$ signals that support two banks of DRAM
- Two byte $\overline{\text{CAS}}$ signals
- Direct support for 4-Mbit (256Kx16) extended data out (EDO) DRAMs
- Prioritized $\overline{\text{PCS}}$ over DRAM space accesses

The various cycles in the microcontroller follow this priority ranking: refresh (highest priority), HOLD, DMA, and CPU (lowest).

For more information about DRAM, see Chapter 6, “DRAM Controller.”

3.7 CLOCK CONTROL

The microcontroller clocks include the general system clock (CLKOUT), and the baud rate generator clock for the two Universal Asynchronous Receiver Transmitters (UART and high-speed UART). The Synchronous Serial Interface (SSI) and the timers (Timers 0, 1, and 2) derive their clocks from the system clock.

3.7.1 Clock Features

The microcontroller includes the following clock features and characteristics. Figure 3-3 illustrates the clocks. For detailed information on the clocks, see the data sheets for Am186CC/CH/CU microcontrollers.

- One crystal-controlled oscillator that uses an external fundamental mode crystal or oscillator to generate the system input clock.
- One internal PLL that generates a system clock (CLKOUT) that is 1x, 2x, or 4x the system input clock.
- SSI clock (SCLK) is derived from the system clock, divided by 2, 4, 8, 16, 32, 64, 128, or 256.
- Timers 0 and 1 can be configured to be driven by the timer input pins (TMRIN1, TMRIN0) or at one-fourth of the system clock. Timer 2 is driven at one-fourth of the CPU clock.
- UART clock can be derived from the internal system clock frequency or from the UART clock (UCLK) input.

CC **CU** The Am186CC and Am186CU microcontrollers also include the Universal Serial Bus (USB) clock with the following features:

- One independent crystal-controlled oscillator that uses an external fundamental mode crystal or oscillator to generate the USB input clock.
- One internal PLL that generates the 48-MHz clock required for the USB from either a 24-MHz or 12-MHz input.
- Single clock source operation possible by sharing the clock source between the system and the USB.

CC CH The Am186CC and Am186CH microcontrollers also include the transmitter/receiver clocks for each High-level Data Link Control (HDLC) channel.

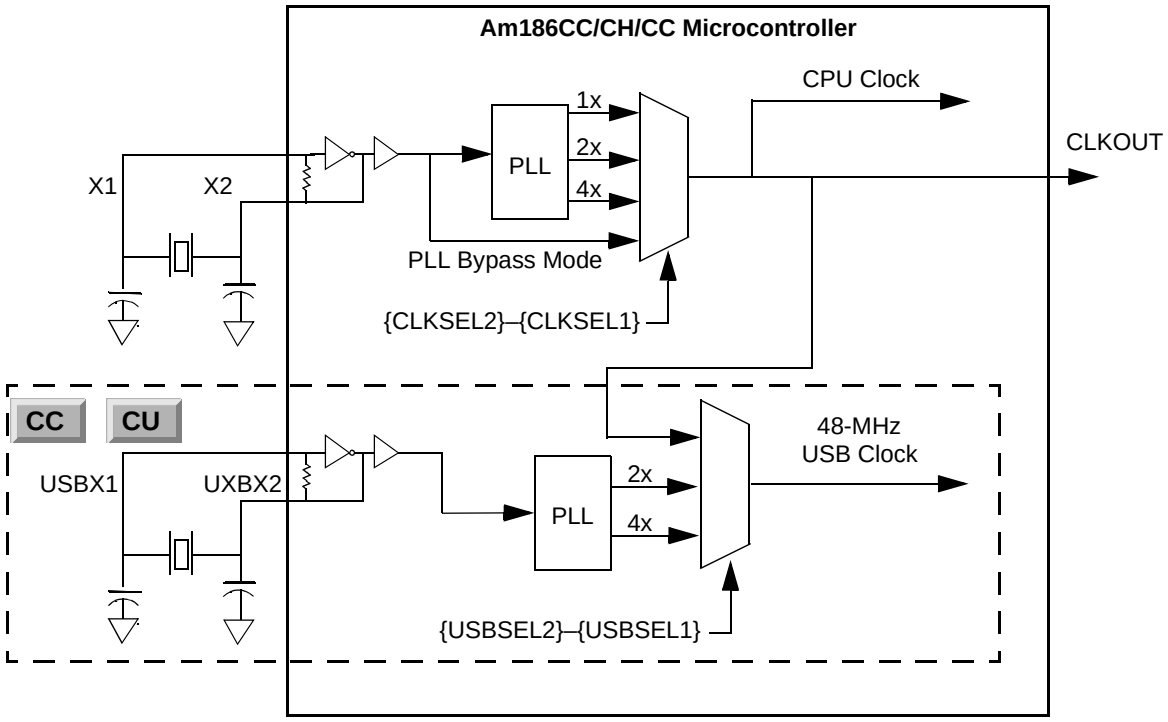
CC In the Am186CC microcontroller, each HDLC channel receives its clock inputs directly from the external communication clock pins (TCLK _X and RCLK_X) in all modes except in GCI mode. In GCI mode the external GCI communication clocks (TCLK_A and RCLK_A) are first converted to an internal clocking format (analogous to PCM Highway) before presentation to the HDLC. The system clock must be at least the same frequency as any HDLC clock. The Am186CC microcontroller supports the following clock frequencies:

- HDLC DCE mode supports clocks up to 10 MHz.
- HDLC PCM mode supports clocks up to 10 MHz.
- HDLC GCI mode supports a 1.536-MHz clock input. (System clock must be at least twice the GCI clock.)

CH In the Am186CH HDLC microcontroller, each HDLC channel receives its clock inputs directly from the external communication clock pins (TCLK _X and RCLK_X) in all modes. The system clock must be at least the same frequency as any HDLC clock. The Am186CH HDLC microcontroller supports the following clock frequencies:

- HDLC DCE mode supports clocks up to 10 MHz.
- HDLC PCM mode supports clocks up to 10 MHz.

Figure 3-3 Am186CC/CH/CU Microcontroller Clocks



3.7.2 PLL Bypass Mode

The Am186CC/CH/CU microcontrollers provide a PLL Bypass mode that allows the X1 input frequency to be anything from 0 to 24 MHz. Select PLL Bypass by asserting CLKSEL1 and CLKSEL2.

When the microcontroller is in PLL Bypass mode, the CLKOUT frequency equals the X1 input frequency. When changing frequency in PLL Bypass mode, the X1 input must not have any short or “runt” pulses. At 24 MHz, the nominal High/Low time is 21 ns. The actual High times and Low times must not fall below 16 ns. These values allow a 60%/40% duty cycle at X1.

CC **CU** In the Am186CC and Am186CU microcontrollers, the USB PLL and USBX1 determine the USB clock. USB requires the CPU clock to be 24 MHz or greater. Therefore, disable the USB peripheral controller before slowing the CPU clock to less than 24 MHz. If USB is not used, you can pull down USBX1.

CC **CH** In the Am186CC and Am186CH microcontrollers, the system clock must be at the same or a greater frequency than the HDLC clock and UCLK (if using UCLK). Therefore, if reducing the system clock frequency, disable these interfaces or run them at a lower frequency.

CC In the Am186CC microcontroller, the system clock must be the same or twice the frequency of the GCI clock. Therefore, if reducing the system clock frequency, disable the GCI interface or run it at a lower frequency.

3.8 HARDWARE-RELATED CONSIDERATIONS

- Pins latched on reset (pinstraps) are not resampled during a watchdog-timer reset.
- If the external reset ($\overline{RES}$) signal is asserted while the watchdog timer is performing a watchdog-timer reset, the external reset takes precedence over the watchdog-timer reset. This means that the RESOUT signal asserts as with any external reset and the WDTCON register does not have the RSTFLAG bit set. In addition, the microcontroller exits reset based on the external reset timing (i.e., 4.5 clocks after the deassertion of $\overline{RES}$ rather than 2^{16} clocks after the watchdog timer time-out occurred).

3.9 COMPARISON TO OTHER DEVICES

- The 80C186 microcontroller provided a 16-bit wide data bus over its entire address range, memory, and I/O, but did not allow accesses to an 8-bit wide bus. However, the data bus width on the Am186CC/CH/CU microcontrollers is programmable to be 8 bits or 16 bits.
- Earlier Am186 microcontrollers included a power save clock mode. The Am186CC/CH/CU microcontrollers are not designed for low-power applications and therefore do not incorporate the power save clock mode. However, the Am186CC/CH/CU microcontrollers do have a PLL Bypass mode that allows the X1 clock input frequency to be anything from 0 to 24 MHz.

3.10 INITIALIZATION

On both an external and internal reset, the following occurs:

- The SYSCON register defaults to 00h, which has the following effects: sets normal timing on $\overline{DEN}$ for read and writes, disables PWD mode, enables memory and I/O addresses on the AD15–AD0 bus, and enables CLKOUT.
- The PRL register defaults to the processor revision level.
- Multiplexed signals default as shown in Table 3-7 on page 3-10.

CC

- On the Am186CC microcontroller, both an external and an internal reset selects full HDLC with flow control for external interface D and sets HDLC Channel C for raw DCE or PCM Highway mode.

On an external reset, the following also occurs:

- Pinstraps are sampled (see Table 3-5 on page 3-7).
- The RESCON register defaults to the value on AD15–AD0.

CHAPTER

4 EMULATOR SUPPORT

4.1 OVERVIEW

This chapter describes the various features available in the Am186CC/CH/CU microcontrollers to facilitate the design and operation of an In-Circuit Emulator (ICE). Most of the discussion centers around the operation of pins. Because different debug tool manufacturers take different approaches to emulator implementation, restrictions imposed by the use of one type of emulator may not apply to another. However, there are a number of common concerns shared among ICE developers. This chapter discusses those concerns.

4.2 SYSTEM DESIGN

The main issues to consider are multiplexed pin use and emulator connection.

4.2.1 Multiplexed Pins

Because pins are an expensive resource, many of the pins on the Am186CC/CH/CU microcontrollers serve more than one purpose. These multiplexed pins enable the system designer to select, by hardware or software means, the required operation of the pin. It can often be difficult for an emulator to know the function of such multiplexed pins, particularly if the system modifies pin operation on-the-fly. Therefore, before committing a design to hardware, the system designer should contact potential emulator suppliers for a list of emulator pin requirements.

Certain pins are critical for successful emulator operation; these are address pins, chip selects, and memory access timing signals. It is important that these pins not be multiplexed in such a way as to compromise the emulator operation. Fortunately, several pin functions can be successfully multiplexed. Emulators generally do not monitor pins relating to input/output (PIO) operation and on-chip peripherals.

The Am186CC/CH/CU microcontrollers were designed to minimize conflicts. In most cases, pin conflict is avoided. For example, if the Address Latch Enable (ALE) signal is required for multiplex bus support, then it is not programmed as PIO33. If the multiplexed AD bus is used for data only (not addresses), then ALE can be programmed as a PIO pin and the emulator will not require the ALE signal. However, an emulator is likely to always use the de-multiplexed address, regardless of how the AD bus is programmed.

The following PIO signals are multiplexed with alternate signals that may be used by emulators: PIO8, PIO15, PIO33, PIO34, and PIO35. Consider any emulator requirements for the alternate signals before using these pins as PIOs.

4.2.2 Emulator Connection

Several package types present emulation problems. At the time of publication, the Am186CC/CH/CU microcontrollers ship in 160-pin PQFP packages.

When a PQFP device is soldered to a board, it cannot be removed and replaced with an emulator. In this situation, the CPU must be disabled somehow, and the emulator must be connected to the CPU to duplicate its functionality. The Am186CC/CH/CU microcontrollers do this with the On-Circuit Emulation (ONCE) mode. Placing the microcontroller in ONCE mode causes the output pins to become three-state and inactive. This feature allows a

designer to clip an emulator pod over the target CPU, then use ONCE mode to disable the target CPU and provide a connection to each of the PQFP processor pins. Be aware of any horizontal and vertical areas required by the emulators' physical attachment method, and plan the board layout accordingly. One common mistake is to place connectors, switches, or other board controls under an area that will be partially covered by the emulator target board. Also consider the arrangement of Pin 1 versus the emulator attachment and plan accordingly.

4.3 OPERATION

4.3.1 Usage

To use an emulator, the microcontroller must be put into ONCE mode. To enter ONCE mode, use the $\overline{\text{ONCE}}$ reset configuration pin (pinstrap). $\overline{\text{ONCE}}$ is sampled on the rising edge of $\overline{\text{RES}}$. If the $\overline{\text{ONCE}}$ pin is asserted, the microcontroller enters ONCE mode. Otherwise, it operates normally. In ONCE mode, all pins are three-stated and remain that way until a subsequent reset occurs. To ensure the microcontroller does not inadvertently enter ONCE mode, $\overline{\text{ONCE}}$ has a weak internal pullup resistor that is active only during an external reset.

Note: *Before using an emulator, ensure multiplexed pins are configured to reflect the use of the emulator and not other functionality.*

4.3.2 Emulator-Related Signals

4.3.2.1 A19–A0

To facilitate emulation, the Am186CC/CH/CU microcontrollers do not multiplex any of the A19–A0 address pins. Therefore, these pins are always available for emulation.

4.3.2.2 AD15–AD0

The Am186CC/CH/CU microcontrollers do not multiplex any AD15–AD0 address/data pins with other functionality, except that the value present on AD15–AD0 as the device comes out of external reset is latched and saved internally to the Reset Configuration (RESCON) register. Using this mechanism, a set of weak pullups and pulldowns can be put on the bus to allow hardware to communicate configuration information to the software. Because this is an input function, it should not interfere with the operation of the emulator. However, the emulator should not interfere with the value present at reset, as software may be relying on the value for proper operation.

4.3.2.3 {$\overline{\text{ADEN}}$} / $\overline{\text{BHE}}$

Deasserting $\overline{\text{ADEN}}$ on reset can prevent the multiplexed AD bus from providing address information for lower ($\overline{\text{LCS}}$) and upper ($\overline{\text{UCS}}$) memory regions. Some older ICE designs force $\overline{\text{ADEN}}$ active to force address information on the AD bus. System designers should be aware if their emulator uses this operation and any conflicts this can cause with their hardware.

186 processors use $\overline{\text{BHE}}$ along with A0 to determine the type and width of external bus accesses. 188's do not have $\overline{\text{BHE}}$, because all data on a 188 is 8 bits wide and routed through AD7–AD0. The Am186CC/CH/CU microcontrollers do not support a 188 version, but do allow defining memory regions as 8-bit memory. When making accesses to 8-bit wide memory regions, $\overline{\text{BHE}}$ cannot be used to derive any information about the access. Use the BSIZE8 signal to determine the width of a memory region unambiguously.

186 processors also use $\overline{BHE}$ with A0 to denote refresh cycles to 16-bit DRAM (both inactive). The Am186CC/CH/CU microcontrollers do not support 8-bit wide DRAM designs, so using this mechanism to determine refresh cycles is reliable under all allowed DRAM designs.

4.3.2.4 ALE

In multiplexed bus mode, ALE indicates that a valid address is on the AD bus. Some emulators may require this signal. In most instances, an active chip select signal can also be used to indicate a valid address.

4.3.2.5 ARDY and SRDY

If the target requires ready signals to operate, ARDY and SRDY cannot be used as PIOs. Some emulators give the user control over the external ready target requirement. For instance, ready may be required by the emulator to match overlay memory speeds to faster target wait-state setups.

4.3.2.6 $\overline{BHE}$

See “{$\overline{ADEN}$}/$\overline{BHE}$” on page 4-2.

4.3.2.7 $\overline{BSIZE8}$

The absence of $\overline{BHE}$ for 8-bit memory regions when an emulator design uses 16-bit overlay memory for a memory region defined as 8 bits wide poses problems for an emulator. The emulator must know when memory accesses are targeted at 8-bit regions to correctly steer the data between the low half of the data bus and the high half of the data bus. Although it is possible to snoop all events that determine the memory width (chip select pulldowns during reset, and UMCS, LMCS and MPCS register accesses), these methods can be unreliable. The Am186CC/CH/CU microcontrollers’ $\overline{BSIZE8}$ pin unambiguously signals the intended size of the memory region during external bus cycles.

4.3.2.8 [$\overline{CAS1}$–$\overline{CAS0}$] and [$\overline{RAS1}$–$\overline{RAS0}$]

The on-chip DRAM controller can be configured to work with DRAM in the lower ($\overline{LCS}$) or upper ($\overline{UCS}$) memory regions. The emulator needs to reconstruct the address used during an access. The $\overline{CAS}$ signal can come too late for fast address generation. However, the complete address appears on the A19–A0 bus during the $\overline{RAS}$ cycle. Additionally, because the full address bus is nonmultiplexed, it is a simple task to identify an access to the DRAM region.

$\overline{CAS}$-before-$\overline{RAS}$ cycles could also be used to determine if an access is a refresh, but the late arrival of the $\overline{RAS}$ signal makes this problematic.

The DRAM can only be accessed in 16-bit mode. This eliminates the problem of determining object size due to dynamic bus sizing.

4.3.2.9 CLKOUT

The internal processor clock can be sent out on the CLKOUT pin. Emulators generally require this.

4.3.2.10 $\overline{LCS}$

The system uses $\overline{LCS}$ as a RAM chip select. Emulators use this to determine when RAM accesses occur, and can intercept it for overlay memory purposes.

4.3.2.11 $\overline{MCS3}$–$\overline{MCS0}$

The system uses $\overline{MCS1}$ and $\overline{MCS2}$ as DRAM CAS strobes. $\overline{MCS0}$ and $\overline{MCS3}$ can be used as extra memory chip selects. Emulators can use these to determine when accesses occur to these memory spaces, and can intercept it for overlay memory purposes.

4.3.2.12 {$\overline{ONCE}$}

$\overline{ONCE}$ is not a dedicated pin but rather a pinstrap option that allows an external emulator to place a target device into On-Circuit Emulation mode. On reset of the microcontroller, if the $\overline{ONCE}$ pinstrap is held low, all Am186CC/CH/CU pins enter a high-impedance state. There is an internal pullup to prevent inadvertent assertion of $\overline{ONCE}$.

4.3.2.13 QS1–QS0

The Am186CC/CH/CU microcontrollers provide information about the execution queue on the Queue Status bus, QS1–QS0. These signals assist in disassembling trace buffer information.

4.3.2.14 [$\overline{RAS1}$–$\overline{RAS0}$]

See “[CAS1–CAS0] and [RAS1–RAS0]” on page 4-3.

4.3.2.15 $\overline{RD}$

The $\overline{RD}$ strobe can be intercepted by the emulator for use with overlay memory.

4.3.2.16 $\overline{RES}$

The Am186 processor family provides a Schmitt trigger on the $\overline{RES}$ input to enable the system designer to use an inexpensive RC circuit to provide system reset. The only restriction on power-up is for $\overline{RES}$ to stay active (Low) for at least 1 ms. Systems that use this feature introduced a problem for In-Circuit Emulators because emulators need to know when the target processor comes out of reset. This can be difficult to determine when the target is being placed in ONCE mode and the reset signal has a very slow rise time. Emulator vendors solve this problem by providing a reset signal with a fast rise time. The hardware designer must use this emulator-supplied reset instead of the standard RC reset circuit.

The Am186CC/CH/CU microcontrollers provide a RESOUT signal that unambiguously indicates when the device has come out of reset, eliminating this problem. However, many emulators still generate a target reset (in response to a user console command, for instance), and therefore need a means to connect the emulator-supplied reset to the target hardware.

Therefore, if ICE usage is required, be aware of the emulators’ reset requirements and take them into consideration when designing the target hardware, typically by providing a convenient means to allow the emulator-supplied reset to be the main system reset.

4.3.2.17 RESOUT

RESOUT is activated by the Am186CC/CH/CU microcontrollers in response to either $\overline{RES}$ being held active, or a system reset being generated by the internal watchdog timer. During reset, this pin is actively driven, regardless of the state of the $\overline{ONCE}$ mode pinstrap (in contrast, all other output pins go to three-state if both $\overline{RES}$ and $\overline{ONCE}$ are active). When $\overline{RES}$ is deasserted, RESOUT is driven inactive. This high-to-low edge on RESOUT is the signal that latches the value of all pinstrap options. When $\overline{ONCE}$ is active and $\overline{RES}$ is inactive, RESOUT is driven inactive (all other outputs are three-stated), and held Low for one clock cycle. After this one-clock period, RESOUT is three-stated. This sequence of events allows an attached emulator to determine with certainty that the device has entered ONCE mode.

4.3.2.18 $\overline{S2}$–$\overline{S0}$

The $\overline{S2}$–$\overline{S0}$ bus indicates the type of memory cycle in progress.

4.3.2.19 S6

The S6 signal is active from t_1–t_4 on the microcontroller and signals a refresh or DMA access.

4.3.2.20 SRDY

See “ARDY and SRDY” on page 4-3.

4.3.2.21 $\overline{UCS}$

The system typically uses $\overline{UCS}$ as a FLASH or ROM chip select. Emulators use this to determine when ROM accesses occur, and can intercept it for overlay memory purposes.

4.3.2.22 {$\overline{UCSX8}$} and $\overline{WLB}$

During processor reset, the $\overline{UCSX8}$ pin configures the upper memory region for 8-bit operation. The BSIZE8 signal unambiguously indicates the width of a memory region for a given access.

4.3.2.23 $\overline{WHB}$ and $\overline{WR}$

The emulator can intercept $\overline{WHB}$ and $\overline{WR}$ for use with overlay memory. Although most emulators use $\overline{S2}$–$\overline{S0}$ to determine cycle type, some may use the $\overline{WR}$ signal to determine when writes occur. This prevents the use of $\overline{WR}$ as a PIO when using the emulator.

4.3.2.24 $\overline{WLB}$

See {UCSX8} and WLBÿ

4.3.2.25 $\overline{WR}$

See “WHB and WR” .

4.3.3 Hardware-Related Considerations

- Be sure to allow room for pucks and emulator heads on your target board.
- The following PIO signals are multiplexed with alternate signals that may be used by emulators: PIO8, PIO15, PIO33, PIO34, and PIO35. Consider any emulator requirements for the alternate signals before using these pins as PIOs.

4.3.4 Comparison to Other Devices

- Previous Am186 watchdog timer implementations required the application to disable the watchdog timer to prevent watchdog time-outs while emulator code was executing. The Am186CC/CH/CU watchdog timer does not have this limitation. A feature of the watchdog timer allows ICE code to inhibit the count of the watchdog timer.

4.4 INITIALIZATION

On both external and internal reset, the following occurs:

- Multiplexed pins used in emulation default to signals shown in Chapter 3, “System Overview.”

CHAPTER

5 CHIP SELECTS

5.1 OVERVIEW

Signals that allow the CPU to select specific memory or peripheral devices are called *chip selects*.

The microcontroller provides six chip select outputs for use with memory devices ($\overline{UCS}$, $\overline{LCS}$, and $\overline{MCS3}$–$\overline{MCS0}$) and eight chip selects for use with peripherals ($\overline{PCS7}$–$\overline{PCS0}$) in either memory or I/O space. The six memory chip selects can be used to address three memory ranges. Each peripheral chip select addresses a 256-byte block offset from a programmable base address in memory or I/O.

The microcontroller can sense a ready signal for each of the memory or peripheral chip select lines. The R2 bit in each of the memory chip select control registers determines whether the external ready signal is required or ignored.

In addition, the R1–R0 bits in each of the memory chip select control registers control the number of wait states inserted in the bus cycle. Although most memory and peripheral devices can be accessed with three or fewer wait states, some slower devices cannot. This feature allows devices to use externally generated wait states to slow down the bus.

Address and data bus size options and enabling or disabling the address bus during the address phase of a bus cycle are configured on a chip select basis. $\overline{UCS}$ and $\overline{LCS}$ can also be configured for DRAM support.

The chip select lines are active for all memory and I/O cycles in their programmed areas, whether they are generated by the CPU or by the integrated DMA unit.

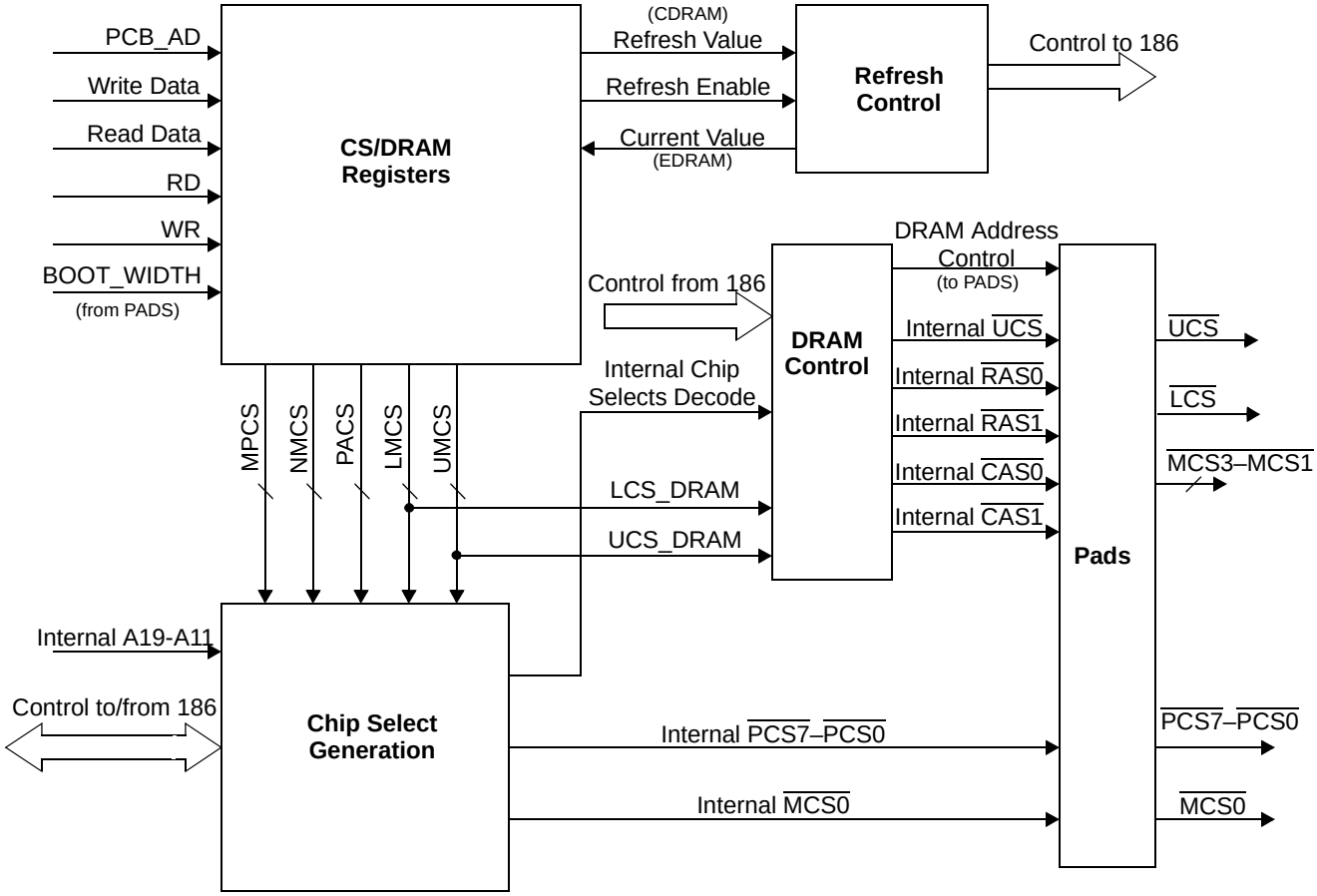
The $\overline{UCS}$ and $\overline{LCS}$ chip selects assert relative to the timing of the nonmultiplexed address (A) bus; the $\overline{MCS}$ and $\overline{PCS}$ chip selects assert relative to the multiplexed address and data (AD) bus. The timing for chip selects is shown in the data sheets for each of the Am186CC/CH/CU microcontrollers.

The $\overline{CAS0}$ and $\overline{CAS1}$ signals can be used to perform byte writes in a manner similar to $\overline{WLB}$ and $\overline{WHB}$, respectively. That is, $\overline{CAS0}$ corresponds to the low byte ($\overline{WLB}$) and $\overline{CAS1}$ corresponds to the high byte ($\overline{WHB}$).

5.2 BLOCK DIAGRAM

Figure 5-1 shows the block diagram for the chip selects.

Figure 5-1 Chip Selects and DRAM Block Diagram



5.3 SYSTEM DESIGN

Table 5-1 lists the chip select signals that are multiplexed with other Am186CC/CH/CU functions. Pinstraps are sampled only at external reset and do not affect the pin's other functions, so they are not shown in this table. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin.

For diagrams of some example applications, see Chapter 3, "System Overview."

Table 5-1 Chip Selects Multiplexed Signals

Signal	Multiplexed Signal(s)	Default Signal	Function
$\overline{\text{LCS}}$	$\overline{\text{RAS0}}$	$\overline{\text{LCS}}$	Lower memory chip select
$\overline{\text{MCS0}}$	PIO4	PIO4	Midrange memory chip selects
$\overline{\text{MCS1}}$	$\overline{\text{CAS1}}$	$\overline{\text{MCS1}}$	
$\overline{\text{MCS2}}$	$\overline{\text{CAS0}}$	$\overline{\text{MCS2}}$	
$\overline{\text{MCS3}}$	$\overline{\text{RAS1}}$ PIO5	PIO5	
$\overline{\text{PCS0}}$	PIO13	$\overline{\text{PCS0}}$	Peripheral chip selects
$\overline{\text{PCS1}}$	PIO14	$\overline{\text{PCS1}}$	
$\overline{\text{PCS2}}$	—	$\overline{\text{PCS2}}$	
$\overline{\text{PCS3}}$	—	$\overline{\text{PCS3}}$	
$\overline{\text{PCS4}}$	PIO3	PIO3	
$\overline{\text{PCS5}}$	PIO2	PIO2	
$\overline{\text{PCS6}}$	PIO32	PIO32	
$\overline{\text{PCS7}}$	PIO31	PIO31	
$\overline{\text{UCS}}$	—	$\overline{\text{UCS}}$	Upper memory chip select

5.4 REGISTERS

Program the chip selects through the five 16-bit peripheral registers (see Table 5-2). Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 5-2 Chip Select Register Summary

Offset	Register Mnemonic	Register Name	Description
3A0h	UMCS	Upper Memory Chip Select	Programs the lower boundary of the Upper Memory Chip Select, $\overline{\text{UCS}}$. Also supports DRAM.
3A2h	LMCS	Lower Memory Chip Select	Programs the upper boundary of the Lower Memory Chip Select, $\overline{\text{LCS}}$. Also supports DRAM.
3A4h	PACS	Peripheral Chip Select	Partially configures the peripheral chip selects, $\overline{\text{PCS7}}$–$\overline{\text{PCS0}}$ (along with the MPCS register). Sets the base address of the memory block selected by $\overline{\text{PCS}}$.
3A6h	MMCS	Midrange Memory Chip Select	Partially configures the Midrange Memory Chip Selects, $\overline{\text{MCS3}}$–$\overline{\text{MCS0}}$ (along with the MPCS register). Sets the base address of the memory block selected by $\overline{\text{MCS}}$.
3A8h	MPCS	$\overline{\text{PCS}}$ and $\overline{\text{MCS}}$ Auxiliary	Partially configures $\overline{\text{PCS7}}$–$\overline{\text{PCS0}}$ (along with the PACS register). Determines whether $\overline{\text{PCS}}$ chip selects are mapped to memory or I/O space. Also partially configures $\overline{\text{MCS3}}$–$\overline{\text{MCS0}}$ (along with the MMCS register). Sets the block size of the memory block selected by $\overline{\text{MCS}}$.

5.5 OPERATION

5.5.1 Usage

Note: *Before using the chip selects, ensure multiplexed pins are configured to reflect the use of the chip selects and not other functionality (see Table 5-1 on page 5-3).*

Except for the $\overline{\text{UCS}}$ chip select, which is active on reset, chip selects are not activated until the associated register is written (not when it is read). All these signals are three-stated during a bus-hold condition and during reset to allow an external bus master to drive these signals directly.

- To use the Upper Memory Chip Select ($\overline{\text{UCS}}$), configure the following UMCS register options:
  - Lower boundary of $\overline{\text{UCS}}$ (LB bit field)
  - AD bus disable (DA bit)
  - DRAM enable (UDEN bit)
  - Data bus width (USIZ bit)
  - External Ready mode (R2 bit)
  - Wait state value (R1 and R0 bits)

  $\overline{\text{UCS}}$ is active on reset.
- To use the Lower Memory Chip Select ($\overline{\text{LCS}}$), configure the following LMCS register options:
  - Upper boundary of $\overline{\text{LCS}}$ (UB bit field)
  - AD bus disable (DA bit)
  - DRAM enable (UDEN bit)
  - Data bus width (LSIZ bit)
  - External Ready mode (R2 bit)
  - Wait state value (R1 and R0 bits)

  $\overline{\text{LCS}}$ is activated when the LMCS register is written.
- To use the Peripheral Chip Select ($\overline{\text{PCS}}$), configure the following options in the PACS and MPCS registers:
  - Base address (BA bit field in PACS)
  - External Ready mode (R2 bit in PACS)
  - Wait state value (R0, R1, and R3 bits in PACS)
  - $\overline{\text{PCS}}$ mapped to memory or I/O (MS bit in MPCS)
  - Memory data bus width for all non-$\overline{\text{UCS}}$ and non-$\overline{\text{LCS}}$ memory (OMSIZ bit in MPCS)
  - I/O data bus width (IOSIZ bit in MPCS)

  The $\overline{\text{PCS}}$ chip selects are activated after both the PACS and MPCS registers are written.

- To use the Midrange Memory Chip Select ($\overline{MCS}$), configure the following options in the MMCS and MPCS registers:
  - Base address (BA[19–13] bit field in MMCS)
  - MCS0-Only mode (MCS0_ONLY bit in MMCS)
  - External Ready mode (R2 bit in MMCS)
  - Wait state value (R1 and R0 bits in MMCS)
  - $\overline{MCS}$ block size (M[6–0] bits in MPCS)
  - Memory data bus width for all non-UCS and non-LCS memory (OMSIZ bit in MPCS)

  The $\overline{MCS}$ chip selects are activated after both the MMCS and MPCS registers are written.

Note: *To configure the bus width for memory that does not reside in the $\overline{LCS}$ or $\overline{UCS}$ chip-select memory regions, program the OMSIZ bit in the MPCS register. To configure the bus width for I/O space, program the IOSIZ bit in the MPCS register.*

5.5.2 Selecting Memory and I/O Space

All the chip selects can refer to addresses in memory. Only the $\overline{PCS}$ chip selects can reference I/O space. Figure 5-2 on page 5-6 shows which part of memory each chip select can address. The $\overline{MCS}$ chip selects should not be configured to overlap with memory space used by $\overline{UCS}$, $\overline{LCS}$, or $\overline{PCS}$. Figure 5-3 on page 5-7 shows the I/O space $\overline{PCS7}$–$\overline{PCS0}$ can select.

5.5.2.1 $\overline{UCS}$

The Am186CC/CH/CU microcontrollers provide the $\overline{UCS}$ chip select for the top of the 1-Mbyte memory address space. The upper boundary is FFFFFh; the lower boundary is programmable with the LB bit field in the UMCS register. The block size must be a multiple of 64 Kbyte.

5.5.2.2 $\overline{LCS}$

The $\overline{LCS}$ chip select is for the bottom of the 1-Mbyte memory address space. The lower boundary is 00000h; the upper boundary is programmable with the UB bit field in the LMCS register. The block size must be a multiple of 64 Kbyte.

5.5.2.3 $\overline{MCS3}$–$\overline{MCS0}$

$\overline{MCS3}$–$\overline{MCS0}$ provide for a user-locatable memory block. The base address can reside anywhere in the 1-Mbyte memory address space as long as the base is an integer multiple of the block size (0 is a valid multiple), and memory space is not already mapped to by $\overline{UCS}$, $\overline{LCS}$ (unless they are mapped to DRAM), or $\overline{PCS}$.

The Am186CC/CH/CU microcontrollers also offer MCS0 Only mode. When the MCS0-ONLY bit in the MMCS register is cleared (the default) and the $\overline{MCS}$ chip selects are enabled, $\overline{MCS3}$–$\overline{MCS0}$ are each asserted over one fourth of the total block size. When this bit is set and the $\overline{MCS}$ chip selects are enabled, $\overline{MCS0}$ is asserted over the entire $\overline{MCS}$ address range, and $\overline{MCS3}$–$\overline{MCS1}$ are still asserted over their individual address ranges. This means the entire middle chip select range is selectable through $\overline{MCS0}$; the remaining $\overline{MCS}$ pins are available for other functions. This mode is useful if only one chip select is required or if DRAM is selected. For more information, see “Selecting DRAM Using the Chip Selects” on page 5-7.

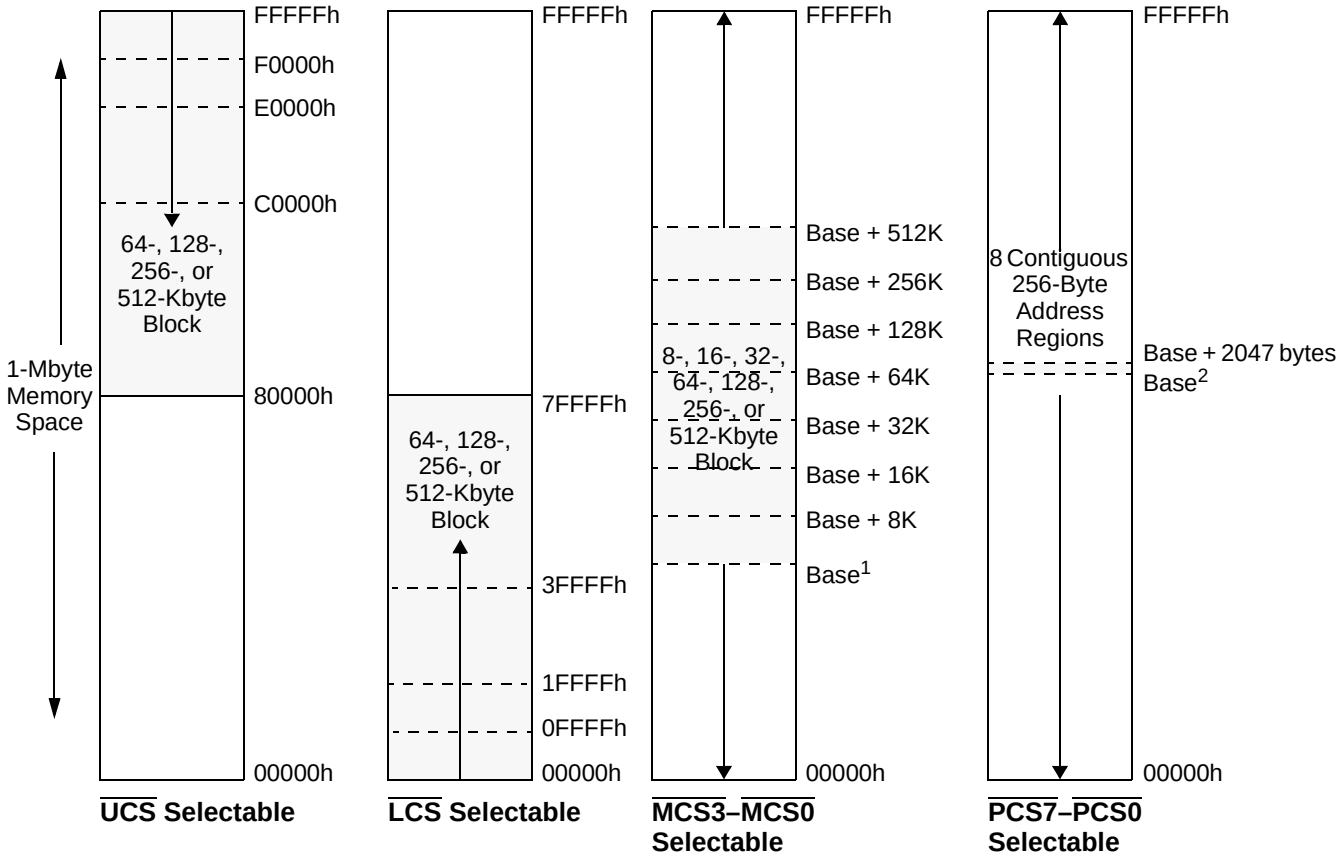
The BA bit field in the MMCS register programs the base address; the M[6–0] bits in the MPCS register program the total block size; the MCS0_ONLY bit in the MMCS register enables MCS0 Only mode.

5.5.2.4 $\overline{PCS7}$–$\overline{PCS0}$

The Am186CC/CH/CU microcontrollers each provide eight chip selects for eight contiguous, user-locatable, 256-byte address ranges within memory or I/O space. The base address can reside anywhere in the 1-Mbyte memory address space as long as it is a multiple of 2 Kbytes (0 is a valid multiple), and the memory space is not already mapped to by $\overline{UCS}$, $\overline{LCS}$, or $\overline{MCS}$. (The $\overline{PCS}$ address range can overlap the $\overline{UCS}$ or $\overline{LCS}$ address ranges if they are mapped to DRAM.) The $\overline{PCS}$ chip selects can also access the 64-Kbyte I/O space, as long as the base address is a multiple of 2 Kbytes.

The $\overline{PCS}$ chip selects are programmable with two registers. The BA bit field of the PACS register sets the base address (0 is a valid address). If the chip selects are programmed to reside in the CPU's I/O space, bits BA[19–16] are forced to 0 by hardware, as the upper bound of the CPU's I/O space is 64 Kbytes. The MS bit in the MPCS register determines whether $\overline{PCS}$ chip selects are mapped to memory or I/O space.

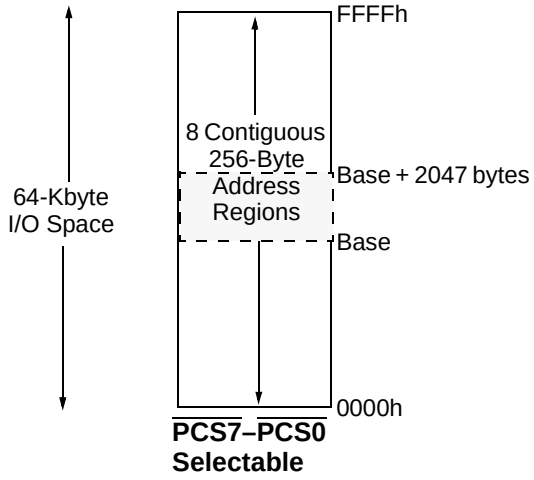
Figure 5-2 Chip Selectable Memory Space



Notes:

1. *Base must be an integer multiple of the block size and can be anywhere in memory space from 00000h to FDFFFh, as long as memory space is not already mapped to by $\overline{UCS}$, $\overline{LCS}$, or $\overline{PCS}$.*
2. *Base must be a multiple of 2 Kbytes and $\overline{PCS}$ memory region must not be configured to overlap with $\overline{MCS}$ space or non-DRAM $\overline{LCS}$ or $\overline{UCS}$ space.*

Figure 5-3 Chip Selectable I/O Space



5.5.3 Selecting DRAM Using the Chip Selects

UCS and LCS can be configured for DRAM support with the UDEN bit in the UMCS register and the LDEN bit in the LMCS register, respectively. PSRAM is not supported. If both UCS and LCS are configured for DRAM, up to two banks of 256 Kbit x 16 DRAM can be accessed. Neither, either, or both DRAM banks can be activated.

Table 5-3 shows how the signals are configured when either UCS or LCS is configured for DRAM.

Table 5-3 Signal Function When UCS or LCS is Configured for DRAM

Signal	Function
UCS configured for DRAM	
MCS1[1]	Acts as upper Column Address Strobe signal (CAS1)
MCS2	Acts as lower Column Address Strobe signal (CAS0)
MCS3	Acts as upper Row Address Strobe signal (RAS1)
UCS	UCS is held High. This means any memory device that uses UCS is disabled. This permits the user to disable a nonvolatile memory device providing boot-up code and replace it with DRAM memory.
LCS configured for DRAM	
LCS	Acts as lower Row Address Strobe signal (RAS0)
MCS1	Acts as upper Column Address Strobe signal (CAS1)
MCS2	Acts as lower Column Address Strobe signal (CAS0)

Notes:

1. Even if MCS3–MCS1 can no longer be used as chip selects, the MCS0 signal can select the entire middle chip select range when MCS Only mode is enabled. Also, the MCS3–MCS1 pins are multiplexed with programmable I/O pins. To enable their DRAM functionality, the PIO Mode and Direction registers must be cleared. For more information, see Chapter 9, "Programmable I/O Signals."

PCS7–PCS0 can overlap any UCS or LCS space which has been configured for DRAM. (Overlap of the PCS signals with UCS or LCS in non-DRAM mode is not recommended.) Overlapping PCS with DRAM is fully supported as long as the PCS chip selects are programmed for a greater or equal number of wait states than that of the DRAM.

Note: *Because of how the DRAM access is terminated, it is illegal to allocate a PCS space with fewer wait states than the DRAM it is overlapping.*

If PCS overlaps LCS or UCS configured for DRAM, PCS access takes precedence over the LCS or UCS access. The DRAM controller asserts RAS and stops the CAS signal from asserting. This does not modify the contents of the DRAM, and the access continues as a normal PCS access.

Overlapping the PCS chip selects with DRAM makes a 2-Kbyte block of the DRAM inaccessible. In its place, the peripherals associated with the PCS can be accessed. This is especially useful when the entire memory space is used with two banks of DRAM or a bank of DRAM and a 512-Kbyte Flash memory.

5.5.4 Overlapping Chip Selects

Although programming the various chip selects on the Am186CC/CH/CU microcontrollers so that multiple chip select signals are asserted for the same physical address is not recommended, it may be unavoidable in some systems. Note that configuring PCS in I/O space with LCS or any other chip select configured for memory address 0 is not considered overlapping of the chip selects. Overlapping chip selects refers to configurations where more than one chip select asserts for the same physical address. PCS overlaps are allowed when UCS or LCS are configured for DRAM. For more information about this overlapping, see “Selecting DRAM Using the Chip Selects” on page 5-7.

In systems where the chip selects must overlap, the chip selects whose assertions overlap must have the same configuration for ready (external ready required or not required) and for the number of wait states to be inserted into the cycle by the processor.

The peripheral control block (PCB) is accessed using *internal* signals. These internal signals function as chip selects configured with zero wait states and no external ready. Therefore, the PCB can reside at addresses that overlap *external* chip select signals if those external chip selects are programmed to zero wait states with no external ready required.

When overlapping an additional chip select with either the LCS or UCS chip selects, note that setting the Disable Address (DA) bit in the LMCS or UMCS register disables the address from being driven on the AD bus for all accesses for which the associated chip select is asserted, including any accesses for which multiple chip selects assert.

The MCS and PCS chip select pins can be configured as either chip selects or as PIO inputs or outputs. However, the ready and wait state generation logic for these chip selects is in effect regardless of their configurations as chip selects or PIOs. This means that if these chip selects are enabled (by a write to the MMCS and MPCS registers for the MCS chip selects, or by a write to the PACS and MPCS registers for the PCS chip selects), the ready and wait state programming for these signals must agree with the programming for any other chip selects with which their assertion would overlap if they were configured as chip selects.

Failure to configure overlapping chip selects with the same ready and wait state requirements may cause the processor to hang with the appearance of waiting for a ready signal. This behavior can occur even in a system in which ready is always asserted (ARDY or SRDY tied High).

5.5.5 Configuring Address and Data Buses

5.5.5.1 $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$

When $\overline{\text{UCS}}$ or $\overline{\text{LCS}}$ are asserted, the DA bit in the UMCS or LMCS register selects whether the AD15–AD0 bus is driven during the address phase of a bus cycle.

The DA bit is still valid when $\overline{\text{UCS}}$ or $\overline{\text{LCS}}$ supports DRAM (either UDEN or LDEN is 1). That is, even though the $\overline{\text{UCS}}$ signal is held High and the $\overline{\text{LCS}}$ signal becomes $\overline{\text{RAS0}}$ in DRAM mode, the address phase on AD15–AD0 is still disabled during DRAM accesses to $\overline{\text{UCS}}$/$\overline{\text{LCS}}$ space if DA is set to 1. In addition, the DISMEM (for memory addresses) and DISIO (for I/O addresses) bits in the SYSCON register can act as global address disables to prevent address bits from appearing on the AD15–AD0 bus. Setting the DISMEM bit overrides clearing the DA bits. The block size programmed should match the size of the DRAM being used, otherwise the full capacity of the DRAM is not utilized.

The $\overline{\text{UCSX8}}$ signal is sampled during every external reset. If $\overline{\text{UCSX8}}$ is 0, the LSIZ/USIZ bit is set in the LMCS/UMCS register, which defines memory as an 8-bit space. This allows the microcontroller to boot from an 8-bit wide device. It is possible to later clear this bit, thus redefining the space to be 16 bits wide. Only a hard system reset can cause this bit to be set; therefore, it is only possible to go from an 8-bit to a 16-bit space through software and not the reverse. If the system does a watchdog timer reset, this bit reverts to the value sampled on $\overline{\text{UCSX8}}$ during the last external reset. The $\overline{\text{UCSX8}}$ signal has a weak pullup that defaults the part into 16-bit operation. If DRAM is enabled for $\overline{\text{UCS}}$ or $\overline{\text{LCS}}$, the bus size is forced to 16 bits. For more information about controlling the bus width, see Table 3-8 on page 3-31.

5.5.5.2 Non-$\overline{\text{UCS}}$ and Non-$\overline{\text{LCS}}$

The OMSIZ bit determines the width of the data bus (i.e., x8 or x16) for memory accesses between the $\overline{\text{LCS}}$ and $\overline{\text{UCS}}$ memory regions (i.e., accesses above the $\overline{\text{LCS}}$ region and below the $\overline{\text{UCS}}$ region).

An $\overline{\text{MCS}}$ space cannot overlap $\overline{\text{LCS}}$ or $\overline{\text{UCS}}$ memory, so it always lies in the space affected by the OMSIZ bit. A $\overline{\text{PCS}}$ space is only affected by the OMSIZ bit if the $\overline{\text{PCS}}$ space is in memory and does not overlap an $\overline{\text{LCS}}$ or $\overline{\text{UCS}}$ region. If a $\overline{\text{PCS}}$ space overlaps an $\overline{\text{LCS}}$ or $\overline{\text{UCS}}$ region, the $\overline{\text{PCS}}$ space is accessed as x16 memory.

If the PCB space resides in an x8 memory region, each word-wide PCB register access generates two external bus cycles, but all 16 register bits are accessed internally on the first cycle. For more information, see “Peripheral Registers” on page 2-4.

5.5.5.3 $\overline{\text{PCS}}$ I/O Space

The IOSIZ bit in the MPCS register determines the width of the data bus (x8 or x16) for all I/O accesses.

If the PCB space is mapped to I/O and the I/O bus width is x8, each word-wide PCB register access generates two external bus cycles, but all 16 register bits are accessed internally on the first cycle. For more information, see “Peripheral Registers” on page 2-4.

5.5.6 Programming Ready Signals and Wait States

The Am186CC/CH/CU microcontrollers can sense a ready signal for each of the peripheral or memory chip select lines. The ready signal can be either the ARDY or SRDY signal. Each chip select control register (UMCS, LMCS, PACS, MMCS, and MPCS) contains a single-bit field, R2, that determines whether the external ready signal is required or ignored. When R2 is set to 1, external ready is ignored. When R2 is cleared to 0, external ready is required.

The number of wait states to be inserted for each access to a peripheral or memory region is programmable. 0–3, 5, 7, 9, or 15 wait states can be inserted for the $\overline{\text{PCS7}}$–$\overline{\text{PCS4}}$ peripheral chip selects.

Note: *Because of how the DRAM access is terminated, it is illegal to allocate a $\overline{\text{PCS}}$ space with fewer wait states than the DRAM it is overlapping.*

Zero to three wait states can be inserted for all other chip selects. Two bits, R1 and R0, in each of the chip select control registers program the wait states. The PACS register also has the R3 bit for the additional $\overline{\text{PCS}}$ wait states.

When external ready is required (R2 is 0), internally programmed wait states always complete before external ready terminates or extends a bus cycle. For example, if the internal wait states are set to insert two wait states (R1–R0 = 10b), the processor samples the external ready signal during the first wait cycle. If external ready is asserted at that time, the access completes after six cycles (four cycles plus two wait states). If external ready is not asserted during the first wait cycle, the access is extended until ready is asserted, which is followed by one more wait state followed by t_4.

When external readys are ignored (R2 is 1), the R1 and R0 bits alone configure the number of wait states. If DRAM is enabled for $\overline{\text{UCS}}$ or $\overline{\text{LCS}}$, external readys are ignored regardless of the setting of R2.

The ARDY signal on the Am186CC/CH/CU microcontrollers is a true asynchronous ready signal. The ARDY signal accepts a rising edge that is asynchronous to CLKOUT and is active High. If the falling edge of ARDY is not synchronized to CLKOUT as specified, an additional clock period may be added.

5.5.7 Chip Select Timing

The timing for the $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$ outputs has been modified from the original 80C186 microcontroller. These outputs now assert with the nonmultiplexed address bus (A19–A0) for normal memory timing. To allow these outputs to be available earlier in the bus cycle, the number of programmable memory size selections has been reduced.

The $\overline{\text{MCS}}$ and $\overline{\text{PCS}}$ chip selects assert with the AD bus.

For more information about chip select timing, see the data sheets for the Am186CC/CH/CU microcontrollers.

5.5.8 Hardware-Related Considerations

- The $\overline{\text{LCS}}$ memory space supports use of either the DRAM interface or the SRAM interface, not both.

5.5.9 Software-Related Considerations

- The chip selects are activated only by a write to a register. Chip selects on previous Am186 devices activated with a read or a write.
- The UMCS, LMCS, and MPCS registers contain a new data bus width bit; therefore, legacy code may accidentally change the bus width when writing to these registers.

5.5.10 Comparison to Other Devices

- General enhancements over the original 80C186 include bus mastering (three-state) support for all chip selects, and activation only when the associated register is written, not when it is read. In addition, each peripheral chip select asserts over a 256-byte address range, which is twice the address range covered by peripheral chip selects in the 80C186.
- The chip selects for the Am186CC/CH/CU microcontrollers are similar to the Am186EM and Am186ES microcontroller implementations except that the $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$ space is now capable of gluelessly supporting DRAM.
- The chip selects are activated by a write to a register. Chip selects on previous Am186 devices activated with a read or a write.
- The Am186CC/CH/CU microcontrollers offer eight peripheral chip selects rather than the six in other Am186 implementations.
- Unlike previous Am186 designs, $\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$ cannot be modified to provide latched address bits A1 and A2.
- Unlike previous Am186 products, no refresh information is ever provided on $\overline{\text{MCS3}}$.
- Unlike the Am186EM and Am186ES products, the Am186CC/CH/CU microcontrollers do not support PSRAM mode.
- Data bus width is programmable to x8 or x16. This feature was not previously available on Am186 devices.

5.6 INITIALIZATION

On both an external and internal reset, the following occurs:

- The microcontroller begins fetching and executing instructions starting at memory location FFFF0h, so upper memory is typically used as instruction memory. To facilitate this usage, $\overline{\text{UCS}}$ defaults to active on reset.
- The $\overline{\text{LCS}}$, $\overline{\text{MSC3}}$–$\overline{\text{MCS0}}$, and $\overline{\text{PCS7}}$–$\overline{\text{PCS0}}$ signals are *not* active on reset; activation requires a write access to the applicable memory chip select control register.
- The $\overline{\text{MCS0}}$, $\overline{\text{MCS3}}$, and $\overline{\text{PCS7}}$–$\overline{\text{PCS4}}$ signals default to PIOs. See Table 5-1 on page 5-3 for the multiplexed pin defaults.
- The value of the UMCS register defaults to F0*x*Bh, where *x* is 00*y*1b and *y* is the inverted state of $\overline{\text{UCSX8}}$ that was latched upon exiting reset. This defaults $\overline{\text{UCS}}$ to a 64-Kbyte memory block starting at F0000h, with the AD bus enabled, $\overline{\text{UCS}}$ DRAM disabled, external ready, and three wait states. This action allows the $\overline{\text{UCS}}$ memory region to function as a non-DRAM bank so a system can boot from a nonvolatile memory device before software switches this memory region to a DRAM bank.
- The value of the LMCS register is set to 0F1Bh, which defaults $\overline{\text{LCS}}$ to a 64-Kbyte memory block ending at 0FFFFh, with the AD bus enabled, $\overline{\text{LCS}}$ DRAM disabled, external ready, and three wait states. However, the $\overline{\text{LCS}}$ chip select is not enabled until software writes to the LMCS register.
- The value of the PACS register is set to 0073h, and the MPCS register is set to 8183h, which defaults $\overline{\text{PCS}}$ to a 256-byte block in I/O space starting at 0h, with external ready and three wait states. However, the $\overline{\text{PCS}}$ chip select is not enabled until software writes to both the PACS and MPCS registers.

- The value of the MMCS register is set to 7FDBh and the MPCS register is set to 8183h, which defaults MCS3–MCS0 each to 2 Kbytes with a total MCS block size of 8 Kbytes at a base address of 3Fh, with external ready, and three wait states. However, the MCS chip selects are not enabled until software writes to both the MMCS and MPCS registers.
- Data bus widths are set as follows:
  - LCS is 16 bits wide.
  - Non-UCS and non-LCS memory (MCS, PCS, and the remaining memory that does not reside in one of the enabled, memory chip-select regions) accesses are 16 bits wide.
  - All I/O accesses are 16 bits wide.
- UCS is the inverse of the state of the UCSX8 that was latched on exiting external reset. If UCSX8 is 0, UCS is 8 bits wide; if UCSX8 is 1, UCS is 16 bits wide. In either case, UCS defaults to non-DRAM.

CHAPTER

6 DRAM CONTROLLER

6.1 OVERVIEW

Dynamic Random Access Memory (DRAM) offers memory at moderate speed and low cost. DRAM memory cells consist of one transistor and one capacitor. DRAM also uses a multiplexed address bus in a row/column format, which results in a lower pin count and smaller device package.

DRAM is volatile; that is, if the capacitors for the memory cells are not periodically recharged, the contents of memory is lost. The process of periodically recharging the capacitors is called refresh.

The DRAM controller's purpose is to use the processor's address, status, and control lines to generate the multiplexed address strobes. The Row Address Strobe ($\overline{RAS}$) and Column Address Strobe ($\overline{CAS}$) signals latch the row and column addresses inside the DRAM.

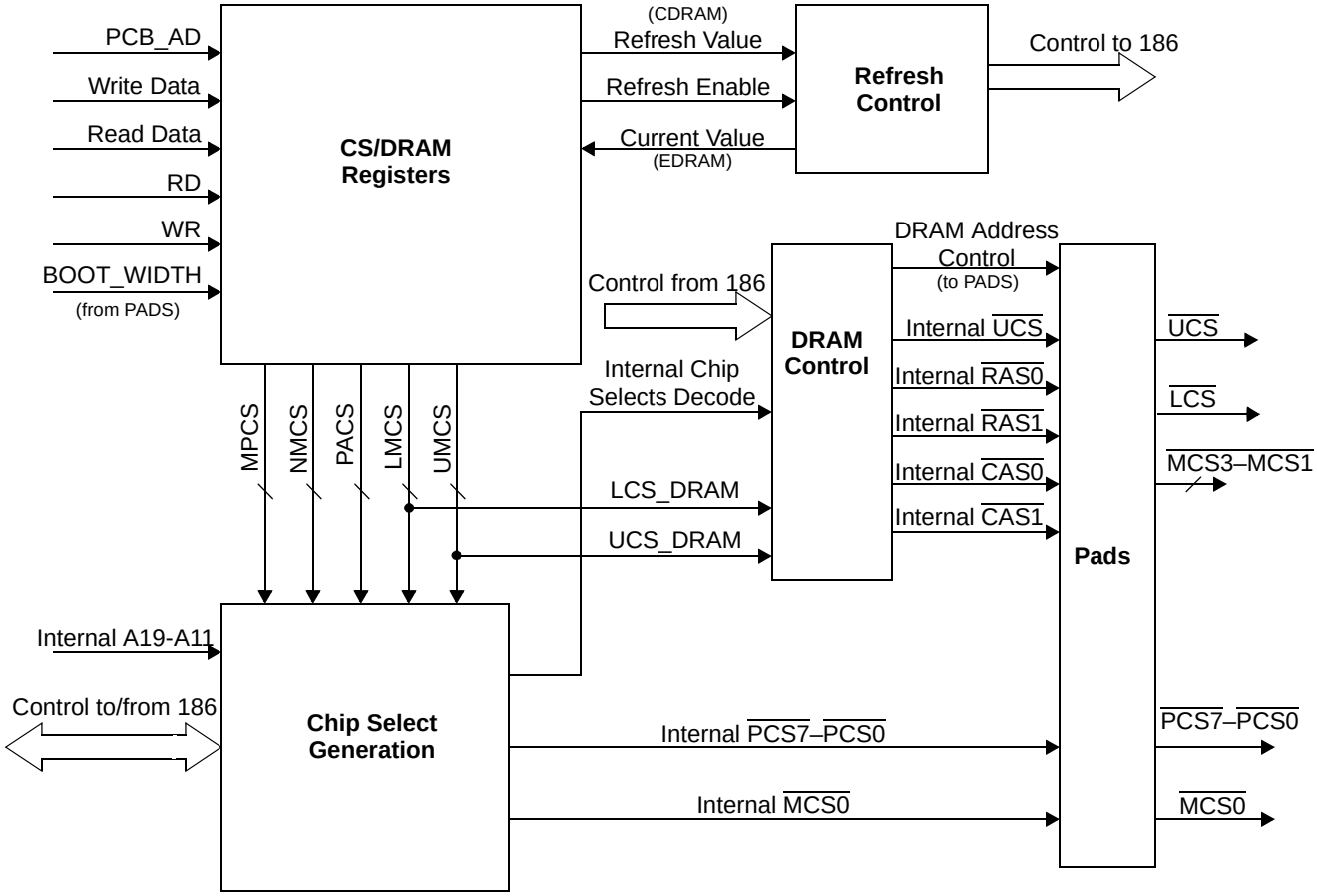
To support DRAM, the Am186CC/CH/CU microcontrollers each have a fully integrated DRAM controller that provides a glueless interface to 40-ns, 50-ns, 60-ns, and 70-ns Extended Data Out (EDO) DRAM (EDO DRAM is sometimes called Hyper-Page Mode DRAM). Up to two banks of 4-Mbit (256 Kbit x 16 bit) DRAM can be accessed. Page Mode, Fast Page Mode (FPM), Asymmetrical, and 8-bit wide DRAM are not supported.

The Am186CC/CH/CU microcontrollers support the most common DRAM refresh option, CAS-Before-RAS. All refresh cycles contain three wait states to support the DRAMs at various frequencies. The DRAM controller never performs a burst access. All accesses are single accesses to DRAM. If the $\overline{PCS}$ chip selects are decoded to be in the DRAM address range, $\overline{PCS}$ accesses take precedence over the DRAM.

6.2 BLOCK DIAGRAM

Figure 6-1 shows the block diagram for the DRAM controller.

Figure 6-1 Chip Selects and DRAM Block Diagram (Same as Figure 5-1)



6.3 SYSTEM DESIGN

Table 6-1 lists the DRAM signals that are multiplexed with other functions. Pinstraps are sampled only at external reset and do not affect the pin's other functions, so they are not shown in this table. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin.

For diagrams of some example applications, see Chapter 3, "System Overview."

Table 6-1 DRAM Multiplexed Signals

Signal	Multiplexed Signal(s)	Default Signal	Function
$\overline{\text{CAS0}}$	$\overline{\text{MCS2}}$	$\overline{\text{MCS2}}$	Column address strobes
$\overline{\text{CAS1}}$	$\overline{\text{MCS1}}$	$\overline{\text{MCS1}}$	
$\overline{\text{RAS0}}$	$\overline{\text{LCS}}$	$\overline{\text{LCS}}$	Row address strobes
$\overline{\text{RAS1}}$	$\overline{\text{MCS3}}$ PIO5	PIO5	

6.4 REGISTERS

Table 6-2 lists the 16-bit peripheral registers that determine the operation of the DRAM controller. You must also program the LDEN bit of the LMCS register and the UDEN bit of the UMCS register for DRAM operation. Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 6-2 DRAM Controller Register Summary

Offset	Register Mnemonic	Register Name	Description
3AAh	CDRAM	Refresh Clock Prescaler	Used to configure the DRAM refresh rate.
3ACh	EDRAM	Enable Refresh Control	Used to enable the refresh counter. It can also be used to sample the present value of the refresh down counter.

6.5 OPERATION

6.5.1 Usage

Note: *Before using the DRAM controller, ensure the multiplexed pins listed in Table 6-1 on page 6-2 (PIOs, chip selects, and DRAM) are configured to reflect the use of the DRAM controller and not other functionality.*

To enable DRAM support for the Am186CC/CH/CU microcontrollers, use the following process:

1. Configure the $\overline{\text{UCS}}$ or $\overline{\text{LCS}}$ chip selects for DRAM. For information, see "Selecting DRAM Using the Chip Selects" on page 5-7.
2. Set the RC bit field in the CDRAM register to the DRAM refresh rate. This is the number of CPU clocks between refresh cycles. All refresh cycles contain three wait states to accommodate the various DRAMs supported. Note that changing the value of this field after DRAM refresh has been enabled does not load the new value into the refresh counter until the current counter value has reached 0.
3. Set the EN bit of the EDRAM register to 1 to enable DRAM refresh.

6.5.2 DRAM Supported

The Am186CC/CH/CU microcontrollers support one or two banks of 40-ns, 50-ns, 60-ns, or 70-ns, 4-Mbit (256 Kbit x 16 bit), symmetrical Extended Data Out (EDO) DRAM (EDO DRAM is sometimes called Hyper-Page Mode DRAM).

Eight-bit (byte-wide) DRAM is not supported, and the DRAM does not operate properly if configured as an 8-bit area. However, it is still possible to perform byte accesses to 16-bit DRAM. Simply perform a 16-bit read and choose the upper or lower byte as needed.

The Am186CC/CH/CU microcontrollers can boot from a nonvolatile memory device in $\overline{\text{UCS}}$ space and later switch the $\overline{\text{UCS}}$ space to a DRAM. The microcontrollers also support an 8-bit $\overline{\text{UCS}}$ boot mode, which allows the user to boot from an 8-bit device and later switch to 16-bit operation. It is not possible to boot from a 16-bit memory device and later switch to an 8-bit device. See Chapter 5, "Chip Selects," for details.

Table 6-3 shows the wait states used to support DRAM.

Table 6-3 DRAM Supported by the Am186CC/CH/CU Microcontrollers

CPU Clock Speed	DRAM Speed	Wait States	Refresh Cycles
25 MHz	50 ns	0	7 clocks
	60 ns		7 clocks
	70 ns		7 clocks
40 MHz	50 ns	0	7 clocks
	60 ns	1	7 clocks
	70 ns	2	7 clocks
50 MHz	40 ns	0	7 clocks
	50 ns	1	7 clocks
	60 ns	2	7 clocks
	70 ns	3	7 clocks

6.5.3 DRAM Interface

The microcontroller provides zero-wait state operation at up to 50 MHz with 40-ns DRAM. Internal wait states can be inserted to support slower DRAM; however, external ready detection is not supported. All signals required by the DRAM are generated on the microcontroller and no external logic is required.

The DRAM multiplexed address pins are connected to the odd address pins starting with A1 on the microcontroller to MA0 on the DRAM (see Table 6-4). The correct row and column addresses are generated on these pins during a DRAM access. Table 6-4 shows how the physical address bits are mapped to row and column addresses on external pins.

The $\overline{\text{CAS0}}$ and $\overline{\text{CAS1}}$ signals select which byte of the DRAM is accessed during a read or write. The $\overline{\text{RAS0}}$ signal controls the lower bank of DRAM, which starts at 00000h in the address map and is bounded by the ending address selected with the UB bit field in the LMCS register. The $\overline{\text{RAS1}}$ signal controls the upper bank of DRAM, which ends at FFFFFh and is bounded by the starting address selected in the LB bit field in the UMCS register. When $\overline{\text{RAS1}}$ is asserted, $\overline{\text{UCS}}$ is automatically deasserted. Neither, either, or both DRAM banks can be activated.

Table 6-4 Address Multiplexing Reference

Am186CC/CH/CU Address Pin	DRAM Address Pin	Row	Column
A1	MA0	PA1	PA2
A3	MA1	PA3	PA4
A5	MA2	PA5	PA6
A7	MA3	PA7	PA8
A9	MA4	PA9	PA10
A11	MA5	PA11	PA12
A13	MA6	PA13	PA14
A15	MA7	PA15	PA16
A17	MA8	PA17	PA18

The user can re-enable $\overline{UCS}$ by clearing the UDEN bit in the UMCS register. Doing so disables refreshing the upper bank of DRAM. If the data in the upper bank of DRAM does not have to be retained, no special action is required. If the data in the upper bank of DRAM must be retained, two options are available. The refresh control unit counter can be monitored through the EDRAM register. When the counter reaches all zeros, a refresh occurs. The user can then disable the upper bank of DRAM using the UDEN bit in the UCMS register, access the $\overline{UCS}$-connected device, and then re-enable the upper bank of DRAM before the next refresh is scheduled to occur (usually 15.6 µs). This retains the data in the upper bank of DRAM.

Alternatively, a software routine can conduct a read from all rows of the upper DRAM. Then the UDEN bit can be switched to enable $\overline{UCS}$ and disable $\overline{RAS1}$. The user then has the total refresh time (usually 16 ms) before the DRAM must be re-enabled to retain its data. After re-enabling the DRAM, the user should once again conduct reads on all the DRAM row addresses before letting the refresh controller resume refreshing the DRAM.

6.5.4 Option to Overlap DRAM with $\overline{PCS}$

The $\overline{PCS7}$–$\overline{PCS0}$ signals can overlap DRAM blocks with different wait states without external or internal bus contention. The $\overline{RAS0}$ or $\overline{RAS1}$ signals assert along with the appropriate $\overline{PCS}$ signal. The $\overline{CAS0}$ and $\overline{CAS1}$ signals do not assert, preventing the DRAM from writing erroneously or driving the data bus during a read. The $\overline{PCS}$ signals must be configured to have the same or greater number of wait states than the DRAM. In the case of an overlap, the bus width during $\overline{PCS}$ accesses is 16 bits.

6.5.5 DRAM Refresh

6.5.5.1 DRAM Refresh Cycle

When DRAM refresh is enabled, it operates off the processor internal clock. The following steps outline the refresh process:

1. The Refresh Control unit (RCU) checks the T bit field in the EDRAM register to see if the counter = 0. If not, the clock decrements by 1 and the counter is checked again. This process is repeated until the counter = 0.
2. When the refresh counter = 0, the counter reloads the value from the RC field of the CDRAM register and starts again, simultaneously generating a CAS-before-RAS request to the bus interface unit. The DRAM refresh process continues until the EN bit in the EDRAM register is cleared.
3. The bus interface acknowledges the request. The refresh request stays active until the bus becomes available.
4. When the bus is free, the bus interface runs a “dummy read” cycle. Note that the refresh clock counter continues counting independent of when the bus interface services the refresh request. If the HLDA signal is active when a refresh request is generated (indicating a bus hold condition), then the microcontroller deactivates the HLDA signal to perform a refresh cycle when the hold is negated. The circuit external bus master must negate the HOLD signal for at least one clock to allow the refresh cycle to execute. The refresh cycle has priority over all other bus cycles (CPU, DMA, and so on). Refresh changes no bits and looks like a read cycle. The various cycles follow this priority ranking: refresh (highest priority), HOLD, DMA, and CPU (lowest).
5. After the refresh cycle completes, the HLDA signal goes active and the controller continues with whatever activity was occurring before the refresh.
6. The request is removed.

6.5.5.2 DRAM Refresh Intervals

During a refresh cycle, the AD bus drives the address to FFFFh, which prevents the $\overline{PCS}$ and $\overline{MCS}$ signals from asserting inadvertently. $\overline{PCS}$ and $\overline{MCS}$ decode should never contain the address FFFFFh. The $\overline{UCS}$ signal does not assert during a refresh cycle. If two banks of DRAM are being used in a system (i.e., $\overline{RAS0}$ and $\overline{RAS1}$), then both banks are refreshed at the same time.

The interval counter (CDRAM register and EDRAM register) is expanded by two bits over earlier Am186 microcontrollers. The refresh counter has a maximum timer count that reaches 163.9 µs at 50 MHz. See Table 6-5 and Equation 6-1.

The normal refresh rate on a DRAM is 15.6 µs. This refresh rate allows for each of the 1024 row addresses to be refreshed in the required 16 ms. Some DRAMs might have different refresh rates for low-power DRAMs and special considerations. Table 6-5 demonstrates the typical values that a programmer might want to use for refresh time intervals to be placed into the RC bit field of the CDRAM register.

The Am186CC/CH/CU microcontrollers support DRAMs with a CAS-before-RAS refreshing scheme. A refresh is generated based on the system clock frequency. The maximum count value for a refresh is 163.9 µs at 50 MHz. The CAS-before-RAS refresh cycle is seven clock cycles long. An 11-bit counter inserts a refresh bus cycle after the last bus cycle concludes to run the CAS-before-RAS cycle.

Table 6-5 Refresh Interval Times

CPU Frequency	Clock Period	CDRAM Counter (hex)	CDRAM Counter (decimal)	Refresh Interval Time
50 MHz	20 ns	30Ch	780d	15.6 µs
40 MHz	25 ns	270h	624d	15.6 µs
25 MHz	40 ns	186h	390d	15.6 µs

Equation 6-1 Refresh Interval Time Equation

$$\text{CDRAM Counter Value (Decimal)} = \frac{\text{Refresh Interval Time}}{\text{Clock Period}}$$

6.5.6 Hardware-Related Considerations

- The $\overline{LCS}$ memory space supports use of either the DRAM interface or the SRAM interface, not both.
- An external bus master needs to be able to deassert HOLD in response to HLDA going inactive for DRAM refresh cycles to take place.

6.5.7 Software-Related Considerations

Do not program the refresh period too small. If you do, the system does not have time to execute code.

6.5.8 Comparison to Other Devices

The DRAM controller is similar to the Am186ED DRAM controller, with these primary enhancements: 50 MHz, extended refresh interval times, and faster DRAMs. The Am186CC/CH/CU microcontrollers support 25-ns to 70-ns EDO DRAM only. It does not support Fast Page mode DRAM.

6.6 INITIALIZATION

On both an external and internal reset, the following occurs:

- The value of the CDRAM register becomes 0000h, setting the DRAM refresh period to 0.
- The value of the EDRAM register becomes 0000h, clearing and disabling the refresh counter.
- The UDEN bit of the UMCS register and the LDEN bit of the LMCS register both become 0, disabling $\overline{\text{UCS}}$ and $\overline{\text{LCS}}$ DRAM. See Chapter 5, “Chip Selects.”.
- The $\overline{\text{CAS}}$ and $\overline{\text{RAS}}$ multiplexed pins default to their alternate functions as shown in Table 6-1 on page 6-2.
- $\overline{\text{UCS}}$ is the inverse of the state of the $\overline{\text{UCSX8}}$ that was latched on exiting external reset. If $\overline{\text{UCSX8}}$ is 0, $\overline{\text{UCS}}$ is 8 bits wide; if $\overline{\text{UCSX8}}$ is 1, $\overline{\text{UCS}}$ is 16 bits wide. In either case, $\overline{\text{UCS}}$ defaults to non-DRAM.

CHAPTER

7 INTERRUPTS

7.1 OVERVIEW

An interrupt is a request to the CPU for service. CPUs receive interrupt requests from a variety of sources, both internal and external. When the CPU receives a request, it stops executing the current task, and if the new task is of higher priority, begins executing that routine. At the end of the routine, the CPU returns to the original task.

Some interrupts can be disabled. These are called *maskable* interrupts. *Nonmaskable* interrupts cannot be disabled.

The Am186CC/CH/CU microcontrollers feature an interrupt controller, which arranges the maskable interrupt requests by priority and presents them one at a time to the CPU. In addition to interrupts managed by the interrupt controller, the microcontroller supports eight nonmaskable interrupts—an external or internal nonmaskable interrupt (NMI), a trace interrupt, and software interrupts and exceptions.

The interrupt controller supports the maskable interrupt sources through the use of 15 channels. To make this possible, most interrupt channels support multiple interrupt sources. These channels are programmable to support the external interrupt pins or various peripheral devices that can be configured to generate interrupts. The maskable interrupt sources include 17 external sources plus a number of internal sources.

CC The Am186CC microcontroller has 19 internal maskable interrupt sources.

CH The Am186CH HDLC microcontroller has 14 internal maskable interrupt sources.

CU The Am186CU USB microcontroller has 13 internal maskable interrupt sources.

CC The following Am186CC microcontroller peripherals can generate internal interrupts:

- Three on-board timers (two of the timers can operate as pulse width modulators)
- Two UARTs
- Four HDLC channels
- The GCI
- Four pairs of transmit/receive SmartDMA channels
- Four general-purpose DMA channels
- The USB peripheral controller

CH

The following Am186CH HDLC microcontroller peripherals can generate internal interrupts:

- Three on-board timers (two of the timers can operate as pulse width modulators)
- Two UARTs
- Two HDLC channels
- Two pairs of transmit/receive SmartDMA channels
- Four general-purpose DMA channels

CH

The following Am186CU USB microcontroller peripherals can generate internal interrupts:

- Three on-board timers (two of the timers can operate as pulse width modulators)
- Two UARTs
- Two pairs of transmit/receive SmartDMA channels
- Four general-purpose DMA channels
- The USB peripheral controller

System configuration determines which of these devices and signals are available as interrupt sources. In addition to these internal interrupts, nine interrupt signals and eight PIOs can be configured as external interrupt sources.

An NMI can be generated externally or internally. An external NMI is generated with the NMI signal. An internal NMI is generated by the microcontroller's watchdog timer. For more information on the watchdog timer, see Chapter 11, "Watchdog Timer."

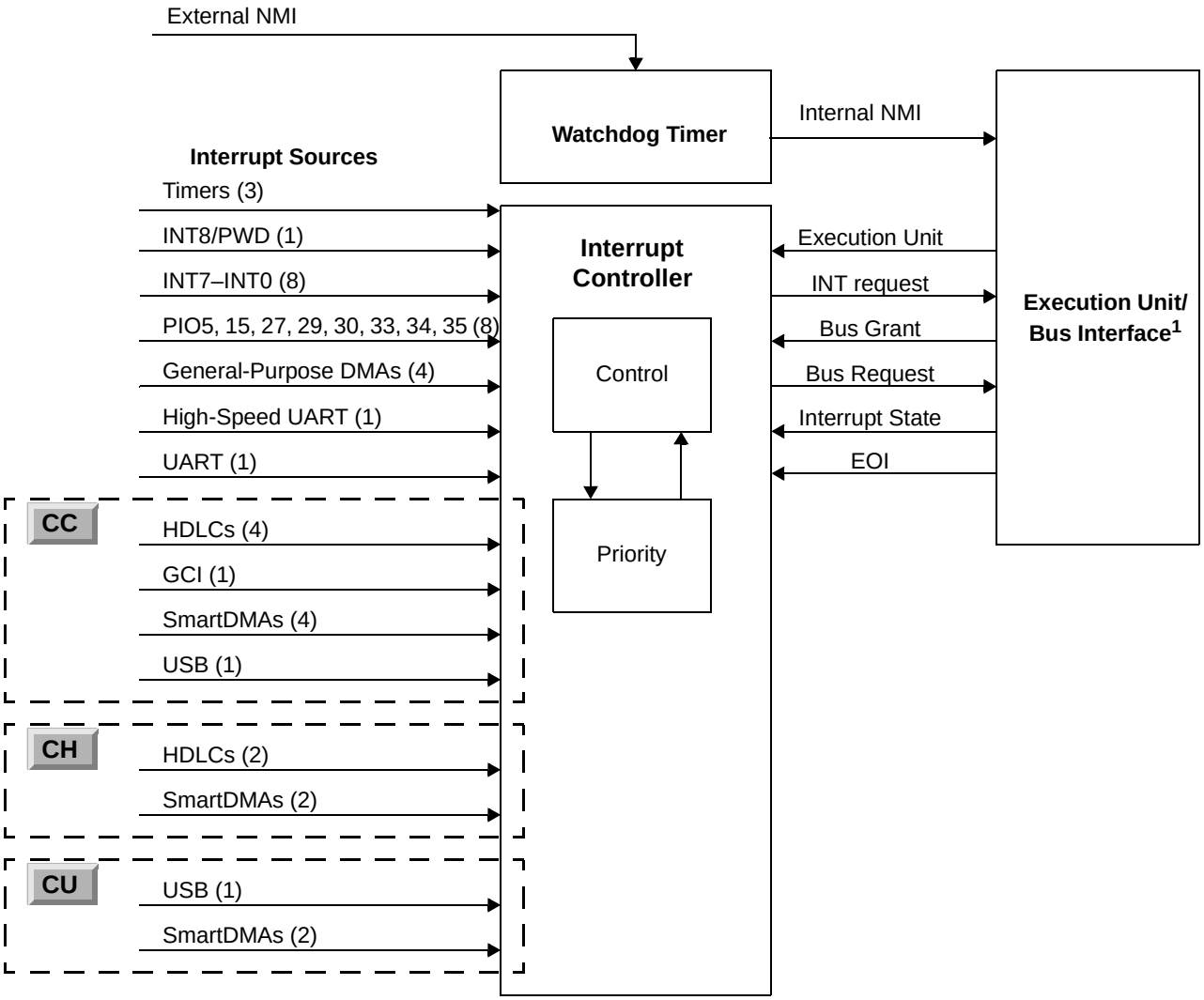
A trace interrupt is generated with the trace flag (TF bit) in the Processor Status Flags (FLAGS) register. See Chapter 2, "Configuration Basics."

Software can also generate interrupts and exceptions. A software interrupt is generated with the INT or INTO instruction; a software exception is an interrupt resulting from an error condition after executing any instruction. Software interrupt and exception sources are: divide error exception, breakpoint interrupt, INTO detected overflow exception, array bounds exception, unused opcode exception, and ESC opcode exception.

7.2 BLOCK DIAGRAM

Figure 7-1 shows how the microcontroller supports interrupts. The interrupt controller is the interface between the execution unit and all the peripheral interrupt requests and external interrupt signals. The watchdog timer can generate an NMI when a time-out value is reached. Software can determine whether an NMI was generated externally or internally by reading the RSTFLAG and EXRST bits in the Watchdog Timer Control (WDTCON) register.

Figure 7-1 Interrupts Block Diagram



Notes:

1. Software interrupt and traps are generated and resolved within the execution unit.

7.3 SYSTEM DESIGN

Table 7-1 lists the interrupt signals that are multiplexed with other microcontroller functions. Pinstraps are sampled only at external reset and do not affect the pin's other functions, so they are not shown in this table. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin.

For diagrams of some example applications, see Chapter 3, "System Overview."

Table 7-1 Interrupt Multiplexed Signals

Signal	Multiplexed Signal(s)	Default Signal	Function
INT0	—	INT0	Maskable interrupt requests
INT1	—	INT1	
INT2	—	INT2	
INT3	—	INT3	
INT4	—	INT4	
INT5	—	INT5	
INT6	PIO19[1]	PIO19[1]	
INT7	PWD[2], PIO7[1]	PIO7[1]	
INT8	PWD[2], PIO6[1]	PIO6[1]	
PIO5[1]	$\overline{\text{RAS1}}$, $\overline{\text{MCS3}}$	$\overline{\text{RAS1}}$	
PIO15[1]	$\overline{\text{WR}}$	$\overline{\text{WR}}$	
PIO27[1]	TMRIN0	PIO27	
PIO29[1]	DT/$\overline{\text{R}}$	DT/$\overline{\text{R}}$	
PIO30[1]	$\overline{\text{DEN}}$/$\overline{\text{DS}}$	$\overline{\text{DEN}}$	
PIO33[1]	ALE	ALE	
PIO34[1]	$\overline{\text{BHE}}$, $\overline{\text{ADEN}}$	$\overline{\text{BHE}}$	
PIO35[1]	SRDY	SRDY	
NMI	—	NMI	Nonmaskable interrupt

Notes:

1. For information about using PIOs as external interrupt sources, see "PIOs as Interrupts" on page 7-18.

2. Selected by setting the PWD bit in the SYSCON register.

7.4 REGISTERS

Table 7-2 lists the registers used by the microcontroller for interrupts. In addition, the IF flag in the Processor Status Flags (FLAGS) processor register is used to enable or disable interrupts (see "Registers Used" on page 7-18). Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 7-2 Interrupt Controller Register Summary

Offset	Register Mnemonic	Register Name	Description
300h	CH0CON	Interrupt Channel 0 Control	Configures one of the 15 interrupt channels.
302h	CH1CON	Interrupt Channel 1 Control	
304h	CH2CON	Interrupt Channel 2 Control	
306h	CH3CON	Interrupt Channel 3 Control	
308h	CH4CON	Interrupt Channel 4 Control	
30Ah	CH5CON	Interrupt Channel 5 Control	
30Ch	CH6CON	Interrupt Channel 6 Control	
30Eh	CH7CON	Interrupt Channel 7 Control	
310h	CH8CON	Interrupt Channel 8 Control	
312h	CH9CON	Interrupt Channel 9 Control	
314h	CH10CON	Interrupt Channel 10 Control	
316h	CH11CON	Interrupt Channel 11 Control	
318h	CH12CON	Interrupt Channel 12 Control	
31Ah	CH13CON	Interrupt Channel 13 Control	
31Ch	CH14CON	Interrupt Channel 14 Control	
320h	EOI	End-Of-Interrupt	Used to clear the in-service bit of an interrupt that is currently in service.
322h	POLL	Poll	Indicates the interrupt type of the highest priority pending interrupt.
324h	POLLST	Poll Status	Copy of the POLL register. Reading the Poll Status register has no effect on the rest of the system.
326h	IMASK	Interrupt Mask	Contains the mask bits for interrupt channels 0–14. These are the same physical mask bits that exist in all of the Channel Control registers, but here all channels are accessible at one time.
328h	PRIMSK	Priority Mask	Determines the minimum priority required for a maskable interrupt source to be requested.
32Ah	INSERV	In-Service	Indicates which channels are in-service—the channel's interrupt service routine is active.
32Ch	REQST	Interrupt Request	Indicates which channels have pending requests.
32Eh	INTSTS	Interrupt Status	Indicates the status of the General-Purpose DMA and Timer interrupt channels.
330h	DMAHLT	DMA Halt	Contains the DHLT bit which, when Set, halts all DMA activity. This bit is set by an NMI, and cleared by any IRET instruction. This bit can be read or written by software.

Table 7-2 Interrupt Controller Register Summary (Continued)

Offset	Register Mnemonic	Register Name	Description
332h	SHREQ	Interrupt Shared Request	Indicates if an INT signal that is enabled for shared interrupts is currently requesting an interrupt on the shared channel, Channel 14.
334h	SHMASK	Interrupt Shared Mask	Determines if an INT signal is masked (disabled) as a source for Channel 14.
336h	INTPOL	Interrupt Polarity	Sets the polarity, active High or active Low, of the INT signals.
338h	PIOPOL	PIO Polarity	Sets the polarity, active High or active Low, of the eight PIO signals that can be configured as interrupt sources.

7.5 OPERATION

7.5.1 Usage

Note: *Before using the interrupts, ensure multiplexed signals are configured to reflect the use of the interrupts and not other functionality (see Table 7-1 on page 7-4).*

7.5.1.1 Types of Interrupt Channels

The interrupt channels can be organized into five groups: Channel 0 (timers), Channel 1 (INT0 only), channels which support both an external and internal source (Channels 2, 3, and 8–13), channels which support two internal sources (Channels 4–7), and Channel 14 (shared interrupts). Channel 1 is a straightforward, single interrupt channel. For a list of interrupt types, see Table 7-3 on page 7-12. For a map of the interrupt channels, see Table 7-4 on page 7-16. The following sections discuss the other groups.

7.5.1.1.1 Timer Interrupt Requests Channel

Interrupt Channel 0 supports the three timers. Each timer has a bit in its control register that determines whether it is enabled to generate interrupt requests to the channel. The timers share a single programmable priority set in the CH0CON register. In addition, the three timers have relative priorities (see Table 7-3 on page 7-12). The Interrupt Controller uses the relative priority to arbitrate between the timers when more than one has an interrupt request pending. The channel logic determines which of the sources has the highest priority pending request and generates the interrupt vector based on that request. In previous parts, it could be confusing that all three interrupts required the same EOI (that of TMR0) even though they had different vectors. This happened because for all other sources, the vector number was identical with the EOI type. In the Am186CC/CH/CU microcontrollers, any of the three vector numbers can be used for the EOI; however, all three function identically by clearing the in-service bit for Channel 0. Table 7-3 on page 7-12 lists the EOI type for each interrupt.

Channel 9 (supports general-purpose DMA0 and general-purpose DMA1 as well as INT4) and Channel 10 (supports general-purpose DMA2 and general-purpose DMA3 as well as INT5) have similar behavior to the timers in regard to their support of the two DMA channels.

7.5.1.1.2 External and Internal Interrupt Request Channels

At any given point in time, interrupt channels 2, 3, 8, 9, 10, 11, 12, and 13 all support either an external or an internal source, but not both. The SRC bit in the CHxCON register determines the source for Channels 2, 3, and 8–11. Channels 12 and 13 support the external source until the PWD bit in the SYSCON register is set. For example, Channel 2 services the USB when the SRC bit is set, or INT1 when the SRC bit is cleared. The setting

or clearing of the SRC bit does not affect the vector generated, so INT1 and the USB share the same interrupt vector. Because only one can be generating interrupts at a time, this is unambiguous. All channels have a single programmable priority that is set in the CHxCON register.

7.5.1.1.3 *Two Internal Interrupts Request Channels*

Channels 4, 5, 6, and 7 support two internal interrupts. There is no SRC bit in the CHxCON registers for these channels because both sources on the channel can be active at the same time. For example, channel 4 supports both HDLC_A and SMDA0. These sources are programmed to either generate or mask their interrupt requests to the channel through bits in the control registers of the individual peripherals. The channel logic distinguishes between the different interrupt request sources and generates the vector based on the source. The channel has a single programmable priority that is set in the CHxCON register. In addition, the two sources for the channel have relative priorities (see Table 7-3 on page 7-12). The Interrupt Controller uses the relative priorities to arbitrate between the two sources when both have interrupt requests pending.

7.5.1.1.4 *Shared Interrupt Request Channel*

Channel 14 is the shared interrupt request channel. All sources on the shared channel have the same interrupt vector and the same priority. Software must examine the Shared Request (SHREQ) register to determine which source generated the interrupt. Note that software must configure a PIO pin as a PIO input or output before using it as an interrupt source.

7.5.1.2 Using Maskable Interrupts

1. Before configuring the external interrupts INT8–INT0 and the PIO interrupts, clear the IF flag in the FLAGS register (with the CLI instruction). However, most of the microcontroller's internal interrupts can be safely configured while maskable interrupts are enabled (i.e., the IF flag is set). The IF flag is cleared, disabling maskable interrupts, when the processor comes out of reset.
2. For PIO interrupts, program the associated PIO pin as a PIO input through the PIOMODEx and PIODIRx registers.
3. For external interrupts INT8–INT0, program the polarity, active High vs. active Low, through the INTPOL register.
4. Program the source and priority for the associated interrupt channel through the SRC and PR bits in the CHxCON register.

Note: *Do not perform Step 3 and Step 4 in a single write for edge-sensitive external interrupts. In this case, the polarity transition may be latched and generate a spurious interrupt request. Level-sensitive interrupts are not latched so any spurious request disappears before external interrupts are enabled.*

5. Specify the minimum priority required for an interrupt request to be recognized by setting the PRI bits in the PRIMSK register.
6. Specify the priority for the interrupts generated on a channel by setting the PRI bits in each of the CHxCON registers. The MSK (mask or enable) bit can be set concurrently.
7. Enable the desired interrupts by programming the CH bits in the IMASK register (if the MSK bits were not configured in step 6). Because these bits are physically identical to the MSK bits in each of the CHxCON registers, individual channels can be configured with the associated CHxCON register.

Note: *Do not write to the IMASK register while interrupts are enabled (the IF bit in the FLAGS register is set). In this case, spurious interrupt requests may be generated, including requests from devices whose interrupts were disabled both before and after the write to*

the IMASK register. It is safe to write the MSK bits in the CHxCON registers while interrupts are enabled.

8. Program the SHMASK register to enable the INT and PIO interrupts that share Channel 14. The SHREQ interrupt request is generated if any shared interrupt is asserted that is not masked off in the SHMASK register.
9. If interrupts are not enabled, enable interrupts by setting the IF flag in the FLAGS register using the STI instruction.

7.5.1.3 Using Nonmaskable Interrupts

To generate an NMI, use the NMI signal or watchdog timer. To generate a trace interrupt, set the TF bit in the FLAGS register. To generate a software interrupt, execute an Am186 instruction that generates an interrupt. This can be the INT or INTO instruction, or a software exception caused by an instruction. For more information, see “Nonmaskable Interrupts” on page 7-18.

7.5.2 Definitions of Interrupt Terms

The following definitions cover some of the terminology used in describing interrupts.

- **Interrupt Channel:** The group of logic that is comprised of a control register, an in-service bit, a request bit, and a mask bit.
- **Interrupt Source:** Any source such as an on-chip peripheral (internal) or physical pin (external) that can request an interrupt.
- **Interrupt Type***:* An eight-bit number assigned to each discrete interrupt, as listed in Table 7-3 on page 7-12. Each interrupt type does not need a unique interrupt channel; one interrupt channel can support more than one interrupt type. However, if one channel supports two interrupt types, then those two types have the same level of programmable priority.
- **Programmable Priority:** Each channel has eight levels of programmable priority, which are set in the Channel Control (CHxCON) register. Programmable priority determines which interrupt to service when two interrupts are requested at the same time. An interrupt service routine is interrupted by another interrupt request of equal or higher programmable priority, as long as the IF flag in the FLAGS register is set. For more information on setting the FLAGS register, see Chapter 2, “Configuration Basics.” If the programmable priority levels are equal, the overall priority number is used to resolve requests generated at the same time. The overall priority is not used to determine if a pending interrupt can interrupt an already executing interrupt service routine (ISR).
- **Overall Priority:** Each interrupt source has an overall priority number which is only used to arbitrate between two interrupt sources that have priority requests pending with the same programmable priority level. Overall priority is not used if the programmable priority is sufficient to resolve the pending highest-priority request.
- **Interrupt Vector Address:** This equals the interrupt type times four and is the location in memory that stores the address of the interrupt service routine for each interrupt type.
- **Interrupt Vector Table:** A memory area of 1 Kbyte beginning at address 00000h that contains up to 256 four-byte interrupt vector addresses organized by segment/offset.
- **Maskable Interrupts:** Maskable interrupts can be affected by programming and are enabled and disabled by setting the IF flag in the FLAGS register.
- **Nonmaskable Interrupts:** Nonmaskable interrupts cannot be affected by programming, nor are they affected by the IF flag.

- **Software Interrupt:** An interrupt initiated by the INT or INTO software instruction, or by a software exception. A software interrupt does not affect the IF flag.
- **Software Exception:** A software interrupt that occurs when an instruction causes a particular condition in the processor. A software exception does not affect the IF flag.
- **Trace Interrupt:** The trace interrupt is the highest priority interrupt. It is a software interrupt in that it is initiated by software, but unlike other software interrupts, it does clears the IF flag.
- **Hardware Interrupt:** Any one of the maskable interrupts, the NMI, and the watchdog timer interrupt. When a hardware interrupt is generated, the IF flag is cleared unless in polled mode.

7.5.3 Interrupt Sequence

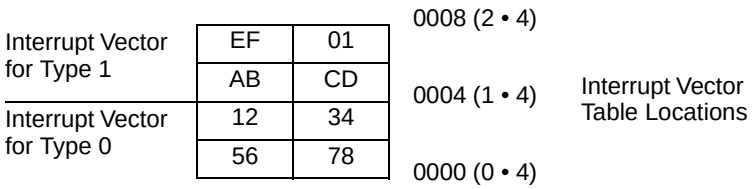
The following sections describe how the microcontroller services interrupts.

7.5.3.1 Requesting the Interrupt

When an interrupt is requested, the internal interrupt controller verifies that the interrupt is enabled and that there are no higher priority interrupt requests being serviced or pending. If the interrupt request is granted, the interrupt controller uses the interrupt type to access a vector from the interrupt vector table.

Each interrupt source has a corresponding interrupt type. Each interrupt type has a four-byte vector available in the interrupt vector table. The interrupt vector table is located in the 1024 bytes from 00000h to 003FFh. Each four-byte vector consists of a 16-bit offset (IP) value and a 16-bit segment (CS) value. The 8-bit interrupt type is shifted left two bit positions (multiplied by four) to generate the index into the interrupt vector table, as shown in Figure 7-2.

Figure 7-2 Interrupt Vector Translation



Interrupt Vector Type 1 = EF01:ABCD
Interrupt Vector Type 0 = 1234:5678

When an interrupt is taken, the type is multiplied by four and the processor fetches the pointer to the interrupt service routine from that interrupt vector address. Table 7-3 on page 7-12 shows a list of the types assigned to each interrupt source, as well as the interrupt vector address and the overall priority. The first entries in the table are the nonmaskable and software interrupt sources. The overall priority numbers are used only to resolve two interrupts that have identical programmable priority requests pending. In these cases, the type with the lowest overall priority number gets the highest priority. For overall priority numbers with letters, the lower letter is considered of higher priority (e.g., 2A is a higher priority than 2B).

7.5.3.2 Servicing the Interrupt

Nonmaskable interrupts—the trace interrupt, the NMI/watchdog timer interrupt, and software interrupts (both user-defined (INT) and software exceptions)—are serviced regardless of the setting of the IF flag in the FLAGS register. For more information about nonmaskable interrupts, see “Nonmaskable Interrupts” on page 7-18.

For maskable hardware interrupt requests to be serviced, the IF flag must be set by the STI instruction, and the mask bit associated with each interrupt must be reset. For more information about maskable interrupts, see “Maskable Interrupts” on page 7-13.

To service an interrupt request, the processor goes through the following steps:

1. When the processor senses a valid hardware interrupt, it pushes the next instruction address (CS:IP) and the FLAGS register onto the stack.
2. After the processor pushes the FLAGS register onto the stack, it clears the interrupt enable flag (IF) to disable maskable interrupts during the interrupt service routine (ISR).
3. The processor then loads the segment:offset values from the interrupt vector table into the code segment (CS) and the instruction pointer (IP), and begins executing the ISR.

7.5.3.3 Acknowledging the Interrupt

When the microcontroller services an interrupt, it sets the corresponding CHx bit in the INSERV register.

The microcontroller generates no external acknowledge cycles; the only external indication that an interrupt is being serviced is that the processor reads the interrupt vector table.

7.5.3.4 End-of-Interrupt (EOI)

Software must write to the End-of-Interrupt (EOI) register to reset the CHx bit in the INSERV register when an interrupt service routine for a maskable interrupt completes. There are two types of writes to the EOI register—specific EOI and non-specific EOI.

In a specific EOI, software must specify the interrupt type in the EOI register S bit field to indicate which CHx bit is to be reset. Specific EOI is applicable when interrupt nesting is possible or when the highest priority CHx bit that was set does not belong to the service routine in progress.

In a non-specific EOI, software does not specify which CHx bit is to be reset. Instead, the interrupt controller clears the CHx bits for all interrupt channels whose priorities match that of the highest priority interrupt in service.

7.5.3.5 Returning from the Interrupt

The interrupt return (IRET) instruction pops the FLAGS register and the return address off the stack. Program execution resumes at the point where the interrupt occurred.

The Interrupt Enable (IF) flag is restored by the IRET instruction along with the rest of the processor status flags. If the IF flag was set before the interrupt was serviced, interrupts are re-enabled when IRET is executed. If there are valid interrupts pending when the IRET is executed, the instruction at the return address is not executed. Instead, the processor services the new interrupt immediately.

If an ISR intends to permanently modify the value of any of the saved flags, it must modify the copy of the FLAGS register that was pushed onto the stack.

7.5.4 Interrupt Priority

Table 7-3 on page 7-12 shows the predefined types and overall priority structure for the Am186CC/CH/CU microcontrollers. The Overall Priority column shows the priority for the interrupts at power-on reset and at watchdog timer reset. Interrupt sources that constitute one request source share the same overall priority level with respect to other interrupt sources but are prioritized among themselves. This priority is indicated by letters following the priority number, with A having the highest priority, then B, etc.

Nonmaskable interrupts (types 0h–7h) are always higher priority than maskable interrupts.

Maskable interrupts have a programmable priority, set in the Channel Control (CHxCON) registers, which overrides the overall priority.

7.5.4.1 Nonmaskable Interrupt and Software Interrupt Priority

The nonmaskable interrupts and software interrupts from 00h to 07h always take priority over the maskable hardware interrupts. Within the nonmaskable and software interrupts, the trace interrupt has the highest priority, followed by the NMI/watchdog-timer interrupt, followed by the remaining software exceptions.

After the trace interrupt and the NMI/watchdog-timer interrupt, the remaining software exceptions are mutually exclusive and can only occur one at a time, so there is no further priority breakdown.

7.5.4.2 Maskable Hardware Interrupt Priority

Beginning with interrupt type 08h (the timer 0 interrupt), the maskable hardware interrupts have both an overall priority and a programmable priority (see Table 7-3). The programmable priority is the primary priority for maskable hardware interrupts and is set with the PR bit in the CGxCON registers. The overall priority is the secondary priority for maskable hardware interrupts.

Each of the maskable hardware interrupts has a programmable priority from zero to seven, with zero being the highest priority. Because all maskable interrupts are set to a programmable priority of seven on reset, the overall priority of the interrupts determines the priority in which each interrupt is granted by the interrupt controller until programmable priorities are changed by reconfiguring the CHxCON registers.

For example, if the INT6–INT0 interrupts are all changed to programmable priority six and no other programmable priorities are changed from the reset value of seven, then the INT6–INT0 interrupts take precedence over all other maskable interrupts. (Within INT6–INT0, the hierarchy is as follows: INT0>INT1>INT2>INT3>INT4>INT5>INT6.)

Table 7-3 Interrupt Types

Interrupt Source	Interrupt/ EOI Type	Vector Table Address	Related Instruction or Channel[1]	Overall Priority
Nonmaskable Interrupts				
Divide Error Exception	00h	00h	DIV, IDIV	1C[2]
Trace Interrupt	01h	04h	All	1A
NMI / Watchdog	02h	08h	N/A	1B
Breakpoint Interrupt	03h	0ch	INT3	1C[2]
INTO Detected Overflow Exception	04h	10h	INTO	1C[2]
Array Bounds Exception	05h	14h	BOUND	1C[2]
Unused Opcode Exception	06h	18h	Undefined Opcodes	1C[2]
ESC Opcode Exception	07h	1ch	ESC Opcodes	1C[2]
Maskable Interrupts				
Timer 0	08h	20h	Channel 0	2A
Timer 1	09h	24h	Channel 0	2B
Timer 2	0Ah	28h	Channel 0	2C
INT0	0Bh	2Ch	Channel 1	3
INT1[3] / USB CC CU	0Ch	30h	Channel 2	4
INT2[3] / High-Speed UART	0Dh	34h	Channel 3	5
HDLC A CC CH	0Eh	38h	Channel 4	6A
SDMA0 CC CH	0Fh	3Ch	Channel 4	6B
HDLC B CC CH	10h	40h	Channel 5	7A
SDMA1 CC CH	11h	44h	Channel 5	7B
HDLC C CC	12h	48h	Channel 6	8A
SDMA2 CC CU	13h	4Ch	Channel 6	8B
HDLC D CC	14h	50h	Channel 7	9A
SDMA3 CC CU	15h	54h	Channel 7	9B
INT3[3] / GCI CC	16h	58h	Channel 8	10A
INT4[3] / GP DMA0	17h	5Ch	Channel 9	10B
GP DMA1	18h	60h	Channel 9	10A
INT5[3] / GP DMA2	19h	64h	Channel 10	11A
GP DMA3	1Ah	68h	Channel 10	11B
INT6[3] / UART	1Bh	6Ch	Channel 11	12
INT7[3] / 2nd PWD[4]	1Ch	70h	Channel 12	13
INT8 / PWD[4]	1Dh	74h	Channel 13	14
PIO5, PIO15, PIO27, PIO29, PIO30, PIO33–PIO35 / INT 7–1 (Channel 14)[5]	1Eh	78h	Channel 14	15

Notes:

1. See the Am186 and Am188 Family Instruction Set Manual, order #21267, for more information about the instructions. See Table 7-5 on page 7-17 for more information about the channels.

2. These software exceptions can only occur one at a time, so there is no further priority breakdown.

3. The type and overall priority for the INT1–INT7 pins in this table assume that these pins are being serviced by a dedicated channel; that is, they are not being serviced by channel 14. When the INT1–INT7 pins are being serviced by Channel 14, they share type 1Eh, overall priority 15, as indicated by the last row in Table 7-3.

4. PWD is generated on the Low-to-High transition of the PWD input; the second PWD is generated on the High-to-Low transition.

5. See the SHREQ register description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for information on the shared Channel 14.

7.5.5 Maskable Interrupts

7.5.5.1 Maskable Interrupt Cycle

When the interrupt controller receives a request, it does the following:

1. Sets the appropriate channel bit in the Interrupt Request (REQST) register to indicate a pending interrupt. If the request is from the on-board timers or general-purpose DMA, it also sets the appropriate bit in the Interrupt Status (INTSTS) register to indicate a pending interrupt. If the request is for a Channel 14 interrupt, it also sets a bit in the Shared Request (SHREQ) register to indicate a pending interrupt.
2. Verifies the request by checking that the interrupt is enabled. An interrupt is enabled when the associated MSK bit in the IMASK register is set. An interrupt request coming in through the shared request channel, Channel 14, must also have the associated MSK bit in the SHMASK register set. If the associated MSK bit is not set, the interrupt is not recognized.
3. Verifies that the requesting interrupt's priority, set in the PRI field of the CHxCON register, is equal to or greater than the priority set in the PRIMASK register. If the interrupt source has not been programmed to equal or greater priority than the PRIMASK, the interrupt is not recognized.
4. Compares the programmable priority of the requesting interrupt with that of any interrupts currently in service. If the interrupt source is not of equal or greater priority than the highest priority interrupt in service, the interrupt is not recognized.
5. If the interrupt is recognized, the controller generates an interrupt request to the execution unit.
6. If the IF flag in the FLAGS register is set, the execution unit recognizes the request. Otherwise, the request remains pending until interrupts are enabled or the interrupting condition is cleared.
7. Passes the interrupt type (also called interrupt number) to the bus interface so the processor can fetch the associated vector from the interrupt vector table. The interrupt type provides an index into the interrupt vector table. The actual interrupt vector, the address of the interrupt service routine, is found in the table at the address indicated by the interrupt type times 4.
8. Sets the associated CH bit for the interrupt channel in the INSERV register to indicate that an interrupt on that channel is currently being serviced by software.
9. The controller clears the CH bit for the channel when an EOI instruction is executed with either of two conditions: a specific EOI that specifies this channel or a non-specific EOI when this channel is the highest priority interrupt whose CH bit is set.

7.5.5.2 Interrupts In Polled Mode

Software can handle interrupt requests in polled mode. In polled mode, configure the interrupt sources exactly as in normal interrupt mode, but do not set the IF bit in the FLAGS register. This disables automatic hardware servicing of interrupt requests. In this case, software must periodically read the POLL or POLLST register to determine if a valid interrupt request is pending. Reading the POLL or POLLST register provides identical information; however, a read of the POLL register generates an interrupt recognition cycle whereas a read of the POLLST register does not.

Except for the manner in which the interrupt recognition is generated, and the fact that software must jump to the interrupt service routine, the behavior under normal interrupt and polled mode interrupt is identical. This means, for example, that the CHx bit in the INSVR register must be cleared by an EOI instruction as in normal interrupt mode.

7.5.5.3 Considerations for NMI, Software Interrupts, and Traps

The nonmaskable interrupt (NMI) is not processed through the interrupt controller. Its detection is not affected by the settings of the IF flag, the bits in the INSRV register, or by the priority mask. When an NMI interrupt is taken, the IF flag is cleared and the DHLT bit is set. This disables maskable interrupts and inhibits DMA transfers.

Although the NMI is the highest priority hardware interrupt, it does not participate in the priority resolution scheme of the maskable interrupts. Setting the IF flag using the STI instruction during an NMI service routine is discouraged because any maskable interrupt may interrupt an executing NMI routine, assuming it meets the criteria outlined above. DMA activity may be re-enabled by clearing the DHLT bit but this could increase the number of cycles required to complete the NMI routine and, consequently, the number of cycles during which interrupts are disabled.

A currently executing NMI service routine may be preempted by a second NMI request. NMI is active when the part is reset and cannot be disabled.

The NMI can be generated externally through the NMI pin or internally through the watchdog timer. The microcontroller logically ORs the two sources internally to provide a single signal to the execution unit. Because the NMI signal is edge-sensitive, it is possible to block the recognition of a watchdog timer NMI by holding the external NMI signal asserted. Systems that do not use external NMI should hold this pin low to yield control to the watchdog timer NMI.

A software interrupt or trap is not processed through the interrupt controller and is not affected by the setting of the IF flag, the bits in the INSRV register, by the priority mask, or by a currently executing NMI service routine.

7.5.5.4 Maskable Interrupt Overview

Interrupt types 08h through 1Eh are maskable (see Table 7-3 on page 7-12). The maskable interrupts are enabled and disabled by the IF flag in the FLAGS register, but the INT command can execute any interrupt regardless of the setting of IF.

Maskable interrupts are supported through the interrupt controller, which contains the configuration and status of all the interrupt sources, as well as priority resolution logic to select which interrupts to process in which order. The interrupt controller supports maskable interrupts with 15 interrupt channels. Because of the large number of interrupt sources available, some sources share interrupt channels. Table 7-4 on page 7-16 and Table 7-5 on page 7-17 show which channels service each source.

The interrupt controller uses the peripheral registers listed in Table 7-2 on page 7-5 to support generating a maskable interrupt. In addition, the FLAGS processor register contains a flag to enable the interrupts and one to set the trace interrupt. For more information about the interrupt registers, see “Registers Used” on page 7-18.

Of the maskable interrupts, 17 signals are provided for external interrupt sources: 9 interrupt signals and 8 PIOs (the NMI signal is *nonmaskable* and is generally used for unusual events like power failure). The interrupt types for these inputs are generated internally. Every interrupt channel has an in-service bit. If a lower-priority device requests an interrupt while the in-service bit (IS) is set for a high-priority interrupt, the interrupt controller does not generate an interrupt. In addition, if another interrupt request occurs from the same interrupt source while the in-service bit is set, the interrupt controller does not generate an interrupt. This allows interrupt service routines operating with interrupts enabled to be suspended only by interrupts of equal or higher priority than the in-service interrupt.

When an interrupt service routine completes, software must reset the proper in-service bit by writing the EOI type to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. Software should execute a write to the EOI register at the end of the interrupt service routine just before the return from interrupt instruction.

7.5.5.5 Maskable Interrupt Block Diagram

Figure 7-3 shows a partial block diagram of how the sources and channels are used (see Figure 7-1 on page 7-3 for another block diagram). The three timers share Channel 0 and produce three separate types. The INT0 signal is dedicated to Channel 1. The GP DMA0 and GP DMA1 are MUXed with the INT4 signal onto Channel 9, and they produce up to two separate types (only one type is generated if Channel 9 services the INT4 signal). The INT4 signal is also connected to the Channel 14 shared interrupts through a mask register and shares the same type as the rest of the Channel 14 shared interrupts.

Figure 7-3 Partial Block Diagram of Interrupt Controller Scheme

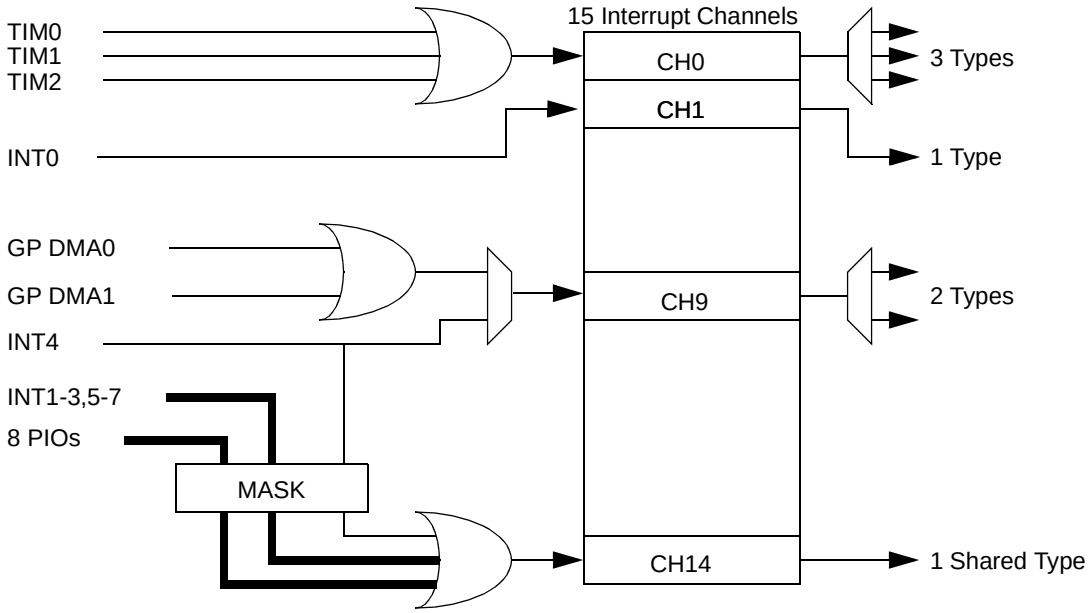


Table 7-4 Interrupt Channel Map

Interrupt Source	Interrupt Channel[1]														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Timer 0	X														
Timer 1	X														
Timer 2	X														
High-Speed UART				X											
UART												X			
HDLC_A CC CH					X										
HDLC_B CC CH						X									
HDLC_C CC							X								
HDLC_D CC								X							
GCI CC									X						
SDMA0 CC CH					X										
SDMA1 CC CH						X									
SDMA2 CC CU							X								
SDMA3 CC CU								X							
GP DMA0										X					
GP DMA1										X					
GP DMA2											X				
GP DMA3											X				
USB CC CU			X												
INT0		X													
INT1			X												X
INT2				X											X
INT3									X						X
INT4										X					X
INT5											X				X
INT6												X			X
INT7													X		X
INT8														X	
PWD[2]														X	
2nd PWD[2] Interrupt													X		
PIO5															X
PIO15															X
PIO27															X
PIO29															X
PIO30															X
PIO33															X
PIO34															X
PIO35															X

Notes:

1. Channels 0 to 3 and 8 to 13 can have only one interrupt source active at a time (e.g., Channel 2 can only service the INT1 signal or the USB at any one time). Channels 4 to 7 (shaded) can service up to two sources at once (e.g., Channel 4 can service the HDLC_A as well as SDMA0 interrupt requests). The peripherals that generate the interrupts on channels 4 to 7 have the option of enabling or disabling their requests. Channel 14 (shaded) is provided to allow a second channel to service interrupt requests from external signals. This is useful for systems that require a large number of peripheral interrupts (e.g., if a system is using USB interrupts via Channel 2, the INT1 signal is able to request an interrupt through Channel 14). Channel 14 can simultaneously service any source indicated in its column. For Channel 14, a register individually masks on or off the signals serviced by this channel so that individual control of interrupt sources is possible.

2. For a complete description of Pulse Width Demodulation (PWD) mode, see Chapter 10, "Programmable Timers."

Table 7-5 Interrupt Channel Sources

Interrupt Channel	Default Source	Optional Source
0	Timer 0, Timer 1, and Timer 2	—
1	INT0	—
2	INT1	USB CC CU
3	INT2	High-Speed UART
4	HDLC Channel A CC CH and SmartDMA Channel Pair 0 CC CH	—
5	HDLC Channel B CC CH and SmartDMA Channel Pair 1 CC CH	—
6	HDLC Channel C CC and SmartDMA Channel Pair 2 CC CU	—
7	HDLC Channel D CC and SmartDMA Channel Pair 3 CC CU	—
8	INT3	GCI CC
9	INT4	General-Purpose DMA 0 and 1
10	INT5	General-Purpose DMA 2 and 3
11	INT6	UART
12	INT7	PWD[1]
13	INT8	PWD[1]
14	Shared Request[2]	—

Notes:

1. The PWD source is selected by setting the PWD bit in the SYSCON register.

2. The Shared Request source is controlled by the SHREQ and SHMASK registers. The following sources can be enabled to use the Shared Request channel: PIO5, PIO15, PIO27, PIO29, PIO30, PIO33, PIO34, and PIO35; and INT pins 7–1.

7.5.5.6 PIOs as Interrupts

Eight PIOs (PIO5, PIO15, PIO27, PIO29, PIO30, PIO33, PIO34, and PIO35) are programmable as external interrupt sources on shared Channel 14. These PIOs are level-triggered. PIO15, PIO29, PIO30, PIO33, PIO34, and PIO35 default to their alternate function at external or internal reset. To use these signals as interrupts, enable them as interrupts in the SHMASK register. Typically, software should configure these signals as inputs in the PIOMODEx and PIODIRx registers when using them as interrupt sources. If any of these signals is configured as both a PIO output and as an interrupt source, the PIO output signal generates interrupts. For more information, see Table 9-3 on page 9-6.

In addition, three PIOs (PIO6, PIO7, and PIO19) are multiplexed with external interrupt signals (INT8, INT7, and INT6, respectively) so they can act as interrupts when the signal's interrupt signal is enabled. These signals can be programmed with the LTM bit in the CHxCON register to be edge- or level-triggered.

7.5.5.7 Registers Used

Each interrupt channel has a control register, an in-service bit, a request bit, and a mask bit programmed with the 15 Channel Control registers, the In-Service register, the Interrupt Request register, and the Interrupt Mask register. Because Channel 14 shares interrupts, it requires two additional registers to implement the shared interrupts: the Shared Mask register and the Shared Request register. The control register for Channel 14 contains a priority field and mask bit only. For detailed information about these and the other peripheral registers used to control interrupts, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

In addition, bits 8 and 9 in the FLAGS register relate to interrupt operation.

Bit 8 of the FLAGS register is the Trace Flag (TF). When TF is set to 1, a trace interrupt occurs after each instruction executes. TF is cleared by the trace interrupt after the processor status flags are pushed onto the stack. The trace service routine can continue tracing by popping the flags back with an IRET instruction.

Bit 9 of the FLAGS register is the Interrupt Enable Flag (IF). IF acts as an enable for all maskable interrupts.

If IF is set to 1, maskable interrupts are enabled and can cause processor interrupts. Individual maskable interrupts can still be disabled through their corresponding mask bit in the IMASK or CHxCON registers. Some peripheral devices have their own interrupt mask bits, as well.

Clearing IF to 0 disables all maskable interrupts regardless of the setting of the mask bits in the IMASK or CHxCON registers or any other peripheral control mask bits. The IF flag does not affect the NMI, trace, or software exception interrupts (interrupt types 00h to 07h), and it does not affect the execution of any interrupt through the INT instruction.

7.5.6 Nonmaskable Interrupts

Interrupt types 00h through 07h (see Table 7-3 on page 7-12) and all software interrupts cannot be masked by programming, and are not affected by the setting of the IF flag. Software interrupts are initiated with the INT or INTO instruction. A software exception interrupt occurs when an instruction causes an interrupt due to some condition in the processor. Interrupt types 00h, 03h, 04h, 05h, 06h, and 07h are software exception interrupts. For more information about INT or other instructions, see the *Am186 and Am188 Family Instruction Set Manual*, order #21267.

7.5.6.1 Software Interrupts

Up to 256 possible interrupts can be initiated by the INT or INTO instructions. INT 21h causes an interrupt to the vector located at 00084h in the interrupt vector table. INT FFh causes an interrupt to the vector located at 003FCh in the interrupt vector table.

7.5.6.2 Divide Error Exception (Interrupt Type 00h)

When a DIV or IDIV instruction quotient cannot be expressed in the number of destination bits, it generates a Divide Error exception.

7.5.6.3 Trace Interrupt (Interrupt Type 01h)

If the Trace Flag (TF) in the FLAGS register is set, the trace interrupt is generated after most instructions. The trace interrupt is the highest priority interrupt. This interrupt allows programs to execute in single-step mode. The interrupt is not generated after prefix instructions like REP, instructions that modify segment registers like POP DS, or the WAIT instruction.

Taking the trace interrupt clears the TF bit after the flags are pushed onto the stack. The IRET instruction at the end of the single step interrupt service routine restores the processor status flags (including the TF bit) and transfers control to the next instruction to be traced. Taking the trace interrupt also clears the IF flag.

Trace mode is initiated by pushing the FLAGS register onto the stack, then setting the TF flag on the stack, and then popping the flags.

For more information about the FLAGS register, see Chapter 2, “Configuration Basics.”

7.5.6.4 Nonmaskable Interrupt (Interrupt Type 02h)

An NMI can be generated internally or externally. An internal NMI is generated with the watchdog timer. For more information about the watchdog timer, see Chapter 11, “Watchdog Timer.”

An external NMI is generated with the NMI signal. This signal indicates to the microcontroller that an interrupt request has occurred. The NMI signal is the highest priority hardware interrupt and, unlike the INT8–INT0 signals, cannot be masked. When NMI is asserted, the processor transfers program execution to the location specified by the nonmaskable interrupt vector in the interrupt vector table.

Additionally, when an NMI occurs, DMAs are suspended. If your application is using a DMA channel, the NMI interrupt handler may need to update the DMA configuration settings to account for the DMA being suspended by the NMI.

An NMI transition from Low to High is latched and synchronized internally, and it initiates the interrupt at the next instruction boundary. To guarantee that the interrupt is recognized, the NMI signal must be asserted for at least one CLKOUT period.

For information about the nonmaskable interrupt and interrupt priority processing, see “Considerations for NMI, Software Interrupts, and Traps” on page 7-14.

7.5.6.5 Breakpoint Interrupt (Interrupt Type 03h)

The 1-byte version of the INT instruction (INT3) causes a breakpoint interrupt.

7.5.6.6 INT0 Detected Overflow Exception (Interrupt Type 04h)

If the OF bit is set in the FLAGS register, an INT0 instruction generates the INT0 Detected Overflow exception. For more information about the FLAGS register, see Chapter 2, “Configuration Basics.”

7.5.6.7 Array Bounds Exception (Interrupt Type 05h)

If an array index is outside the array bounds, a BOUND instruction generates an Array Bounds exception. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked. For more information, see the *Am186 and Am188 Family Instruction Set Manual*, order #21267.

7.5.6.8 Unused Opcode Exception (Interrupt Type 06h)

If the processor attempts to execute an undefined opcode, it generates an Unused Opcode exception.

7.5.6.9 ESC Opcode Exception (Interrupt Type 07h)

If the processor attempts to execute an ESC opcode (D8h–DFh), it generates the ESC Opcode exception. The processor does not check the escape opcode trap bit. The return address of this exception points to the ESC instruction that caused the exception. If a segment override prefix preceded the ESC instruction, the return address points to the segment override prefix.

Note: *All numeric coprocessor opcodes cause a trap. The Am186CC/CH/CU microcontrollers do not support the numeric coprocessor interface.*

7.5.7 Software-Related Considerations

- The watchdog timer can generate an NMI. This interrupt can be taken at any time. Unlike the maskable interrupts, the controller is not inhibited from taking a second NMI request while the NMI interrupt service routine is executing. Therefore a watchdog timer-generated NMI can interrupt, or be interrupted by, an externally generated NMI. For more information about the watchdog timer NMI, see “Considerations for NMI, Software Interrupts, and Traps” on page 7-14.
- Writing a zero to the appropriate channel bit in the Interrupt Request (REQST) register clears the pending interrupt. This facility provides a simple way to clear a spurious edge-triggered interrupt that may have occurred when initially configuring a PIO pin as an interrupt source.

7.5.8 Comparison to Other Devices

The interrupt controller supports the Fully Nested Master mode and Polled mode operation available in all AMD Am186 devices. The interrupt controller does *not* support Slave mode, Cascade mode, or Special Fully Nested mode. Support for the NMI and software interrupts are similar to Master mode in AMD’s Am186ES microcontroller.

7.6 INITIALIZATION

On both an external and internal reset, the following occurs:

- All priority bits in the Channel Control (CHxCON) registers are set to 1. This places all sources at the lowest priority (level 7). The overall priority of the interrupts determines the priority in which each interrupt is granted by the interrupt controller until programmable priorities are changed by reconfiguring the CHxCON registers.
- All mask bits in the Channel Control (ChxCON) registers are set to 1, so all interrupts are masked.
- All Level-Triggered Mode (LTM) bits in the Channel Control (CHxCON) registers are cleared, resulting in edge-triggered mode.
- All source bits in the Channel Control (CHxCON) registers are cleared, defining the source as that channel’s external interrupt source.

- The End-of-Interrupt (EOI) register is cleared, so no in-service bits are cleared.
- The Poll (POLL) and Poll Status (POLLST) registers are cleared, so Polling mode is disabled.
- The Interrupt Mask (IMASK) register and Shared Mask (SHMASK) registers are set to FFFFh, so all interrupts are masked.
- The PRM bits in the Priority Mask (PRIMSK) register are set to 7d, allowing interrupts of all priorities.
- The In-Service (INSERV) register is cleared, indicating that no interrupts are active.
- The Interrupt Request (REQST), Interrupt Status (INTSTS), and Shared Request (SHREQ) registers are cleared, indicating there are no pending interrupts.
- The DMA Halt (DMAHLT) register is cleared, so DMA activity is unaffected.
- All bits in the Interrupt Polarity (INTPOL) and PIO Polarity (PIOPOL) registers are set to 1, so INT9–INT0 and PIO5, PIO15, PIO27, PIO29, PIO30, and PIO33–PIO35 are set to be active High.
- Multiplexed signals INT8–INT6 default to PIO functionality as shown in Table 7-1 on page 7-4.
- Multiplexed signals PIO15, PIO29, PIO30, and PIO33–PIO35 default to their alternate functionality, as described in Chapter 9, “Programmable I/O Signals.”
- The IF flag in the FLAGS register is cleared, disabling maskable interrupts.

CHAPTER

8 DMA CONTROLLER

8.1 OVERVIEW

Direct memory access (DMA) permits the transfer of data between memory and peripherals without CPU involvement. With DMA transfers, the DMA controller becomes the bus master. The arbitration for the bus is internal to the processor and is not visible externally. When the DMA no longer has transfers pending (no internal or external DRQs are asserted) or a higher priority event occurs, the DMA controller removes its request for the bus thus freeing the bus for other types of cycles. The type of DMA transfer dictates how long the DMA controller has control of the bus. However, because a DMA transfer is using the bus, the processor can be slowed down if it also needs the bus.

Each of the Am186CC/CH/CU microcontrollers contains a DMA controller that provides both SmartDMA channels and general-purpose DMA channels. The general-purpose DMA channels can be used for data transfer between memory and I/O spaces (i.e., memory-to-I/O or I/O-to-memory) or within the same space (i.e., memory-to-memory or I/O-to-I/O). In addition, the general-purpose DMA controller supports data transfer between some internal peripherals and memory or I/O.

The SmartDMA channels provide a method for transmission and reception of data across multiple memory buffers and a sophisticated buffer-chaining mechanism. These channels are always used in pairs: transmitter and receiver. The transmit channels can only transfer data from memory to a peripheral; the receive channels can only transfer data from a peripheral to memory.

CC The Am186CC microcontroller provides a total of 12 DMA channels: eight SmartDMA channels and four general-purpose DMA channels. Four of the SmartDMA channels (two pairs) are dedicated for use with two of the on-board HDLC channels. The remaining four SmartDMA channels (two pairs) can support either the third or fourth HDLC channel or Universal Serial Bus (USB) endpoints A, B, C, or D. On-chip peripherals that support general-purpose DMA are Timer 2, the two asynchronous serial ports (the UART and the High-Speed UART), and the USB peripheral controller. External peripherals support DMA transfers through the external DMA request signals. Each general-purpose channel accepts a DMA request from one of four sources: the DMA request signals (DRQ1–DRQ0), Timer 2, the UARTs, or the USB peripheral controller. (Note that Timer 2 acts only as a DMA request source; no data is transferred to or from Timer 2.)

CH The Am186CH HDLC microcontroller provides a total of eight DMA channels: four SmartDMA channels (two transmit-receive pairs, 0 and 1) and four general-purpose DMA channels. The SmartDMA channel pairs are dedicated to the two on-board HDLC channels. On-chip peripherals that support general-purpose DMA are Timer 2, and the two asynchronous serial ports (the UART and the High-Speed UART). External peripherals support DMA transfers through the external DMA request signals. Each general-purpose channel accepts a DMA request from one of three sources: the DMA request signals (DRQ1–DRQ0), Timer 2, or the UARTs. (Note that Timer 2 acts only as a DMA request source; no data is transferred to or from Timer 2.)

CU The Am186CU USB microcontroller also provides four SmartDMA channels (two transmit-receive pairs, 2 and 3) and four general-purpose DMA channels. The SmartDMA channel

pairs support the USB endpoints A, B, C, or D. On-chip peripherals that support general-purpose DMA are Timer 2, the two asynchronous serial ports (the UART and the High-Speed UART), and the USB peripheral controller. External peripherals support DMA transfers through the external DMA request signals. Each general-purpose channel accepts a DMA request from one of four sources: the DMA request signals (DRQ1–DRQ0), Timer 2, the UARTs, or the USB peripheral controller. (Note that Timer 2 acts only as a DMA request source; no data is transferred to or from Timer 2.)

Up to 64 Kbytes or 64 Kwords can be transferred to or from even or odd addresses on the Am186CC/CH/CU microcontrollers. Two bus cycles (a minimum of eight clocks) are necessary for each general-purpose DMA data transaction. For word transfers, both the source and destination addresses must be configured as 16-bit addresses.

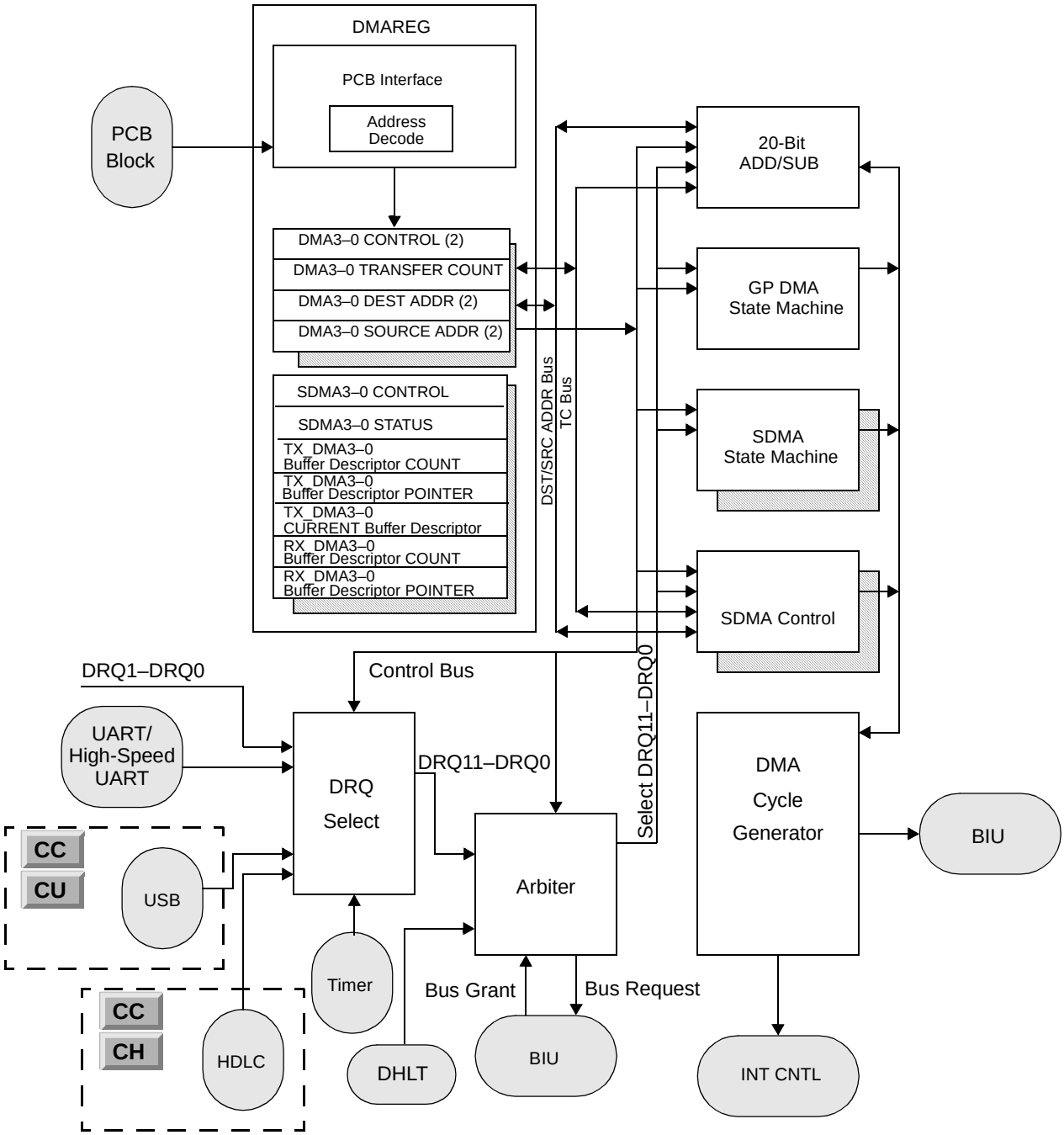
The SmartDMA channels only support byte transfers. Data is written or read from sequential byte addresses in the memory buffers. The SmartDMA channels also feature fly-by DMA transfers—what would typically take two cycles (a read and write) is moved in a single cycle on the external processor bus; read and write are performed concurrently in one cycle.

The general-purpose DMA channels and the SmartDMA channels can be programmed so that one channel/channel pair is always given priority over the other, or they can be programmed to alternate cycles when both have DMA requests pending.

8.2 BLOCK DIAGRAM

Figure 8-1 shows the block diagram for the general-purpose DMA and SmartDMA controllers.

Figure 8-1 DMA Block Diagram



8.3 SYSTEM DESIGN

Table 8-1 lists the DMA signals that are multiplexed with other microcontroller functions. Pinstraps are sampled only at external reset and do not affect the pin's other functions, so they are not shown in this table. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin.

Table 8-1 DMA Multiplexed Signals

Signal	Function	Multiplexed Signal(s)	Default Signal
DRQ0	DMA requests	PIO9	PIO9
DRQ1		—	DRQ1

8.4 REGISTERS

The DMA controller is programmed through the use of registers: seven registers for each general-purpose channel and nine for each pair of SmartDMA channels (see Table 8-2). In addition, software can use the DMA Halt (DMAHLT) register (an Interrupt Controller register) to halt DMA activity. Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

DMA channel control registers can be changed while the channel is operating. Any changes made during DMA operations affect the current DMA transfer.

All DMA registers except the GDxCON0 and GDxCON1 registers can be modified or altered during any DMA activity. Any changes made to these registers are reflected immediately in DMA operation.

Table 8-2 DMA Controller Register Summary

Offset	Register Mnemonic	Register Name	Description
General-Purpose DMA Channel Registers			
100h	GD0CON0	General-Purpose DMA0 Control 0	Set up general-purpose DMA Channel 0. Software must stop DMA operation before writing to these registers, or results will be unpredictable.
102h	GD0CON1	General-Purpose DMA0 Control 1	
104h	GD0SRCL	General-Purpose DMA0 Source Address Low	The 16 bits of this register, combined with four bits of the high register, produce a 20-bit source address for general-purpose DMA Channel 0.
106h	GD0SRCH	General-Purpose DMA0 Source Address High	Four bits of this register [19–16], combined with the 16 bits of the low register, produce a 20-bit source address for general-purpose DMA Channel 0.
108h	GD0DSTL	General-Purpose DMA0 Destination Address Low	The 16 bits of this register, combined with four bits of the high register, produce a 20-bit destination address for general-purpose DMA Channel 0.

Table 8-2 DMA Controller Register Summary (Continued)

Offset	Register Mnemonic	Register Name	Description
10Ah	GD0DSTH	General-Purpose DMA0 Destination Address High	Four bits of this register [19–16], combined with the 16 bits of the low register, produce a 20-bit destination address for general-purpose DMA Channel 0.
10Ch	GD0TC	General-Purpose DMA0 Transfer Count	Sets the transfer count (the number of DMA transfers to be performed) for general-purpose DMA Channel 0.
110h	GD1CON0	General-Purpose DMA1 Control 0	Behaves the same as General-Purpose DMA0 registers, but for DMA Channel 1.
112h	GD1CON1	General-Purpose DMA1 Control 1	
114h	GD1SRCL	General-Purpose DMA1 Source Address Low	
116h	GD1SRCH	General-Purpose DMA1 Source Address High	
118h	GD1DSTL	General-Purpose DMA1 Destination Address Low	
11Ah	GD1DSTH	General-Purpose DMA1 Destination Address High	
11Ch	GD1TC	General-Purpose DMA1 Transfer Count	
120h	GD2CON0	General-Purpose DMA2 Control 0	Behaves the same as General-Purpose DMA0 registers, but for DMA Channel 2.
122h	GD2CON1	General-Purpose DMA2 Control 1	
124h	GD2SRCL	General-Purpose DMA2 Source Address Low	
126h	GD2SRCH	General-Purpose DMA2 Source Address High	
128h	GD2DSTL	General-Purpose DMA2 Destination Address Low	
12Ah	GD2DSTH	General-Purpose DMA2 Destination Address High	
12Ch	GD2TC	General-Purpose DMA2 Transfer Count	
130h	GD3CON0	General-Purpose DMA3 Control 0	Behaves the same as General-Purpose DMA0 registers, but for DMA Channel 3.
132h	GD3CON1	General-Purpose DMA3 Control 1	
134h	GD3SRCL	General-Purpose DMA3 Source Address Low	
136h	GD3SRCH	General-Purpose DMA3 Source Address High	
138h	GD3DSTL	General-Purpose DMA3 Destination Address Low	
13Ah	GD3DSTH	General-Purpose DMA3 Destination Address High	
13Ch	GD3TC	General-Purpose DMA3 Transfer Count	

Table 8-2 DMA Controller Register Summary (Continued)

Offset	Register Mnemonic	Register Name	Description
SmartDMA Channel Pair 0 and 1 Registers CC CH			
140h	SD0CON	SmartDMA0 Control	Sets up SmartDMA Channel 0.
142h	SD0TRCAL	SmartDMA0 Transmit Ring Count / Address Low	Indicates the number of available buffer descriptors per transmit ring in SmartDMA Channel 0. Also contains the low address bits [15–4] of the transmit buffer descriptor ring address for SmartDMA Channel 0.
144h	SD0TRAH	SmartDMA0 Transmit Ring Address High	Points to the transmit buffer descriptor ring high address bits [19–16] for SmartDMA Channel 0.
146h	SD0RRCAL	SmartDMA0 Receive Ring Count / Address Low	Indicates the number of available buffer descriptors per receive ring in SmartDMA Channel 0. Also contains the low address bits [15–4] of the receive buffer descriptor ring address for SmartDMA Channel 0.
148h	SD0RRAH	SmartDMA0 Receive Ring Address High	Points to the receive buffer descriptor ring high address bits [19–16] for SmartDMA Channel 0.
14Ah	SD0STAT	SmartDMA0 Status	Indicates the status of SmartDMA Channel 0.
14Ch	SD0CBD	SmartDMA0 Current Buffer Descriptor	Indicates the buffer descriptor currently accessed by SmartDMA Channel 0.
14Eh	SD0CTAD	SmartDMA0 Current Transmit Address	Indicates the current address of the data being transmitted by SmartDMA Channel 0.
150h	SD0CRAD	SmartDMA0 Current Receive Address	Indicates the current address of the data being received by a SmartDMA channel.
158h	SD1CON	SmartDMA1 Control	Behaves the same as SmartDMA Channel 0 registers, but for SmartDMA Channel 1.
15Ah	SD1TRCAL	SmartDMA1 Transmit Ring Count / Address Low	
15Ch	SD1TRAH	SmartDMA1 Transmit Ring Address High	
15Eh	SD1RRCAL	SmartDMA1 Receive Ring Count / Address Low	
160h	SD1RRAH	SmartDMA1 Receive Ring Address High	
162h	SD1STAT	SmartDMA1 Status	
164h	SD1CBD	SmartDMA1 Current Buffer Descriptor	
166h	SD1CTAD	SmartDMA1 Current Transmit Address	
168h	SD1CRAD	SmartDMA1 Current Receive Address	

Table 8-2 DMA Controller Register Summary (Continued)

Offset	Register Mnemonic	Register Name	Description
SmartDMA Channel Pair 2 and 3 Registers CC CU			
170h	SD2CON	SmartDMA2 Control	Behaves the same as SmartDMA Channel 0 registers, but for SmartDMA Channel 2.
172h	SD2TRCAL	SmartDMA2 Transmit Ring Count / Address Low	
174h	SD2TRAH	SmartDMA2 Transmit Ring Address High	
176h	SD2RRCAL	SmartDMA2 Receive Ring Count / Address Low	
178h	SD2RRAH	SmartDMA2 Receive Ring Address High	
17Ah	SD2STAT	SmartDMA2 Status	
17Ch	SD2CBD	SmartDMA2 Current Buffer Descriptor	
17Eh	SD2CTAD	SmartDMA2 Current Transmit Address	
180h	SD2CRAD	SmartDMA2 Current Receive Address	
188h	SD3CON	SmartDMA3 Control	Behaves the same as SmartDMA Channel 0 registers, but for SmartDMA Channel 3.
18Ah	SD3TRCAL	SmartDMA3 Transmit Ring Count / Address Low	
18Ch	SD3TRAH	SmartDMA3 Transmit Ring Address High	
18Eh	SD3RRCAL	SmartDMA3 Receive Ring Count / Address Low	
190h	SD3RRAH	SmartDMA3 Receive Ring Address High	
192h	SD3STAT	SmartDMA3 Status	
194h	SD3CBD	SmartDMA3 Current Buffer Descriptor	
196h	SD3CTAD	SmartDMA3 Current Transmit Address	
198h	SD3CRAD	SmartDMA3 Current Receive Address	

8.5 OPERATION

The Am186CC/CH/CU microcontrollers contain two distinct types of DMA channels: general-purpose DMA channels and SmartDMA channels. The SmartDMA channels are further broken down into transmit and receive channels, which are used in pairs. The microcontroller's DMA channels can be used as shown in Table 8-3, Table 8-4, and Table 8-5. The discussion of general-purpose DMA channels begins in "General-Purpose DMA Channels" on page 8-11; the discussion of SmartDMA channels begins in "SmartDMA Channels" on page 8-26. In some cases, a hybrid between DMA processing and interrupt processing is appropriate. This is described in "DMA and Interrupts" on page 8-10.

Table 8-3 Am186CC Communications Controller DMA Channel Use

CC

DMA Channel	Associated Peripheral
General-Purpose DMA Channels 0–3	Memory-to-memory transfers (note I/O space can be used in addition to memory space), Timer 2, external peripherals (via the DRQ signals), internal UART or High-Speed UART, or any USB data endpoint (A–D) configured in either direction
SmartDMA Pair 0 Receive Channel	HDLC A receiver
SmartDMA Pair 0 Transmit Channel	HDLC A transmitter
SmartDMA Pair 1 Receive Channel	HDLC B receiver
SmartDMA Pair 1 Transmit Channel	HDLC B transmitter
SmartDMA Pair 2 Transmit Channel	HDLC C transmitter or USB data endpoint B if configured as a USB IN endpoint[1]
SmartDMA Pair 2 Receive Channel	HDLC C receiver or USB data endpoint A if configured as a USB OUT endpoint[1]
SmartDMA Pair 3 Transmit Channel	HDLC D transmitter or USB data endpoint D if configured as a USB IN endpoint[1]
SmartDMA Pair 3 Receive Channel	HDLC D receiver or USB data endpoint C if configured as a USB OUT endpoint[1]

Notes:
1. For SmartDMA channels 2 and 3, the transmit and receive cannot be assigned to different peripherals. For example, if SmartDMA Channel 2 receive is assigned to USB data endpoint A, then SmartDMA Channel 2 transmit can be used for USB data endpoint B, but cannot be used with the HDLC controller.

Table 8-4 Am186CH HDLC Microcontroller DMA Channel Use

CH

DMA Channel	Associated Peripheral
General-Purpose DMA Channels 0–3	Memory-to-memory transfers (note I/O space can be used in addition to memory space), Timer 2, external peripherals (via the DRQ signals), internal UART or High-Speed UART
SmartDMA Pair 0 Receive Channel	HDLC A receiver
SmartDMA Pair 0 Transmit Channel	HDLC A transmitter
SmartDMA Pair 1 Receive Channel	HDLC B receiver
SmartDMA Pair 1 Transmit Channel	HDLC B transmitter

Table 8-5 Am186CU USB Microcontroller DMA Channel Use

CU

DMA Channel	Associated Peripheral
General-Purpose DMA Channels 0–3	Memory-to-memory transfers (note I/O space can be used in addition to memory space), Timer 2, external peripherals (via the DRQ signals), internal UART or High-Speed UART, or any USB data endpoint (A–D) configured in either direction
SmartDMA Pair 2 Transmit Channel	USB data endpoint B if configured as a USB IN endpoint
SmartDMA Pair 2 Receive Channel	USB data endpoint A if configured as a USB OUT endpoint
SmartDMA Pair 3 Transmit Channel	USB data endpoint D if configured as a USB IN endpoint
SmartDMA Pair 3 Receive Channel	USB data endpoint C if configured as a USB OUT endpoint

8.5.1 When to Use DMA

Using DMA is appropriate under certain circumstances. In general, using DMA means trading programming simplicity for efficient data movement between peripherals. For most devices, using DMA does not mean that interrupt handlers are no longer needed, although the interrupt handlers perform differently than when DMA is not used.

Using DMA can produce the following benefits:

- Reduced system interrupt latency, thereby helping to guarantee data integrity

  Note that this is often a second-order effect (e.g., if system interrupt latency is too long for device A, it may be useful to use DMA with device B to rectify it).

- Reduced bus or CPU cycles

  DMA code consumes fewer bus and CPU cycles than interrupt or polling code.

- Improved communications performance

  Some peripheral devices require DMA for proper operation. For example, the USB peripheral controller in the Am186CC and Am186CU microcontrollers requires DMA to transmit or receive packets which are larger than an endpoint's FIFO size.

8.5.2 DMA Priority

DMA channels can be programmed to three levels of priority (a single value applies to both the transmit and receive channels for one SmartDMA channel pair). Higher priority DMA channels are granted the bus before lower priority channels and can effectively hold off lower priority DMA requests for multiple transfers. When two DMA requests of the same programmed priority have transfers pending, they alternate transfers. When three or more requests of equal priority have transfers pending, they take turns (i.e., 123123123...).

DMA cycles always have priority over internal CPU cycles except between internally locked memory accesses or word accesses to odd memory locations. However, an external bus hold or a refresh cycle takes priority over a DMA cycle.

Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time suffers during sequences of continuous DMA cycles. However, an NMI request causes all internal DMA activity to halt. This allows

the CPU to respond quickly to the NMI request. Software can also inhibit DMA transfers by setting the DHLT bit in the DMAHLT register.

Priorities for the general-purpose DMA channels are set through the GDxCON0 registers; SmartDMA channel priorities are set with the SDxCON registers.

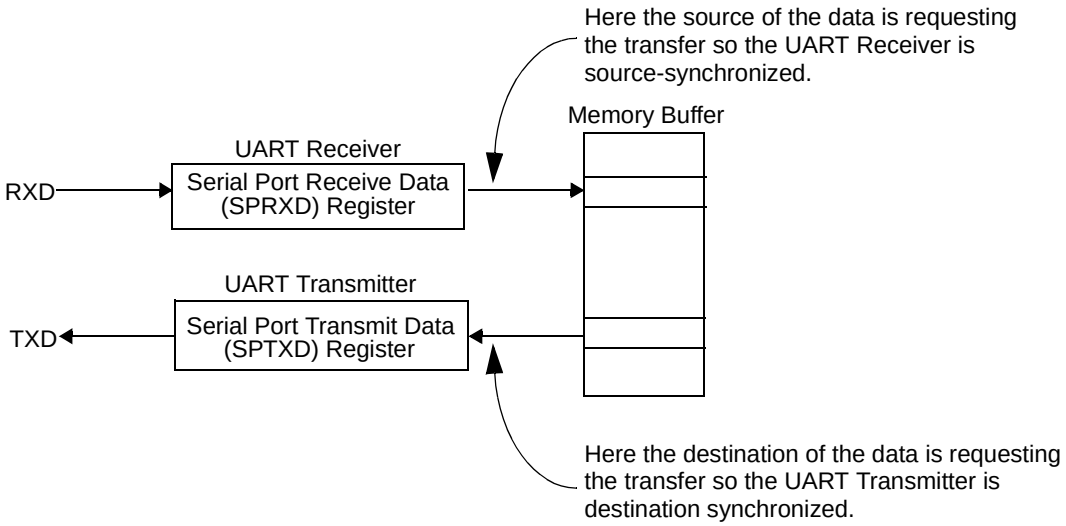
8.5.3 DMA Request Synchronization

Synchronized data transfers are either source or destination synchronized—either the source of the data or the destination of the data generates a DRQ to request the data transfer. Note that the terms source and destination are relative to the data movement. For example, a UART receiver is source-synchronized; the UART is the source of the data and the DRQ (see Figure 8-2).

DMA transfers can also be unsynchronized (i.e., DRQ is always asserted, and the transfer takes place continually until the correct number of transfers has occurred).

For more information about general-purpose DMA channel synchronization, see “Setting Synchronization” on page 8-17. For more information about SmartDMA channel synchronization, see “SmartDMA Channel Request Source and Synchronization” on page 8-27.

Figure 8-2 Source Versus Destination Synchronization



8.5.4 DMA Acknowledge

The Am186CC/CH/CU microcontrollers do not provide an explicit DMA acknowledge signal. Because both source and destination registers are maintained, a read from a requesting source or a write to a requesting destination serves as the DMA acknowledge signal. Because the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA source and destination address registers can point to the same given block, a chip-select line can indicate a DMA acknowledge.

8.5.5 DMA and Interrupts

In some cases, a combination of both DMA processing and interrupt processing is appropriate (e.g., when a certain amount of protocol processing must be performed for each character, and this processing should take place at the interrupt level). In this situation, using a circular receive buffer with extended reads can effectively extend or replace the UART FIFO with a buffer in main memory. For more information, see “Using Buffer Queues

or Circular Buffers" on page 8-20. This method helps guarantee data integrity by ensuring that data is transferred to main memory whether or not the interrupt task can execute. The DMA channel can be set to interrupt immediately on receipt of data by setting the Interrupt (INT) and Terminal Count (TC) bits in the GDxCON0 register. The interrupt task is then a relatively low priority because it does not have to pull the characters out of the UART before they are overwritten by new data. (The effective UART FIFO size has been increased by the DMA buffer size.) When the interrupt task has finished processing all the data in the circular buffer (its read pointer is equal to the destination address), the interrupt can set TC to cause another interrupt as soon as additional characters arrive.

Processing the received data within a low-priority interrupt routine means that flow-control information, such as XONs and XOFFs, may not be seen as quickly. To alleviate this condition, transmission can be done without using DMA (e.g., from within the same interrupt routine, or by programming the interrupt code). In the latter, the interrupt code could program the Transfer Count (GDxTC) register to send a maximum of n characters at a time, using DMA, where '2 • n' is a value that does not overrun the far side's receive FIFO high-water mark.

8.5.6 General-Purpose DMA Channels

The Am186CC/CH/CU microcontrollers each provide four general-purpose DMA channels, which are similar to legacy Am186 general-purpose DMA channels. The four channels can be used for data transfers as shown in Table 8-6.

Table 8-6 General-Purpose DMA Data Transfers

Source	Destination
Memory	Memory I/O Internal peripherals: UART High-Speed UART USB CC CU External peripherals
I/O	Memory I/O Internal peripherals: UART High-Speed UART USB External peripherals
Internal Peripherals: Timer 2[1] UART High-Speed UART USB CC CU	Memory I/O
External Peripherals	Memory I/O

Notes:
1. Timer 2 acts as a DMA request source only; no data is transferred to or from Timer 2.

8.5.6.1 General-Purpose DMA Usage

Note: *Before using the general-purpose DMA channels, ensure multiplexed signals are configured to reflect the use of DMA and not other functionality (see Table 8-1 on page 8-4).*

To use any of the four general-purpose DMA channels, software must perform the following steps. Note that while the source, destination, and count registers can be set in any order, the control registers must be set last.

1. Specify the source address in the GDxSRCL and GDxSRCH source address registers for the corresponding DMA channel.
2. Specify the destination address in the GDxDSTL and GDxDSTH destination address registers for the corresponding DMA channel.
3. Specify the transfer count in the GDxTC count register for the corresponding DMA channel.
4. When performing a DMA transfer to or from a peripheral, configure the DMA before enabling the peripheral. For configuration of peripherals, see the applicable chapter.
5. Configure the DMA transfer options in the GDxCON0 and GDxCON1 control registers for the corresponding DMA channel.
6. Enable DMA transfer in the GDxCON0 and GDxCON1 control registers for the corresponding DMA channel.

All DMA registers except the GDxCON0 and GDxCON1 registers can be modified while the channel is operating. Any changes made to these registers affect the current DMA transfer.

8.5.6.2 General-Purpose DMA Cycle

The four general-purpose DMA channels on the Am186CC/CH/CU microcontrollers are completely interchangeable, and the register sets are identical. From the programmer's point of view, a DMA cycle proceeds as follows:

1. The DMA channel receives a DMA request. Each DMA channel can service requests from Timer 2, an external peripheral, or the internal UARTs.

   CC **CU**

   DMA channels on the Am186CC and Am186CU microcontrollers can also service requests from the USB peripheral controller.

   DMA channels can also be set to unsynchronized, which causes the DRQ to be continuously asserted. This is used for memory-to-memory transfers.

2. The DMA controller reads a byte or word from the programmed source address, which can be in I/O space or in memory, and then writes that byte or word to the programmed destination address, which can also be in I/O space or memory. Unless the channel is set to unsynchronized or to accept requests from Timer 2, the channel should be programmed so that either reading from the source or writing to the destination clears the request. For example, reading from the Serial Port Receive Data (SPRXD) register clears a UART receive DMA request.
3. The source and destination address pointers are then adjusted by independently programmable amounts. The adjustment increment for each pointer can be 0 (e.g., for a peripheral address that does not change), +1, +2, –1, or –2. (Unpredictable results may occur when the transfer size is a word (two bytes) and the adjustment increment is 1 or –1; when the transfer size is one byte and the adjustment increment is +2 or –2, the high byte is ignored.) To implement circular buffers, the pointers can also wrap on 1-, 2-, 4-, 8-, 16-, 32-, or 64-Kbyte boundaries.

4. If the GDxTC register is non-zero, it decrements.
5. If the GDxTC register became zero and either the Terminal Count (TC) bit is set or the transfer type is unsynchronized, the Start/Stop (ST) bit is reset, and further DMA requests on that channel are ignored.
6. If the GD*x*TC register became zero and the Interrupt (INT) bit is set, an interrupt request is generated. This action is independent of whether the TC bit is set.

An entire sequence of DMA cycles is initiated by setting the ST bit. This bit can be set manually at any time, and is also set automatically by any write to the GDxTC register when the Auto Start (AST) bit is set. Setting the ST bit initiates a sequence of DMA transactions if one of the following is true:

- The GDxTC register is non-zero.
- The TC bit is 0 and the transfer type is source or destination synchronized.

If neither of these conditions are met, hardware resets the ST bit without executing any DMA transfers. Otherwise, the ST bit is reset by the hardware after executing one or more transfers as discussed in the previous DMA cycle description. If a transfer is synchronized and the TC bit is 0, DMA transfers continue as long as DMA requests are being made until the ST bit is manually cleared. This mode is typically used with the address wrap option to implement circular buffers (see “Using Buffer Queues or Circular Buffers” on page 8-20).

8.5.6.3 General-Purpose DMA Transfer Suspension

The following conditions suspend general-purpose DMA transfers:

- Deassertion of DRQ
- A bus hold condition
- A refresh cycle by an NMI/watchdog timer interrupt
- A pending DMA request of equal or higher priority
- The DHLT bit in the DMAHLT register set to 1 by an NMI or by software

8.5.6.4 General-Purpose DMA Source and Destination Addresses

Each general-purpose DMA channel has a 20-bit source address and a 20-bit destination address. The 20-bit addresses are split over two source registers (GDxSRCL and GDxSRCH) and two destination registers (GDxDSTL and GDxDSTH), with the four most significant bits (AD19–AD16) going into a separate register from the 16 low-order bits (AD15–AD0). The address is specified as a 20-bit linear address, not as a segment:offset pair. For example, for the segment C000h and offset 1000h, the linear address would be: (C000h x 16) + 1000h = C1000h; therefore, the low register = 1000h and the high register = 0Ch. To use a DMA channel, software must initialize all four address registers for that channel.

The addresses can be individually incremented or decremented after each transfer. For more information, see “Incrementing or Decrementing Addresses” on page 8-15.

The source and destination addresses can each be in either memory space or I/O space. This is specified by programming the SM/$\overline{IO}$ bit in the GDxCON1 register. The AD19–AD16 bits are ignored when the address is in I/O space. Because the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the destination and source address registers. Higher transfer rates can be achieved if all word transfers are performed to and from even addresses so that accesses can occur in single 16-bit bus

cycles. Word transfers to 8-bit address spaces are supported only when the source decrement or increment is 2 bytes.

The Am186CC/CH/CU microcontrollers have the added feature of being able to transfer by DMA to and from the UART and High-Speed UART.

CC CU The Am186CC and Am186CU microcontrollers can also transfer by DMA to and from USB peripherals.

Transfering between DMA and peripherals is accomplished by programming the DMA controller to perform transfers between a data buffer (located either in memory or I/O space) and the peripheral data register. It is important to note that when a DMA channel is in use by a peripheral, the corresponding external DMA request signal is deactivated. For a discussion of using DMA and the on-chip peripherals, see “Selecting DMA Request Sources” on page 8-15.

8.5.6.5 General-Purpose DMA Terminal Count

Each DMA channel has a 16-bit Transfer Count (GDxTC) register. Software must program the GDxTC register with the desired number of transfers and set the Terminal Count (TC) bit in the GDxCON0 register to 1 to enable terminal count. If terminal count is enabled, the channel performs the requested number of transfers, decrementing the value in the GDxTC register after each transfer. When the count reaches zero, the DMA transfer terminates.

If the TC bit is 0, the DMA controller decrements the value of GDxTC after each transfer but does not terminate the transfer when the count reaches zero. The GDxTC register wraps back to its maximum value and continues decrementing. If the current transfer is an unsynchronized transfer, DMA terminates when the count reaches zero.

If the Auto Start (AST) bit in the GDxCON0 register is set, DMA resumes transferring every time the GDxTC register is reloaded with a new value.

CC CU When a channel is connected to a USB transmit endpoint, the Am186CC or Am186CU microcontroller generates a signal internally when the terminal count is reached. The USB peripheral can use this signal to signal the end-of-packet on a transmit.

8.5.6.6 General-Purpose DMA Channel Operations

The general-purpose DMA control registers (GDxCON0 and GDxCON1) determine the DMA channel operations. These registers specify the following options:

- The relative priority of the DMA channel with respect to other DMA channels (see “DMA Priority” on page 8-9)
- Whether the DMA resumes a transfer every time the count register is reloaded with a new value (see “General-Purpose DMA Terminal Count” on page 8-14)
- Whether the source or destination address is in memory or I/O space (see “General-Purpose DMA Source and Destination Addresses” on page 8-13)
- If an interrupt is generated when the transfer count is reached (see “Generating Interrupts” on page 8-15)
- Whether bytes or words are transferred (see “Transferring Bytes or Words” on page 8-15)
- Whether the source or destination address is incremented, decremented, or maintained constant after each transfer (see “Incrementing or Decrementing Addresses” on page 8-15)
- The DMA request source for the channel (see “Selecting DMA Request Sources” on page 8-15)

- The DMA synchronization for the channel (“Setting Synchronization” on page 8-17)
- Whether DMA transfers use buffer queues or circular buffers (see “Using Buffer Queues or Circular Buffers” on page 8-20)

8.5.6.6.1 *Generating Interrupts*

The general-purpose DMA channels can generate an interrupt request when the terminal count value in the GDxTC register reaches 0. To program this feature, set the INT bit in the GDxCON0 register to 1.

8.5.6.6.2 *Transferring Bytes or Words*

The TS bit in the GDxCON0 register can enable either byte or word transfers.

8.5.6.6.3 *Incrementing or Decrementing Addresses*

The source and destination addresses can increment or decrement after each transfer, or remain constant. Specify the action with the SINC and DINC bits in the GD*x*CON1 register. The increment or decrement factor of the source and destination addresses are programmed independently; however, both the source and destination have to be the same size. Word transfers are only supported when the address is incremented or decremented by 2 (an increment by one causes unpredictable results). Byte transfers can be incremented or decremented by 1 or 2. When a byte transfer is incremented or decremented by 2, the high byte is ignored.

Because the DMA controller stores addresses as 20-bit linear values, there are no segment restrictions on the address increment and decrement. However, when using the circular buffer feature, the address boundary is limited to the size of the buffer. For example, when using a 1-Kbyte circular buffer, the address has to start at a 1-Kbyte boundary. For more information, see “Using Buffer Queues or Circular Buffers” on page 8-20.

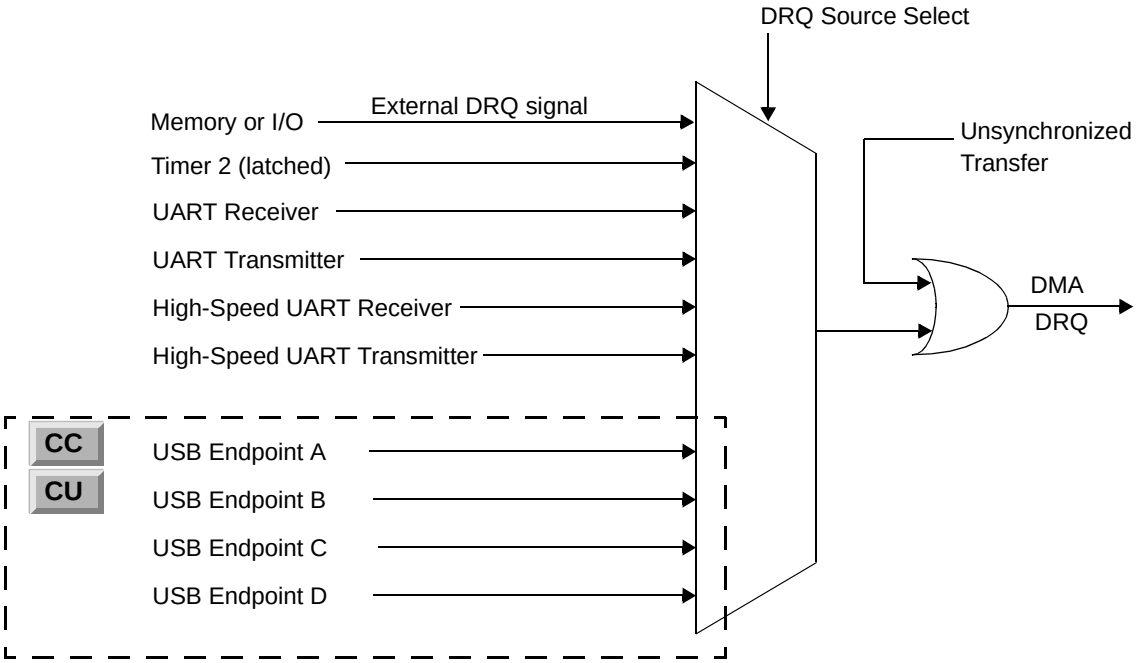
8.5.6.6.4 *Selecting DMA Request Sources*

The DSEL bit field in the GDxCON0 register sets the DMA request source for that channel. As shown in Figure 8-3, the DMA request source can be an external DRQ signal, Timer 2, UART receiver, UART transmitter, High-Speed UART receiver, or High-Speed UART transmitter.

CC **CU** The Am186CC and Am186CU microcontrollers also support USB endpoints A, B, C, or D as request sources. Each USB endpoint can be configured either for receive or transmit.

In addition to setting the DMA request source, the DSEL bit field also selects the synchronization type (see “Setting Synchronization” on page 8-17).

Figure 8-3 DMA Request Sources



DMA Request from Timer 2

The GDxCON0 register can configure a DMA channel to accept the output of Timer 2 as a DRQ signal to generate periodic data transfers. Note that this feature generates a DRQ periodically—even if there is no data. To use this feature, software should program Timer 2 with the T2CON register for “continuous” mode—to reach the maximum count and start counting again. Each time the timer reaches the maximum count, it generates a single DRQ. The DMA controller latches the Timer 2 DRQ signal to guarantee that the DMA channel does not miss the DRQ if the bus is not immediately available (e.g., if a higher priority DMA has control of the bus). If a second Timer 2 DRQ is generated before the first request is serviced, the second request is lost.

DMA Request from UARTs

Transfers between the DMA and the UARTs are accomplished by programming the GDxCON0 register to perform transfers between a data buffer (located either in memory or I/O space) and a serial port data register (SPTXD, SPRXD, HSPTXD, or HSPRXD).

Note: *Using a DMA channel with a UART deactivates the corresponding external DMA request signal.*

For DMA *to* the UART or High-Speed UART, specify the following configuration details for the DMA by writing the address of the register into the GDxDSTL and GDxDSTH registers: the transmit data register (SPTXD or HSPTXD) address; either I/O-mapped or memory-mapped; as a byte destination, or word destination if using extended writes. The destination address (the address of the transmit data register) should remain constant throughout the DMA operation.

For DMA *from* the UART or High-Speed UART, specify the following configuration details for the DMA by writing the address of the register into the GDxSRCL and GDxSRCH registers: the receive data register (SPRXD or HSPRXD) address; either I/O-mapped or memory-mapped; as a byte source, or word source if using extended writes. The source address (the address of the receive data register) should remain constant throughout the DMA operation.

DMA Request from USB

CC CU Because USB can use either general-purpose DMA or SmartDMA channels, this is discussed separately in “DMA and USB” on page 8-43.

8.5.6.6.5 Setting Synchronization

The DSEL bit field in the GDxCON0 register sets the DMA request source for that channel (see “Selecting DMA Request Sources” on page 8-15). Unlike prior Am186 parts, this bit also sets the synchronization. General-purpose DMA transfers can be unsynchronized, source-synchronized, or destination-synchronized.

The source or destination device implies the synchronization type as shown in Table 8-7.

DMA synchronization affects the behavior of the DMA operation and system performance as a whole. All DMA transfers observe the programmed ready and wait-state conditions for any chip select active for that cycle.

DRQ must be deasserted before the end of the DMA transfer to prevent another DMA cycle from occurring. The timing for the required deassertion depends on whether the transfer is source-synchronized or destination-synchronized.

Table 8-7 General-Purpose DMA Request Source and Synchronization

DMA Request Source	Synchronization Type
Memory or I/O	Unsynchronized
Timer 2	Source
UART Receiver	Source
UART Transmitter	Destination
High-Speed UART Receiver	Source
High-Speed UART Transmitter	Destination
USB Requset Sources CC CU	
USB Endpoint A Receiver	Source
USB Endpoint A Transmitter	Destination
USB Endpoint B Receiver	Source
USB Endpoint B Transmitter	Destination
USB Endpoint C Receiver	Source
USB Endpoint C Transmitter	Destination
USB Endpoint D Receiver	Source
USB Endpoint D Transmitter	Destination

Unsynchronized Transfers

For unsynchronized DMA transfers, the DRQ signal is internally tied High. When initiated, an unsynchronized DMA transfer begins immediately and consumes all bus cycles until the terminal count value in the GDxTC register reaches 0. Unsynchronized DMA is generally used for copying data between memory locations, between I/O locations, or between memory and I/O locations. For example, unsynchronized DMA can initialize RAM during start-up.

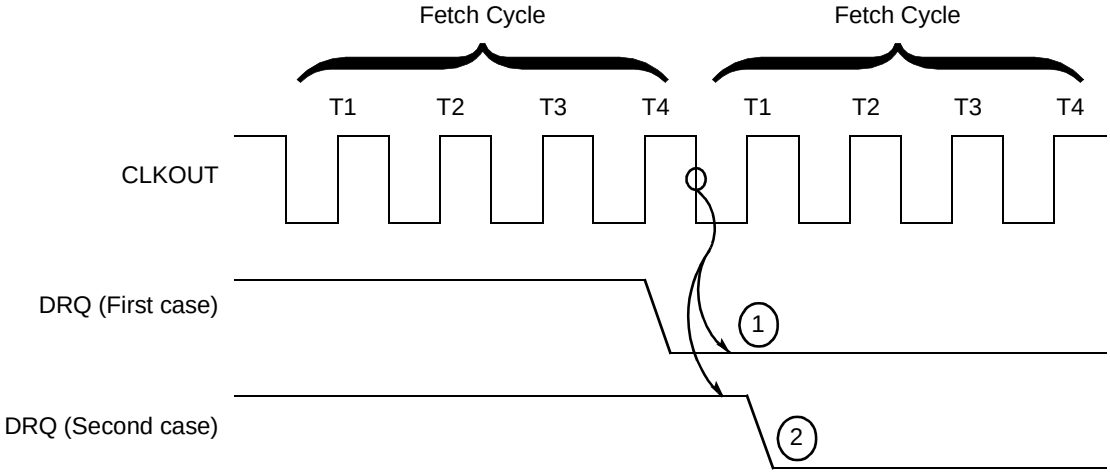
Source-Synchronized Transfers

Source-synchronized DMA transfers require either an internally generated DRQ (e.g., from a UART receiver) or an external device that asserts the associated external DRQ signal for

a general-purpose DMA channel. In source synchronization, the device providing the data asserts the DMA request.

Figure 8-4 shows a typical source-synchronized DMA transfer. When an external device is asserting DRQ, the request must be deasserted at least four clock cycles before the end of the transfer (at T1 of the deposit phase) to prevent another transfer from taking place. If more transfers are not required, a source-synchronized transfer allows the source device at least three clock cycles from the time it is acknowledged to deassert its DRQ line. Like unsynchronized DMA transfers, source-synchronized DMA transfers have the capability of consuming all bus cycles if the DRQ remains asserted for multiple transfers. An example of this would be the emptying of a FIFO.

Figure 8-4 Source-Synchronized General-Purpose DMA Transfers



Notes:

1. *This source-synchronized transfer is not followed immediately by another DMA transfer, because DRQ is deasserted at least four clock cycles before the end of the transfer.*
2. *This source-synchronized transfer is immediately followed by another DMA transfer, because DRQ is not deasserted soon enough.*

Destination-Synchronized Transfers

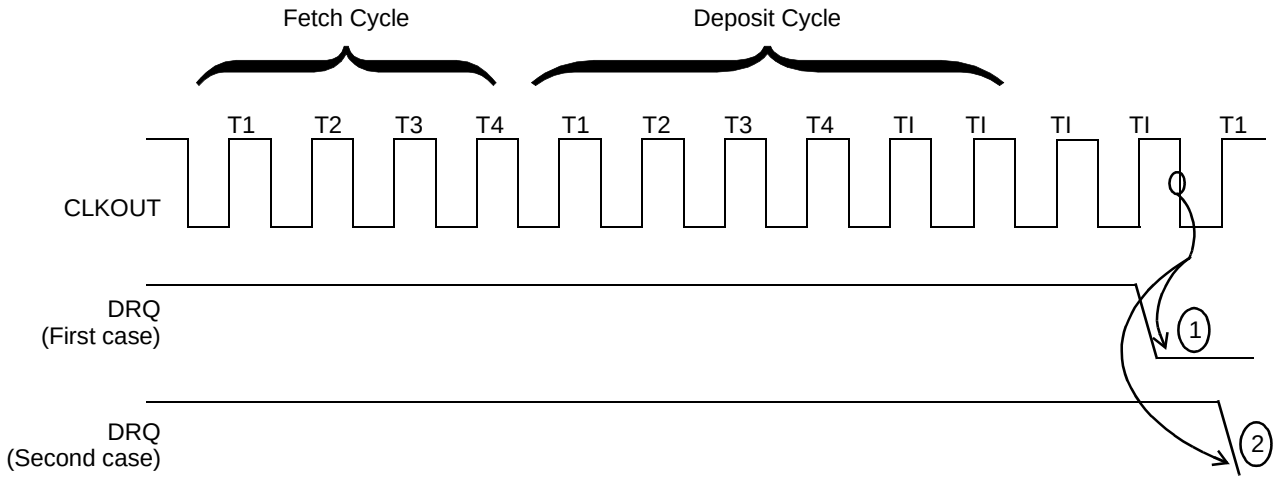
Destination-synchronized DMA transfers require either an internally generated DRQ (e.g., from a UART transmitter), or an external device that asserts the associated external DRQ signal for a general-purpose channel. In destination synchronization, the device receiving the data asserts the DMA request.

Figure 8-5 shows a typical destination-synchronized DMA transfer. The DMA controller does not sample the DRQ line for a channel until four cycles after the end of the write phase of a destination-synchronized DMA transfer. This delay allows the external device sufficient time to remove its request if it does not want another transfer. The delay also allows other devices access to the bus, including instruction or data fetches by the processor and other DMA transfers (including transfers by lower priority DMA requests). If another device starts a bus cycle during the DMA idle cycles, the entire bus cycle completes before giving the bus back to the DMA. If no other bus activity is initiated, another DMA cycle begins.

Because the DMA controller relinquishes the bus after every destination-synchronized transfer, the CPU can initiate a bus cycle. As a result, a complete bus cycle is often inserted

between destination-synchronized transfers. Table 8-8 shows the maximum DMA transfer rates based on the different synchronization strategies.

Figure 8-5 Destination-Synchronized General-Purpose DMA Transfers



Notes:

1. *This destination-synchronized transfer is not followed immediately by another DMA transfer, because DRQ is deasserted during the four idle states.*
2. *This destination-synchronized transfer is immediately followed by another DMA transfer, because DRQ is not deasserted soon enough.*

Table 8-8 Maximum DMA Transfer Rates

Synchronization Type	Maximum DMA Transfer Rate (Mbytes/s)		
	50 MHz	40 MHz	25 MHz
Unsynchronized	12.5	10	6.25
Source-synchronized	12.5	10	6.25
Destination-synchronized (CPU needs bus)	8.33	5	3.125
Destination-synchronized (CPU does not need bus)	10.0	5	3.125

Deasserting DRQ

In externally synchronized transfers, DRQ1 or DRQ0 must be deasserted before the end of the DMA transfer to prevent another DMA cycle from occurring. The timing for the required deassertion depends on whether the transfer is source-synchronized or destination-synchronized.

A DMA request is not acknowledged from the same source for four processor clock cycles after the end of the deposit cycle. In a source-synchronized DMA transfer, the DRQ signal must be deasserted at least four clocks before the end of the transfer. If more transfers are not required, a source-synchronized transfer allows the source device at least three clock cycles from the time it is acknowledged to deassert its DRQ line. For more information, see “DMA Acknowledge” on page 8-10.

A destination-synchronized transfer differs from a source-synchronized transfer in that the four cycle delay allows the destination device to deassert its DRQ signal four clocks before another request is latched. Without this delay, the destination device would not have time to deassert its DRQ signal. Because of the four extra cycles, a destination-synchronized DMA channel allows other bus masters to take the bus during the idle states.

8.5.6.6.6 *Using Buffer Queues or Circular Buffers*

Note: *This discussion assumes the channel is using a memory buffer (i.e., that the source or destination address is programmed to increment or decrement). If the address is programmed to remain constant, no memory buffer is in use, and neither buffer queues nor circular buffers are used. See "Incrementing or Decrementing Addresses" on page 8-15 for more information.*

The GDxCON1 register contains two fields that specify whether the source and/or destination addresses for that DMA channel should wrap when the addresses reach a programmed boundary. These fields are programmed independently; wrapping could be enabled for one address and not for the other. When wrapping is disabled, the memory buffer is treated as a linear array. This is typically called a *buffer queue*.

With a buffer queue, data is written/read to sequential byte or word addresses until a terminal count is reached. The DMA should be programmed to terminate when the terminal count is reached, or data may be written past the end of the buffer.

When wrapping is enabled, the memory buffer is treated as a *circular buffer* (sometimes called a *ring buffer*). In this case, data is written/read to sequential byte or word addresses until the programmed buffer length is reached, at which point the address is reset to its initial value; data is never written outside the programmed buffer space. Circular buffers can be programmed to be 1, 2, 4, 8, 16, 32, or 64 Kbytes in length, and must be aligned to an address which is a multiple of the programmed size.

The use of a circular buffer reduces the overhead required in programming the DMA channel and may result in more efficient use of the transmitting or receiving device. However, in the case where a circular buffer is being used to receive data, software must ensure that valid received data is removed from the buffer before it is overwritten by the DMA controller on the next pass. Conversely, for transmit circular buffers, software must write valid transmit data into the buffer before that buffer address is read by the DMA controller.

To avoid overwriting data in a circular buffer, compare the source address with the buffer address. For example, the address contained in a DMA channel's source address registers is the address of the next byte of data to be transmitted. Data that is logically between this address and the buffer address being written to by software (in a circular fashion) has not yet been transmitted. If the source address registers contain the address xxxx0050 when software is writing to address xxxx0150, then the addresses from xxxx0050 through xxx0149 contain valid data for transmission. Addresses outside of this range, but within the buffer, do not contain valid data. This may be data that has already been transmitted or may be addresses that have never been written.

For string data or other data which is naturally represented as consecutive bytes or words in memory, using circular buffers involves additional overhead because the data must be moved between its storage location and the circular buffer. For these data types, a buffer queue may be a more efficient solution.

Example of Using Buffer Queues and Circular Buffers with the UARTs

Note: *This section discusses implementation tradeoffs for using the general-purpose DMA channels. To have a concrete system to discuss, the integrated UART and High-Speed UART are used as examples, but much of this information is applicable to using the general-purpose DMA channels with other peripherals as well. In general, there are two distinct ways that general-purpose DMA can be used: buffer queues and circular buffers. These two techniques are discussed and contrasted below. In addition, these two methods can be mixed (e.g., queues of messages for transmit and circular buffers for receive). A careful analysis of the final system is required to determine the best method to use.*

Many systems, especially those communicating with other equipment rather than with human beings in interactive mode, transmit and receive messages in large blocks. These messages can be forwarded to a host PC through the USB interface, or forwarded through an ISDN line or other WAN setup using HDLC. With this sort of message protocol, it may be advantageous under some circumstances to perform a DMA transfer directly to or from a queue of buffers, rather than to or from a single circular buffer per direction.

Buffer queues are a viable way to transfer data to and from some devices with general-purpose DMA. However, buffer queues are only useful for DMA with the UARTs under very special circumstances.

The primary advantage to using DMA transfer straight from a queue of buffers is the reduction of data motion. Transmission is relatively straightforward: the DMA channel is programmed with the correct source address and transfer count for each buffer; and the DMA channel is set up to stop transmitting and to interrupt when the end of the buffer is reached. When the interrupt occurs, the buffer is freed, and the next buffer is set up to be transferred out using DMA.

Reception is more difficult because it is not always known up front exactly how long the incoming message is. Even if the message size is fixed, line errors can corrupt the perceived length. For both reception and transmission, issues such as compression, transparency, and CRC generation and checking mean that software must usually examine each character individually. In this case, using a circular buffer is generally the best way to use DMA with a UART (because each character is being read by software anyway, and the number of characters to be transmitted is different than the number of characters in the original buffer), although DMA is not necessarily the best way to transfer data to and from the UART.

The determination of whether to use DMA at all for this sort of protocol processing is dependent on system loading and maximum UART baud rate. If CPU cycles are at a premium (e.g., for data compression), it may be worthwhile to use DMA.

Many UART serial drivers use circular buffers for temporary storage of incoming and outgoing characters. The primary drawback to using a circular buffer is that it doubles the bus bandwidth required to handle each character received or transmitted. For example, if a string is written out to a serial port, using a circular buffer requires four bus transactions for each character (read it from the string, write it to the buffer, read it from the buffer, write it to the transmit port), whereas without the buffer, two transactions would suffice (read the character from the string, write it to the transmit port). Nevertheless, circular buffers are popular because the alternative often requires more coding and is usually more error-prone (e.g., a buffer containing a string could inadvertently be reused before the string is completely transmitted).

For this reason, the Am186CC/CH/CU microcontrollers' DMA has excellent circular buffer support. With the general-purpose DMA channels, this is achieved by setting bits in the GDxCON1 register to a nonzero value to select a buffer size between 1 and 64 Kbytes.

Software must ensure that this buffer is aligned on a multiple of its size. This is easily done for statically allocated buffers with a good linker/locator; for dynamically allocated buffers, the software must waste the size of one buffer. This waste can usually be reduced or eliminated by allocating and deallocating additional buffers; this is highly dependent on the operating system and memory allocation library.

Transmitting using DMA and circular buffers is easy and does not require interrupt support. Note that using DMA for transmission for the UART is not required for data integrity reasons, so DMA should only be used for UART transmissions if one of the following applies:

- CPU throughput can be improved by reducing the time spent in the UART interrupt handler.
- Transmission throughput can be improved by reducing the latency between transmitted characters. Note that it is easy to measure intercharacter latency to determine the maximum possible improvement available by improving UART transmission.

Table 8-9 gives typical register values for using circular buffers with the UARTs.

Table 8-9 Example Register Settings for UARTs and Circular Buffers

General-Purpose DMA Register	Bit(s) in Register	Value for Transmit DMA	Value for Receive DMA
GDxCON0 (Control 0)	ST (Start/Stop)	Clear (is set by load of GDxTC register)	Set after all other fields and UART set correctly
	AST (Auto Start)	Set	Clear
	TC (Terminal Count)	Set	Typically clear
	INT (Interrupt)	Clear for single-tasking; set for multitasking	Set
	P (Relative Priority)	Depends on rest of system	Depends on rest of system; higher than transmit
	TS (Transfer Size)	0 for 8 bit, 1 for 9 bit	0 for 8 bit, 1 for 9 bit or extended status
	DSEL (DMA Request Select)	(High-Speed) UART Transmitter	(High-Speed) UART Receiver
GDxCON (Control 1)	SM/$\overline{IO}$ (Source Address Space Select)	Set (memory)	Clear (I/O)
	SAW (Source Address Wrap)	Set to size of buffer	Clear
	SINC (Source Increment)	1 for 8 bit, 2 for 9 bit	0
	DM/$\overline{IO}$ (Destination Address Space Select)	Clear (I/O)	Set (memory)
	DAW (Destination Address Wrap)	Clear	Set to size of buffer
	DINC (Destination Increment)	0	1 for 8 bit, 2 for 9 bit or extended status
GDxSRCL (Source Address Low)	DSA[15–0]	Buffer address MOD 64K	(H)SPRXD

Table 8-9 Example Register Settings for UARTs and Circular Buffers (Continued)

General-Purpose DMA Register	Bit(s) in Register	Value for Transmit DMA	Value for Receive DMA
GDxSRCH (Source Address High)	DSA[19–16]	Buffer address DIV 64K	0
GDxDSTL (Destination Address Low)	DDA[15–0]	(H)SPTXD	Buffer address MOD 64K
GDxDSTH (Destination Address High)	DDA[19–16]	0	Buffer address DIV 64K
GDxTC (Transfer Count)	TC	Set to total string length whenever writes performed	Set to high-water mark

When the DMA channel is initialized as shown in Table 8-9, transmitting a string is performed as follows:

1. Copy the string from the source to the buffer at the current write pointer position, being careful to take buffer wrap into account, and being careful not to overwrite data already in the buffer which is not yet transmitted. (Whether or not data has been transmitted can be easily determined by reading the DMA channel's source address register and comparing it against the write pointer position.) Calculate a new write pointer position, and save it in memory to use for subsequent writes.
2. Stop the DMA (reset the ST bit) to ensure that the transfer count is stable.
3. Add the length of the new string to the Transfer Count (GDxTC) register. A read/modify/write cycle adjusts the GDxTC register, and the write to the register automatically restarts the DMA (if the AST bit is set).

Interrupts are required for transmit under the following conditions:

- If the XON/XOFF protocol is used, the DMA must be shut off to transmit flow control characters, and also to stop transmitting when an XOFF is received. One way this protocol could work is that when DMA is to be stopped, the AST and ST bits can be reset; when DMA is to be restarted, both these bits can be set again. If this technique is used, steps 2 and 3 above (stopping DMA and updating transfer count) should be performed as an atomic operation (i.e., with interrupts disabled) to avoid conflicts with the XON/XOFF interrupt handler.
- If a multitasking system is used, an attempt to write too much data to the buffer should write as much as possible, and then block the task performing the write until additional space is available. In this case, the GDxTC register should not be programmed with the actual count of characters in the buffer, but instead be programmed with the lesser of the actual count and the amount of space to wait for before restarting the blocked task. Also, the interrupt bit should be set. When the interrupt occurs, the transfer count should immediately be reprogrammed to the actual remaining buffer count to avoid delay in transmission, and the blocked task should be marked as ready to run.

Reception Using Circular Buffers

For the UART, DMA reception using a circular buffer is potentially more useful than transmission because transferring received characters into a circular buffer can help improve data integrity. (Characters are never lost due to interrupt latency.)

Like basic transmission, basic reception using a circular buffer is simple. Software maintains a read pointer into the buffer and can dynamically determine the number of characters available for reading at any time by reading the destination address, subtracting the read

pointer from it, and dividing the result by the buffer size. The remainder of this division is the number of bytes available for reading.

The difficulty again revolves around multitasking and flow control, with the added problem of error handling.

Receive XON/XOFF Flow Control

XON/XOFF flow control with DMA is problematic because, in general, the received flow control characters should not be stored in the buffer, and also because the characters should typically be acted on immediately. This may make DMA impractical for implementing XON/XOFF flow control with the UART. The High-Speed UART can be programmed to stop using DMA and interrupt the CPU whenever a flow control character arrives, so that an interrupt routine can act on the flow-control character and then restart the DMA operation. Note that baud rate, system latency, and the depth of the FIFO must be considered when determining if this is practical for a given implementation.

In addition to detecting and acting on flow-control characters in the data stream, the receiving task must also detect when the circular buffer is getting full so that an XOFF can be sent. This can be accomplished by programming the Transfer Count register with a value that is the current room available in the buffer minus a constant high-water mark, and setting the INT bit, but not the TC bit in the GDxCON0 register. This causes the DMA to interrupt when there is room (buffer size minus high-water mark) in the buffer, and an XOFF can be sent. The value chosen for the high-water mark should take into account far-end latency in dealing with an XOFF, plus latency associated with sending the XOFF through the High-Speed UART transmit FIFO, if it is enabled.

Receive Hardware Flow Control

Hardware flow control is simple if the connected device performs true hardware flow control (i.e., stops transmitting on the next character boundary when RTR is dropped).

Some UARTs and systems perform pseudo-hardware flow control. In these UARTs, the flow control signal can cause an interrupt, but may not stop characters already queued to go out. In this case, the High-Speed UART's receive FIFO may be sufficient to guarantee that overruns do not occur.

If the UART is being used, or if the High-Speed UART's receive FIFO is not large enough to guarantee that the other side will stop quickly enough, then RTR should be performed using a PIO, and an algorithm similar to the one described for receive XON/XOFF control should be used, so that the far side is requested to stop sending before the DMA buffer is actually full. The FIFO threshold is one half of the FIFO depth and is not programmable.

If the attached device is capable of real hardware flow control, then the TC bit in the GDxCON0 register can be set, and the Transfer Count register can be programmed with the amount of room left in the buffer. When DMA ceases, the hardware flow control signal is automatically asserted.

Receive Multitasking

In a single-tasking system, received characters are held in the circular buffer until higher-level code is ready for them. In a multitasking system, it may be desirable for receipt of characters to cause an interrupt to signal that a task switch should take place. When DMA is used with one of the UARTs on the Am186CC/CH/CU microcontrollers, the hardware is typically programmed to cause interrupts under these two conditions:

- After a programmed number of characters have been received.

  To do this, the programmed transfer count is usually the lesser of this desired number and the number required to implement proper flow control.

- When no characters have been received for a certain period of time (signifying that the other end has probably finished transmitting a message).

  This is accomplished by interrupting on the UART IDLED bit in the (H)SPSTAT register, which causes an interrupt when 40 bit times have gone by without detecting a start bit.

Receive Error Processing

Several kinds of errors and exceptional conditions can occur when receiving asynchronous character data. Breaks, parity errors, and framing errors indicate an exceptional external condition. Overrun errors indicate that data has been lost due to system latency. A character match interrupt (High-Speed UART only) can indicate that an XON or XOFF has been received.

The Am186CC/CH/CU microcontrollers offer great flexibility in dealing with these exceptions. If the EXDRD bit in the UART's Control 1 register is set, and the DMA is set up to transfer a word for every character, exceptions can be stored in the circular buffer along with the character that caused them. If the EXDRD bit is reset, exceptions that cause interrupts cause DMA activity to stop until an interrupt task services the exception.

The decision of whether to set or clear the EXDRD bit depends on the intended usage. If the target baud rate is high relative to system loading, setting the EXDRD bit can prevent the loss of data due to interrupt latency. This is especially true when using the UART, or when using the High-Speed UART without the FIFO. If break or address bit information is to be stored with the character for later retrieval, setting the EXDRD bit is appropriate. Setting this bit can complicate system software, especially if XON/XOFF flow control is used, because the flow control characters are stored in the circular buffer. The system software must find the character and perform the correct action immediately, and then ignore the character when reading data out of the buffer later.

When using the FIFO on the High-Speed UART, these exceptions (break, parity error, framing error, address bit, overrun error, and character match) are placed in the FIFO and move with the associated data so that the software can match the exception with the correct character. This means that a system programming error that keeps data from being pulled out of the FIFO (e.g., misprogramming of the DMA) may keep interrupts from ever occurring. For this reason, the High-Speed UART has an additional Overrun Error-Immediate (OERIM) interrupt bit that is not placed in the FIFO. Software can monitor or interrupt on OERIM to detect and correct this sort of system programming error.

Small or Misaligned Circular Buffers

If a circular buffer is smaller than 1K, is not aligned on a multiple of its size, or the size is not a power of 2, the address wrap features of the DMA are not available. The DMA can still be used to implement a circular buffer, but it requires more programming effort, and the required interrupt could introduce unacceptable latency into the system. System software must always use interrupts, set the TC bit, and carefully program the Transfer Count register

so that the address never exceeds the boundary of the buffer. Software must manually wrap the buffer address back to the start of the buffer whenever an interrupt signifies the end of the buffer. This is not too burdensome for UART transmit buffers because there is no hard latency requirement for asynchronous transmission, but this could be a problem for receive buffers if the interrupt latency could cause characters to be missed.

8.5.7 SmartDMA Channels

The Am186CC/CH/CU microcontrollers each contain SmartDMA channels, compatible with the DMA in the AMD Am79C90 C-LANCE (Local Area Network Controller for Ethernet). This LANCE-compatible buffer descriptor ring interface provides a method for transmission and reception of data across multiple memory buffers. The ring descriptor interface also provides a method for reporting status on multiple received and transmitted packets while ensuring that status information is always correctly linked with the associated data.

Unlike the general-purpose DMA channels, which can be used for memory-to-memory or I/O-to-I/O transfers, the SmartDMA channels are highly specialized. These channels must be used in pairs. Each pair consists of a transmit channel and a receive channel. *The transmit channels transfer data from memory to a transmitting device (such as an HDLC transmitter). Receive channels transfer data from a receiving device (such as an HDLC receiver) to memory.*

CC Four of the eight SmartDMA channels (two pairs) in the Am186CC microcontroller are dedicated for use with the on-board HDLC channels. The remaining four SmartDMA channels (two pairs) can support either the third or fourth HDLC channel or USB endpoints A, B, C, or D.

CH The four SmartDMA channels (two pairs), SDMA0 and SDMA1, in the Am186CH HDLC microcontroller support the two on-board HDLC channels.

CU The four SmartDMA channels (two pairs), SDMA2 and SDMA3, in the Am186CU USB microcontroller support USB endpoints A, B, C, or D.

This section describes these SmartDMA channels.

8.5.7.1 SmartDMA Channels Introduction

With a traditional DMA controller, such as the general-purpose DMA, the typical mode of operation is to DMA transfer a buffer of information (either filling a receive buffer, or emptying a transmit buffer) and program the DMA controller to interrupt the CPU when the end of the buffer is reached.

However, if the data rate is high relative to system loading and interrupt latency, data could be lost before the interrupt service routine reinitializes the DMA controller to point to the next buffer. For some peripherals, such as UARTs, this problem is easily solved by the ability of the general-purpose DMA controller to manage a circular buffer. If such a circular buffer is managed correctly, DMA is never halted to wait on CPU interrupt activity.

A circular buffer does not work as well for packet-oriented communications such as HDLC and USB because of the requirement to delineate packet boundaries. Also, in a typical system, each packet can be routed to a different destination, so the data would have to be copied out of the circular buffer and into another peripheral's circular buffer.

SmartDMA channels solve these problems by maintaining a circular queue of buffer descriptors—a *descriptor ring*—rather than a circular data buffer. The hardware itself updates a buffer descriptor when a full buffer is transferred, then automatically fetches the next descriptor and starts transferring data to the new buffer. Because the hardware

performs this operation without software intervention, the latency is significantly lower than if an interrupt task performed the same operation.

Software must still read and write the buffer descriptors, but the latency requirements are greatly relaxed because multiple descriptors are queued at one time. The software can take, on average, the time it takes to transmit or receive a buffer to update each descriptor, and software can increase allowable latency even more by updating several descriptors at the same time.

8.5.7.2 SmartDMA Channel Request Source and Synchronization

The SmartDMA channels support only specific, predetermined request sources. These sources in turn determine the synchronization type for each channel. Synchronization type for the SmartDMA channels is not programmable. The synchronization types are shown in Table 8-10, Table 8-11, and Table 8-12.

The memory buffer addresses are taken from the buffer descriptor ring, as explained in “SmartDMA Channel Memory Overview” on page 8-28.

CC In the Am186CC microcontroller, SmartDMA Channel 2 and SmartDMA Channel 3 provide the DSEL bit in the SDxCON control register for selecting between the HDLC and USB request source. The address of the peripheral does not need to be programmed into the SmartDMA channel because these devices have an internal interface to the associated SmartDMA channel.

CU In the Am186CU USB microcontroller, the DSEL bit must be programmed correctly for USB support.

The SmartDMA channels support only byte transfers. The data is written or read from sequential byte addresses in the memory buffers.

Table 8-10 Am186CC SmartDMA Channel Request Source and Synchronization

CC

SmartDMA Channel	Direction	Source	Destination	Synchronization
0	Transmit	Memory buffer	HDLC A transmit FIFO	Destination
	Receive	HDLC A receive FIFO	Memory buffer	Source
1	Transmit	Memory buffer	HDLC B transmit FIFO	Destination
	Receive	HDLC B receive FIFO	Memory buffer	Source
2	Transmit	Memory buffer	HDLC C transmit FIFO or USB endpoint B transmit FIFO	Destination
	Receive	HDLC C receive FIFO or USB endpoint A receive FIFO	Memory buffer	Source
3	Transmit	Memory buffer	HDLC D transmit FIFO or USB endpoint D transmit FIFO	Destination
	Receive	HDLC D receive FIFO or USB endpoint C receive FIFO	Memory buffer	Source

Table 8-11 Am186CH SmartDMA Channel Request Source and Synchronization

CH

SmartDMA Channel	Direction	Source	Destination	Synchronization
0	Transmit	Memory buffer	HDLC A transmit FIFO	Destination
	Receive	HDLC A receive FIFO	Memory buffer	Source
1	Transmit	Memory buffer	HDLC B transmit FIFO	Destination
	Receive	HDLC B receive FIFO	Memory buffer	Source

Table 8-12 Am186CU SmartDMA Channel Request Source and Synchronization

CU

SmartDMA Channel	Direction	Source	Destination	Synchronization
2	Transmit	Memory buffer	USB endpoint B transmit FIFO	Destination
	Receive	USB endpoint A receive FIFO	Memory buffer	Source
3	Transmit	Memory buffer	USB endpoint D transmit FIFO	Destination
	Receive	USB endpoint C receive FIFO	Memory buffer	Source

8.5.7.3 SmartDMA Channel Memory Overview

Figure 8-6 on page 8-29 and Figure 8-7 on page 8-30 illustrate how SmartDMA channels use memory. Each SmartDMA channel (both transmit and receive) has two registers that contain the base descriptor ring address and the number of entries in the descriptor ring. (A *descriptor ring* is merely a block of memory that the CPU and software use to control and describe data buffers.)

There are two descriptor rings for each SmartDMA channel: one for transmit and one for receive. The SDxTRCAL and SDxRRCAL registers contain the three bits that encode the number of entries in the ring, and 12 bits (bits 15–4) to determine the 12 low address bits of the descriptor ring address, which is the start location in memory of the buffer descriptor ring. The SDxTRAH and SDxRRAH registers contain the four high bits (19–16) of the addresses. Because the base address of the ring must be paragraph aligned (aligned to a 16-byte physical memory boundary), address bits 3–0 are always zeros. The address is specified as a 20-bit linear address, not as a segment:offset pair. For example, for the segment C000h and offset 1000h, the linear address would be: (C000h x 16) + 1000h = C1000h; therefore, the low register = 1000h and the high register = 0Ch.

The size of the transmit and receive descriptor rings (the ring count) is independently programmable to 1, 2, 4, 8, 16, 32, 64, or 128 descriptors. Even when the ring size is set to 1, that entry is still interpreted as a descriptor, not as the memory buffer itself.

Each entry in the descriptor ring is composed of a 20-bit linear address for a buffer, an owner semaphore bit, a frame-start indicator bit, a frame-end indicator bit, a terminal count interrupt bit, and a 15-bit buffer byte count. Other fields are also present but are dependent on whether the descriptor is in a transmit or receive descriptor ring.

The address in each descriptor ring entry contains the address of the data buffer pointed to by that entry. Note that the HDLC and USB peripheral controllers transmit data packets, which contain a block of data between a start and end indicator. A packet (e.g., all the HDLC data between two HDLC flag bytes) can be broken up into multiple buffers, but a buffer cannot contain data for different packets.

The owner semaphore (OWN) bit is a single-bit field in each buffer descriptor. This bit is set by software when the buffer is valid—either it contains valid data for transmission or it is available to be overwritten by the receiver. The SmartDMA controller never sets the OWN bit.

Software must never clear the OWN bit while the SmartDMA controller is active—the software should first stop the DMA operation by resetting the TXST or RXST bit in the SDxCON register. (If the SmartDMA controller is already working on that buffer, clearing the OWN bit has no effect; if the SmartDMA controller was going to get the buffer, it would be in poll mode and wait until the buffer is available.) The SmartDMA controller clears the OWN bit when it releases control of the buffer.

Some systems may need to have DMA transfer continue even if software has not kept up with the DMA. This can be accomplished by setting the TXS0 bit in SDxCON for the transmit DMA channel or the RXS0 bit in SDxCON for the receive DMA channel. Setting these bits inhibits the associated SmartDMA channel from clearing the OWN bit after it is through processing a buffer.

Note: *Take care when setting these bits, because you may lose received data or transmit stale data.*

Figure 8-6 SmartDMA Channel Descriptor Ring Example

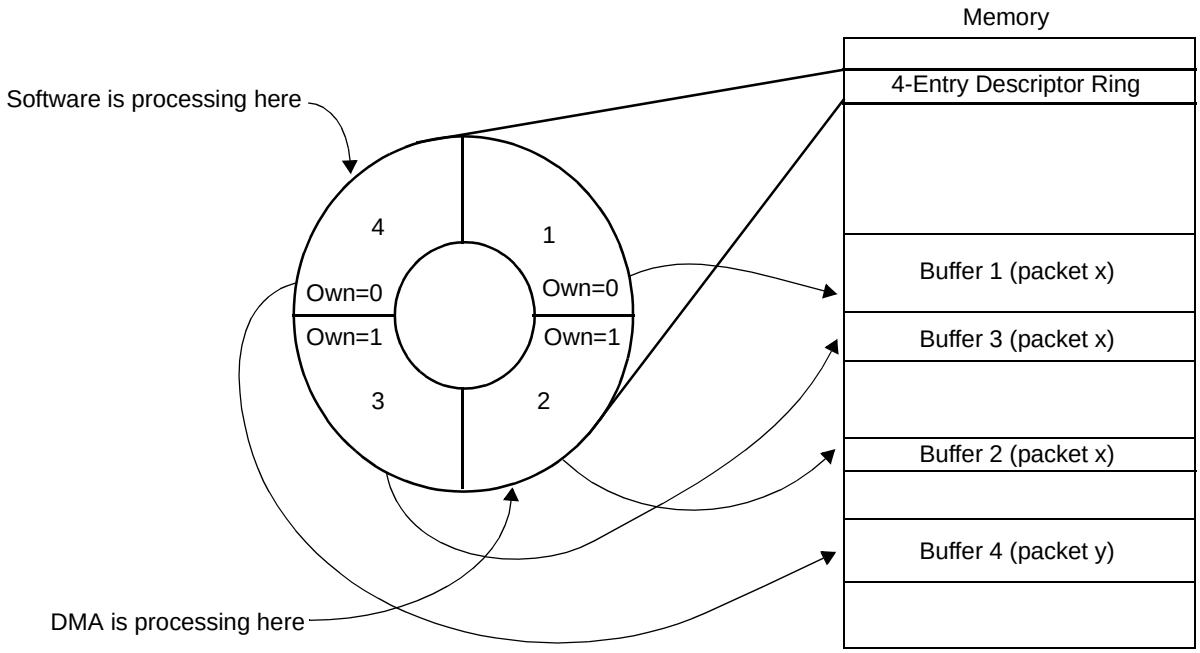


Figure 8-7 SmartDMA Channel Memory Management

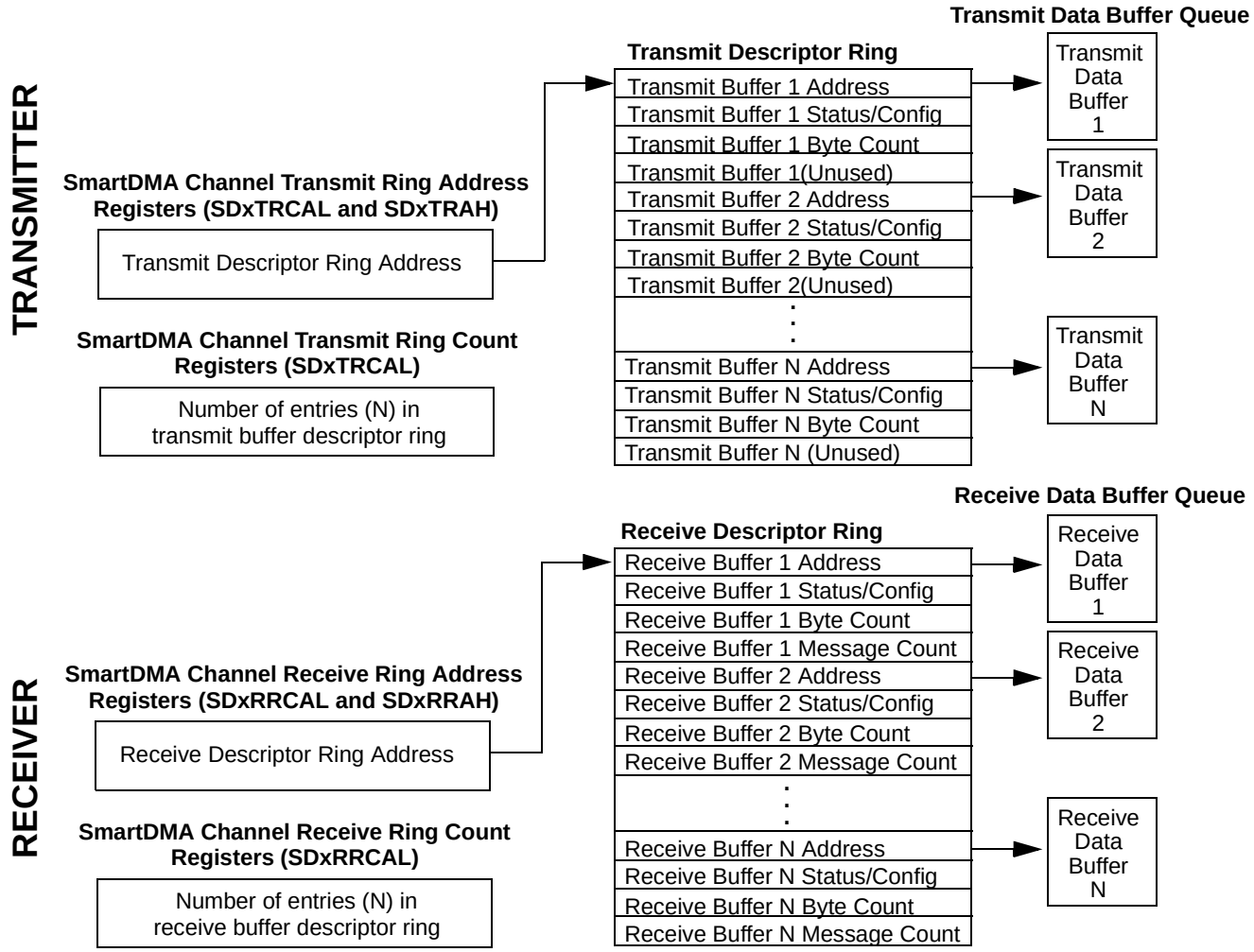


Figure 8-6 shows a descriptor ring with four entries. When OWN = 1, the SmartDMA channel owns the descriptor entry and can take data out or put data in the buffer. When OWN = 0, software owns the descriptor entry and the SmartDMA channel cannot access the buffer. This means in a transmit buffer, data is valid when OWN = 1; and in a receive buffer, data is valid when OWN = 0 (subject to any status error bit settings).

Descriptor ring entries are always accessed in order. In a transmit, the hardware always follows the software; in a receive, software follows the hardware. If a SmartDMA channel reaches a buffer whose OWN bit is not 1, the SmartDMA enters poll mode and waits for software to set the OWN bit. It does not advance past a buffer whose OWN bit is not set.

Keeping this in mind, the example in Figure 8-6 indicates the following for a transmit or a receive.

8.5.7.3.1 Transmit Descriptor Ring

If Figure 8-6 is a transmit descriptor ring, then software wrote the complete packet x (which spans buffers 1, 2, and 3 pointed to by descriptors 1, 2, and 3). After writing the complete packet, software set the OWN bit in each of the three descriptors. The OWN bits should be set in order, from the last descriptor in the packet to the first (3, then 2, then 1), to guarantee correct transmission of the packet. The channel has already transmitted the data from buffer 1 and is currently processing buffer 2. While packet x is being transmitted, software is writing data to buffer 4 for transmission of the next packet, packet y.

8.5.7.3.2 *Receive Descriptor Ring*

If Figure 8-6 is a receive descriptor ring, then the OWN bit in the current buffer descriptor was 1 when the receive channel began to receive packet x. The packet was not completely received before reaching the terminal count for buffer 1, so the channel cleared the owner semaphore for descriptor 1, releasing it for processing by software, and advanced to 2 for continued reception of the packet. Descriptor 3 has the OWN bit set, indicating that it is available for use by the receiver either for a continuation of packet x or for the next packet. Software is currently processing the data in buffer 4 and will set the OWN bit in the descriptor when it has completed.

8.5.7.4 SmartDMA Channel Usage

Note: *Before using the SmartDMA channels, ensure multiplexed pins are configured to reflect the use of DMA and not other functionality (see Table 8-1 on page 8-4).*

To use any of the eight SmartDMA channels (four pairs), the following must be programmed. Note that this must be done for both the transmitter and the receiver (because SmartDMA channels must be used in pairs) as well as for each channel pair used. The transmit and receive channels do not have to be enabled at the same time, but the SmartDMA channel should be initialized before the request source device is enabled. This allows the controller to fetch data about the initial receive and transmit buffers before receiving any DRQs. In addition, because the SmartDMA channel works off of requests from the device, it is always safe to enable the DMA before the device. Enabling the device before the DMA may result in data loss or an initial error condition being reported.

8.5.7.4.1 *Enabling the Transmit Channel*

To enable a SmartDMA transmit channel, software must perform the following tasks:

1. Create the transmit buffer descriptor ring.
2. Program the interrupt channel and configure the SmartDMA channel for interrupts.
3. Add data buffers to the ring.
4. Enable the transmit channel.

Create the Transmit Buffer Descriptor Ring

1. Disable the transmit channel by clearing the TXST bit in the SDxCON register to 0.
2. Allocate the memory for the transmit buffer descriptor ring (see “SmartDMA Channel Descriptor Format” on page 8-38 for the descriptor ring data structure).
3. Clear the OWN bit for each descriptor to 0 (owned by software).
4. Program the address and size of the transmit buffer descriptor ring into the SmartDMA channel registers.
   a. Program the TRA bits in the SDxTRCAL register to the 12 low-address bits (bits 15–4) of the descriptor ring address, which is the start location in memory of the buffer descriptor ring.
   b. Program the TRA bits in the SDxTRAH register to the four high-address bits (19–16) of the descriptor ring address. Because the base address of the ring must be paragraph aligned (aligned to a 16-byte physical memory boundary), address bits 3–0 are always zeroes.
   c. Program the TRC bits in the SDxTRCAL register to the number of entries in the transmit descriptor ring (the ring count). Valid values are 1, 2, 4, 8, 16, 32, 64, or 128 descriptors. For more information about 3-bit encoding, see the *Am186™CC/CH/CU*

Microcontrollers Register Set Manual, order #21916. Even when the ring size is set to 1, that entry is still interpreted as a descriptor, not as the memory buffer itself.

5. Point to the first buffer descriptor by clearing the SDxCBD register to 0.

Program the Interrupt Conditions

The interrupt conditions are typically configured only once.

1. Write the interrupt handler address to the vector table. See Chapter 7, “Interrupts.”
2. To generate an interrupt after transmitting the last byte of the packet, set the TEPI bit in the SDxCON register to 1.
3. To generate an interrupt after detecting an unavailable buffer during transmission, set the TBUI bit in the SDxCON register to 1.
4. To generate an interrupt after transmitting the last byte of the current buffer, set the TTCI bit in the SDxCON register to 1. (Note that the TTCE bit in Word 2 of the transmit buffer descriptor ring must also be set to 1.)
5. Program the priority of this channel relative to other channels during simultaneous transfers using the P bit in the SDxCON register (this is typically configured only once). A 00b is a low priority; a 01b, medium; and a 10b, high.

   Software clears the status bits in SDxSTAT after receiving an interrupt. Software can use the SDxCBD register to monitor the transmit and receive buffers. Software can also use the SDxCTAD register to determine the address in memory where the DMA transmit process was interrupted.

Add Data Buffers to the Transmit Descriptor Ring

To place a data buffer in an entry in the transmit buffer descriptor ring:

1. Find the first buffer descriptor for which the OWN bit is clear (bit = 0). This must be done in a circular manner relative to the current buffer descriptor index. In systems where the TXS0 or RXS0 bits are set, thereby inhibiting clearing of the OWN bits, software must determine when it is safe to modify a descriptor ring entry.
2. Program the data buffer address.
   a. Program the LADR bits in Word 0 to the low-order 16 address bits of the data buffer pointed to by the descriptor.
   b. Program the HADR bits in Word 1 to the high-order eight address bits of the data buffer pointed to by the descriptor. The highest four bits of the address must be set to 0000b. These address bits do not exist on the Am186CC/CH/CU microcontrollers’ 20-bit address but are provided for LANCE compatibility.
3. Program the data buffer size by setting the BCNT bits in Word 2 to the length in bytes of the data buffer pointed to by the descriptor.
4. Initialize the transmit buffer descriptor ring entries.
   a. Set to 1 the TTCE bit in Word 2 to enable interrupt on terminal count; or clear to 0 to disable terminal count interrupt. (Note that the TTCI bit in the SDxCON register must also be set to 1.)
   b. Set to 1 the STP bit in Word 1 to indicate that this is the first buffer of the packet, or clear to 0 if the buffer contains a continuation of a packet from another buffer.
   c. Set to 1 the ENP bit in Word 1 to indicate that this is the last buffer of the packet, or clear to 0 if the packet does not fit in one buffer and is continued in another.

d. Set to 1 the OWN bit in Word 1 to indicate the descriptor entry is owned by the SmartDMA channel.

e. To force a poll of the OWN bit of the current buffer descriptor, set to 1 the POLL bit in the SDxCON register. This has no effect if the SmartDMA is not currently waiting for a buffer to become available.

CC

5. In the Am186CC microcontroller, when using SmartDMA channel 2 or 3, select one of two alternate sources by clearing the DSEL bit in the SDxCON register to 0 to select HDLC or to 1 to select USB.

CH

In the Am186CH HDLC microcontroller, the DSEL bit in the SDxCON register must be cleared to 0.

CU

In the Am186CU USB microcontroller, the DSEL bit in the SDxCON register must be set to 1.

Enable the Transmit Channel

Enable the transmit channel by setting the TXST bit in the SDxCON register to 1. At this point, the SmartDMA transmit channel does not transmit any data because there are no valid buffers in the descriptor ring. As transmit data becomes available, software should modify entries in the ring to point to the data to be transmitted. Buffers are added to the ring at the first ring location following the Current Transmit Buffer Descriptor value that has an OWN bit set to 0.

8.5.7.4.2 Enabling the Receive Channel

To enable a SmartDMA receive channel, software must perform the following tasks:

1. Create the receive buffer descriptor ring.
2. Program the interrupt channel and configure the SmartDMA channel for interrupts.
3. Add data buffers to the ring.
4. Enable the receive channel.
5. Replace used data buffers.

Create the Receive Buffer Descriptor Ring

1. Disable the receive channel by clearing the RXST bit in the SDxCON register to 0.
2. Allocate the memory for the receive buffer descriptor ring (see “SmartDMA Channel Descriptor Format” on page 8-38 for the descriptor ring data structure).
3. Set the OWN bit for each descriptor to 1 (owned by hardware).
4. Program the address and size of the receive buffer descriptor ring into the SmartDMA channel registers.

   a. Program the RRA bits in the SDxRRCAL register to the 12 low address bits (bits 15–4) of the descriptor ring address, which is the start location in memory of the buffer descriptor ring.

   b. Program the RRA bits in the SDxRRAH register to the four high address bits (19–16) of the descriptor ring address. Because the base address of the ring must be paragraph aligned (aligned to a 16-byte physical memory boundary), address bits 3–0 are always zeroes.

   c. Program the RRC bits in the SDxRRCAL register to the number of entries in the receive descriptor ring (the ring count). Valid values are 1, 2, 4, 8, 16, 32, 64, or 128 descriptors. For information about 3-bit encoding, see the *Am186™CC/CH/CU Microcontrollers*

Register Set Manual, order #21916. Even when the ring size is set to 1, that entry is still interpreted as a descriptor, not as the memory buffer itself.

5. Point to the first buffer descriptor by clearing the SDxCBD register to 0.

Program the Interrupt Conditions

The interrupt conditions are typically configured only once.

1. To generate an interrupt after receiving the last byte of the packet, set the REPI bit in the SDxCON register to 1.
2. To generate an interrupt after detecting an unavailable buffer during reception, set the RBUI bit in the SDxCON register to 1.
3. To generate an interrupt after receiving the last byte of the current buffer, set the RTCI bit in the SDxCON register to 1. (Note that the RTCE bit in Word 2 of the transmit buffer descriptor ring must also be set to 1.)

Add Data Buffers to the Receive Descriptor Ring

To place a data buffer in an entry in the receive buffer descriptor ring:

1. Find the first buffer descriptor for which the OWN bit is clear (bit = 0). This must be done in a circular manner relative to the current buffer descriptor index. In systems where the TXS0 or RXS0 bits are set, thereby inhibiting clearing of the OWN bits, software must determine when it is safe to modify a descriptor ring entry.
2. Program the data buffer address.
   a. Program the LADR bits in Word 0 to the low-order 16 address bits of the data buffer pointed to by the descriptor.
   b. Program the HADR bits in Word 1 to the high-order eight address bits of the data buffer pointed to by the descriptor. The highest four bits of the address must be set to 0000b. These address bits do not exist on the Am186CC/CH/CU microcontrollers' 20-bit address but are provided for LANCE compatibility.
3. Program the data buffer size by setting the BCNT bits in Word 2 to the length in bytes of the data buffer pointed to by the descriptor.
4. Initialize the receive buffer descriptor ring entries.
   a. To enable interrupt on terminal count, set to 1 the RTCE bit in Word 2, or clear it to 0 to disable terminal count interrupt. (Note that the RTCI bit in the SDxCON register must also be set to 1.)
   b. Program the priority of this channel relative to other channels during simultaneous transfers using the P bit in the SDxCON register. A 00b is a low priority; a 01b, medium; and a 10b, high.
   c. Set to 1 the OWN bit in Word 1 to indicate the descriptor entry is owned by the SmartDMA channel.
   d. To force a poll of the OWN bit of the current buffer descriptor, set to 1 the POLL bit in the SDxCON register.

CC

5. In the Am186CC microcontroller, when using SmartDMA channel 2 or 3, select one of two alternate sources by clearing the DSEL bit in the SDxCON register to 0 to select HDLC or to 1 to select USB.

CH

In the Am186CH HDLC microcontroller, the DSEL bit in the SDxCON register must be cleared to 0.

CU

In the Am186CU USB microcontroller, the DSEL bit in the SDxCON register must be set to 1.

Software clears the status bits in SDxSTAT after receiving an interrupt. Software can use the SDxCBD register to monitor the transmit and receive buffers. Software can also use the SDxCRAD register to determine the address in memory where the DMA receive process was interrupted.

Enable the Receive Channel

Enable the receive channel by setting the RXST bit in the SDxCON register to 1.

At this point, the SmartDMA receive channel is enabled. As received data is processed, software should modify entries in the ring to point to empty data buffers. Buffers are added to the ring at the first ring location following the Current Buffer Descriptor value that has an OWN bit cleared to 0.

Replace Used Data Buffers

1. Program the new data buffer address.
   a. Program the LADR bits in Word 0 to the low-order 16 address bits of the data buffer pointed to by the descriptor.
   b. Program the HADR bits in Word 1 to the high-order eight address bits of the data buffer pointed to by the descriptor. The highest four bits of the address must be set to 0000b. These address bits do not exist on the Am186CC/CH/CU microcontrollers' 20-bit address but are provided for LANCE compatibility.
2. Set to 1 the OWN bit in Word 1 to indicate the descriptor entry is owned by the SmartDMA channel.
3. To force a poll of the OWN bit of the current buffer descriptor, set to 1 the POLL bit in the SDxCON register. This bit has no effect if the SmartDMA channel is not currently waiting for a buffer to become available.

8.5.7.4.3 Enable the Peripheral Device

CC CH

Details for configuring and enabling the HDLC peripheral device being used can be found in Chapter 15, "High-Level Data Link Control (HDLC)."

CC CU

Details for configuring and enabling the USB peripheral device being used can be found in Chapter 18, "Universal Serial Bus (USB)."

The DMA should always be enabled before the requesting device is enabled. The DMA should always be disabled after the requesting device is disabled.

8.5.7.5 SmartDMA Channel Cycle

This section and the following sections describe the procedure the SmartDMA controller follows for both a transmit and a receive.

8.5.7.5.1 SmartDMA Transmit Channel Cycle

The flow diagram for the SmartDMA transmit channel is shown graphically in Figure 8-8 on page 8-37 and discussed below.

1. When the transmit channel is first enabled, the SmartDMA controller enters initialization mode.
2. The transmit channel reads the current descriptor and checks to see if the owner semaphore (OWN) bit is set to 1.

If the OWN bit is 0, the software owns the current descriptor. In this case, the SmartDMA transmit channel periodically polls the descriptor until the OWN bit becomes 1. The transmit channel does not advance past a descriptor for which the OWN bit is 0. For information about forcing a poll, see “SmartDMA Channel Descriptor Polling” on page 8-41.

3. If the OWN bit is 1 in the current descriptor, the transmit channel checks to see if the start-of-packet bit (STP) bit is set to 1.

   If the STP bit is 0, the transmit channel enters Search-For-Start-of-Packet mode. This mode simply clears the OWN bit in the current descriptor and advances to the next descriptor ring entry. The transmit channel then returns to Initialization mode, repeating these steps until it finds an entry with both the OWN and STP bits set to 1.

4. If the OWN and STP bits are both set to 1, the transmit channel reads the length of the buffer from the descriptor ring (BCNT bits in Word 2) and programs that value into an internal terminal count register. The address of the buffer associated with this descriptor is read from the descriptor (LADR and HADR bits) into the SDxCTAD source address register. The transmit channel then enters normal-transmit mode.

5. In transmit mode, the channel transmits one byte of data from the memory buffer to the destination device for every DRQ. After each transfer, the source address in SDxCTAD is incremented and the internal transfer count is decremented.

6. When the internal terminal count is reached, the transmit channel checks the end-of-packet (ENP) bit.

   a. If the ENP bit is 0 in the current descriptor, the transmit channel attempts to acquire the next buffer. The transmit channel releases the current buffer by clearing the OWN bit (unless the TXS0 bit is set). It then advances to the next descriptor in the ring. If the OWN bit is 0 (the software owns the descriptor), the transmit channel periodically polls the descriptor until OWN becomes 1. If an error condition occurs (e.g., a FIFO underflow) before the transmit channel acquires the next descriptor, the error causes the requesting transmit source to shut down and the SmartDMA channel to be reprogrammed. If the transmit channel successfully acquires the next descriptor, the new buffer address and terminal count are loaded into the appropriate internal registers.

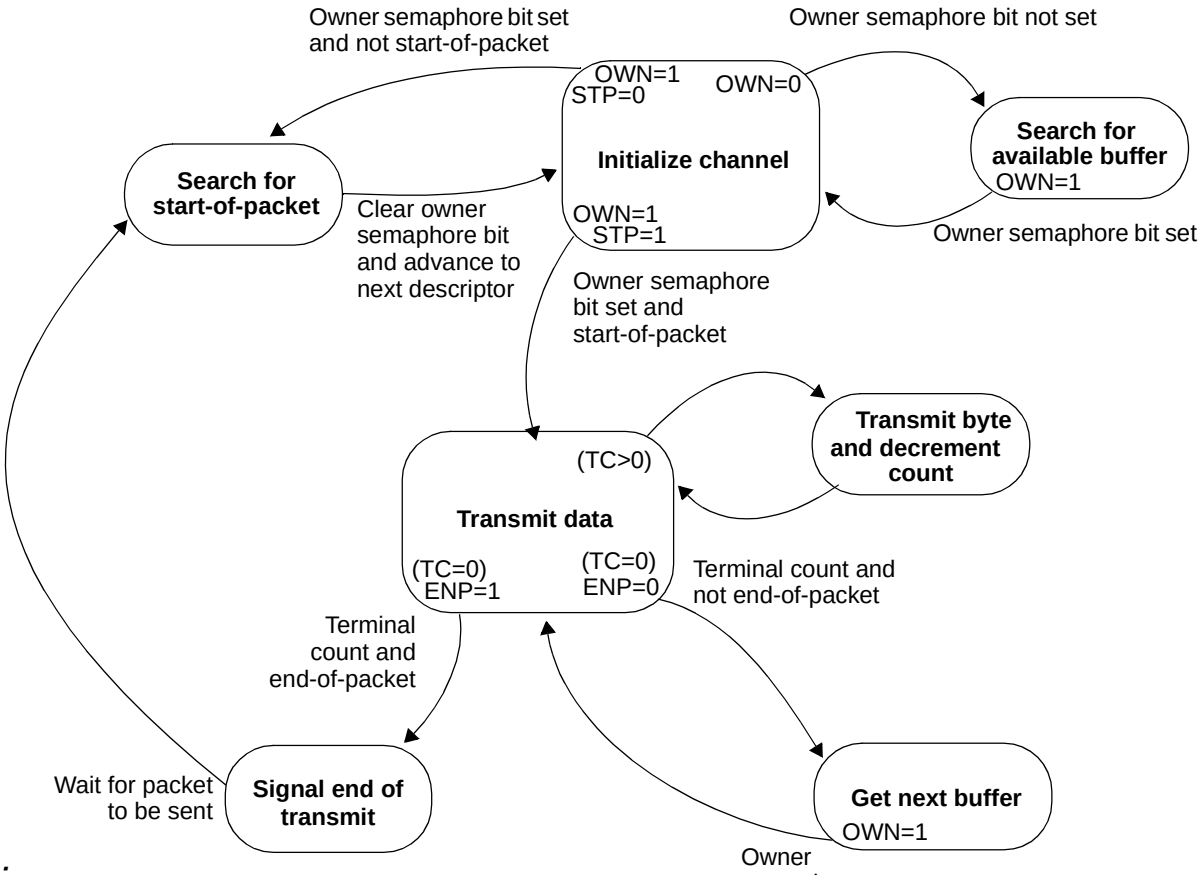
   b. When the terminal count is reached for a buffer for which the ENP bit is set, the transmit channel enters Transmit-End mode. In this mode, the transmit channel signals the end-of-packet to the device by asserting an internal signal during the transfer of the last data byte. The SmartDMA transmit channel waits for the packet to be sent successfully, then advances the index to the next buffer.

      If a complete packet is transmitted, the channel releases the current buffer by clearing the OWN bit before attempting to advance to the next buffer. If a packet is incomplete when the channel has reached terminal count on the buffer, it releases control of the buffer and advances to the next buffer in the ring. If the TXS0 bit is set, the channel moves to the next buffer without clearing the OWN bit.

      Whenever a packet needs to be retransmitted, the transmit channel must be disabled and the Current Buffer Descriptor (SDxCBD) register must be programmed with the index of the buffer descriptor containing the STP bit for that packet. The transmit channel does not report any status in the buffer descriptor other than clearing the OWN bit.

Note: *Before disabling the transmit channel, you should stop the HDLC channel.*

Figure 8-8 SmartDMA Transmit Channel Flow Diagram



Notes:

The TC bit is internal and not seen by users.

8.5.7.5.2 SmartDMA Receive Channel Cycle

The flow diagram for the SmartDMA receive channel is shown graphically in Figure 8-9 on page 8-38 and discussed below.

1. When the receive channel is first enabled, the SmartDMA controller enters Initialization mode.
2. The receive channel fetches the data for the first descriptor in the receive descriptor ring and checks to see if the owner semaphore (OWN) bit is set to 1.

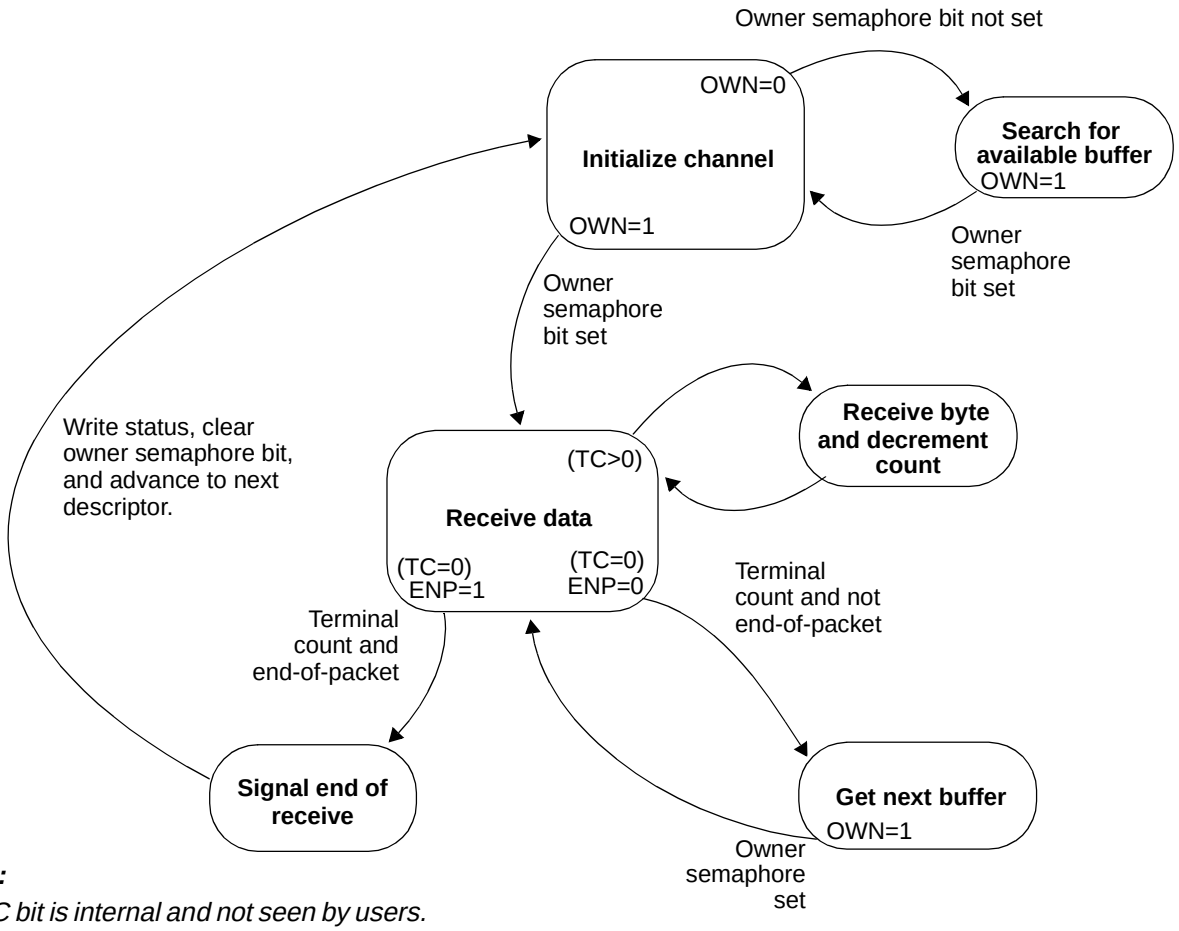
   If the OWN bit is 0, the software owns the current descriptor. In this case, the SmartDMA receive channel periodically polls the descriptor until the OWN bit becomes 1. The receive channel does not advance past a descriptor for which the OWN bit is 0. For information about forcing a poll, see “SmartDMA Channel Descriptor Polling” on page 8-41.
3. If the OWN bit is 1 in the current descriptor, the SmartDMA controller loads the address of the buffer into an internal receive address register. The length of the buffer is also read from the descriptor and programmed into an internal terminal count register. The receive channel then enters Normal-Receive mode.
4. In Receive mode, the terminal count is decremented and the destination address is incremented for each byte transferred. The receiver remains in Normal-Receive mode until an end-of-packet is detected or the terminal count is reached.

5. If the terminal count reaches zero before the end-of-packet signal from the device is asserted, the receiver closes the current buffer and enters Get-Next-Buffer mode. In this mode, the receiver reads the next descriptor in the descriptor ring and determines if the OWN bit is set. If the OWN bit is 0, the receiver remains in Get-Next-Buffer mode, periodically polling the descriptor, until the OWN bit becomes set. When the OWN bit is detected as set, the receiver loads the buffer address and terminal count from the new descriptor and returns to normal-receive mode.
6. When the receiver detects the end-of-packet signal from the device, the receiver moves to Receive-End mode. In Receive-End mode, the receiver reads the status information from the device and writes it to the descriptor. The end-of-packet bit is set in the descriptor and the OWN bit is cleared. If RXS0 is set, the EOP bit is set but the OWN bit not cleared.
7. The receiver advances the descriptor ring pointer and enters Initialize mode.

8.5.7.6 SmartDMA Channel Descriptor Format

Each entry in the descriptor ring consists of four 16-bit words. Table 8-13 shows the format of the transmit descriptor ring; Table 8-14, the receive descriptor ring.

Figure 8-9 SmartDMA Receive Channel Flow Diagram



Notes:

The TC bit is internal and not seen by users.

Table 8-13 SmartDMA Transmit Channel Descriptor Format

Bit Number	Bit Name	Description
Transmit Buffer Address (Word 0)		
15–0	LADR[1]	The LADR (Low Order) field contains the 16 low-order address bits of the data buffer pointed to by this descriptor. The LADR field is written by the software and not changed by the SmartDMA channel.
Transmit Buffer Status/Config (Word 1)		
15	OWN	0 = Descriptor entry is owned by the software. 1 = Descriptor entry is owned by the SmartDMA channel. The software sets the OWN bit after filling the buffer pointed to by this descriptor. The SmartDMA channel clears the OWN bit (unless the TXS0 bit is set) after transmitting the contents of the buffer. Neither the software nor the SmartDMA channel can alter a descriptor entry after it has relinquished ownership.
14–10	Reserved	
9	STP	The STP (Start of Packet) bit indicates that this is the first buffer to be used by the SmartDMA channel for this packet. It is used to chain data buffers. The STP bit is set by the software and is not changed by the SmartDMA channel. The STP bit must be set in the first buffer of the packet, or the SmartDMA channel skips over this descriptor and polls the next descriptor(s) until both the OWN and STP bits are set.
8	ENP	The ENP (End of Packet) bit indicates that this is the last buffer used by the SmartDMA channel for this packet. It is used to chain data buffers. If both the STP and ENP bits are set, the packet fits into one buffer and there is no data chaining. The ENP bit is set by the software and is not changed by the SmartDMA channel.
7–0	HADR[1]	The HADR (High Order) field contains the eight high-order address bits of the data buffer pointed to by this descriptor. The highest four bits of the address must be set to 0000b. These address bits do not exist in the microcontroller's 20-bit address but are provided for LANCE compatibility. The HADR field is written by the software and not changed by the SmartDMA channel.
Transmit Buffer Byte Count (Word 2)		
15	TTCE	0 = Disable TTC interrupt. 1 = Enable TTC interrupt. This bit is used to enable the Transmit Terminal Count interrupt.
14–0	BCNT	The BCNT (Buffer Byte Count) field contains the length in bytes of the buffer pointed to by this descriptor. This number is expressed in 2's complement format and indicates the number of bytes from this buffer that are transmitted by the SmartDMA channel. This field is written by the software and not changed by the SmartDMA channel. For example, if you want to transfer 64 bytes, take the number 64 (40h), complement it (7FBFh), and increment it by 1 (7FC0h). Place this number (7FC0h) in the BCNT field.
Transmit Buffer Word 3: This word is used for receive channels only; the transmit channels do not write any status to this word.		

Notes:

1. The address programmed in the LADR and HADR fields is a linear address, not a segment:offset address. For example, if a transmit data buffer starts at segment address C000h and offset 1000h, it would be programmed as follows:

Linear address= (segment address x 16) + offset address

= (C000h x 16) + 1000h = C1000h

LADR = 1000h

HADR = 0Ch

Table 8-14 SmartDMA Receive Channel Descriptor Format

Bit Number	Bit Name	Description
Receive Buffer Address (Word 0)		
15–0	LADR[1]	The LADR (Low Order) field contains the 16 low order address bits of the data buffer pointed to by this descriptor. The LADR field is written by the software and not changed by the SmartDMA channel.
Receive Buffer Status/Config (Word 1)		
15	OWN	0 = Descriptor entry is owned by the software. 1 = Descriptor entry is owned by the SmartDMA channel. The SmartDMA channel clears the OWN bit (unless the RXS0 bit is set) after filling the buffer pointed to by this descriptor. The software sets the OWN bit after emptying the buffer. Neither the software nor the SmartDMA channel can alter a descriptor entry after it has relinquished ownership.
14	ERR	The ERR (Error Summary) bit is the logical OR of FRAM, OFLO, CRC and HBUF.
13	FRAM	The FRAM (Framing Error) bit indicates that the received frame did not contain a multiple of eight bits. The CRC bit is not checked when the FRAM bit is set. The FRAM bit is valid only when the ENP bit is set and the OFLO bit is not. This bit is not used when the USB is the receive request. This bit is cleared by software.
12	OFLO	The OFLO (Overflow Error) bit indicates that the internal receive FIFO has detected an overflow condition. The OFLO bit is valid only when the ENP bit is not set. This bit is cleared by software.

11	CRC	The CRC (Cyclic Redundancy Check Error) bit indicates:
		• When HDLC is the requesting source, the current frame has a CRC error.
		• When USB is the requesting source, one of the following errors occurred:
		  • If the USB endpoint type is BULK, the possible errors are: CRC, bit stuff, more than max packet value sent by software, data PID error, or data toggle error.
		  • If USB endpoint type is ISO, the possible errors are: CRC, bit stuff, more than max packet value sent by software, or data PID error.
		The CRC bit is valid only when the ENP bit is set and the OFLO bit is not. This bit is cleared by software.

10	HBUF	The HBUF (Buffer Error) bit indicates that the current frame has one of the following errors: • Frame ended in an abort instead of a flag. • Frame length was longer than the maximum length allowed. In this case, the MCNT field is equal to the maximum length allowed. • Frame length was shorter than the minimum allowed. Part of the data of the frame was already discarded in the receiver. The MCNT field indicates the number of bytes that were output by the FIFO, not the number of bytes in the frame. This bit is not used when the USB is the receive request. This bit is cleared by the software.
9	STP	The STP (Start of Packet) bit indicates that this is the first buffer used by the SmartDMA channel for this packet. It is used to chain data buffers. The STP bit is set by the SmartDMA channel.
8	ENP	The ENP (End of Packet) bit indicates that this is the last buffer used by the SmartDMA channel for this packet. It is used to chain data buffers. If both the STP bit and the ENP bit are set, the packet fits into one buffer and there is no data chaining. The ENP bit is set by the SmartDMA channel.

Table 8-14 SmartDMA Receive Channel Descriptor Format (Continued)

Bit Number	Bit Name	Description
7–0	HADR[1]	The HADR (High Order) field contains the eight high-order address bits of the data buffer pointed to by this descriptor. The highest four bits of the address must be set to 0000b. These address bits do not exist in the microcontroller's 20-bit address space but are provided for LANCE compatibility. The HADR field is written by the software and not changed by the SmartDMA channel.
Receive Buffer Byte Count (Word 2)		
15	RTCE	0 = Disable RTC interrupt. 1 = Enable RTC interrupt. This bit is used to enable the Receive Terminal Count interrupt.
14–0	BCNT	The BCNT (Buffer Byte Count) field contains the length in bytes of the buffer pointed to by this descriptor. This number is expressed in 2's complement format and indicates the number of bytes allocated for this buffer. This field is written by the software and not changed by the SmartDMA channel. For example, if you want to transfer 64 bytes, take the number 64 (40h), complement it (7FBFh), and increment it by 1 (7FC0h). Place this number (7FC0h) in the BCNT field.
Receive Buffer Message Count (Word 3)		
15	Reserved	Read/Write as zero.
14–0	MCNT	The MCNT (Message Byte Count) field contains the length in bytes of the frame. The MCNT field is valid only when the ERR bit is 0 and the ENP bit is 1. This field is written by the SmartDMA channel and cleared by software.

Notes:

1. The address programmed in the LADR and HADR fields is a linear address, not a segment:offset address. For example, if a receive data buffer starts at segment address C000h and offset 1000h, it would be programmed as follows:

Linear address= (segment address x 16) + offset address

= (C000h x 16) + 1000h = C1000h

LADR = 1000h

HADR = 0Ch

8.5.7.7 SmartDMA Channel Descriptor Polling

When any of the SmartDMA channels on the Am186CC/CH/CU microcontrollers require a new buffer and the owner semaphore (OWN) bit for the next descriptor in the descriptor ring is not set, the channel does a periodic poll (read) of the descriptor to determine if software has set the OWN bit in the intervening time. To assure that this polling has a minimal effect on interrupt latency and system performance, the DMA uses a single counter to trigger the poll. Each channel is given a unique timer value that is used to initiate the poll; these values are evenly dispersed throughout the timer period. This behavior guarantees that only a single SmartDMA channel attempts to poll its descriptor ring at any given time. The automatic poll timer completes one cycle every 64K processor clocks. This results in a potential poll cycle every 8K clocks.

The SmartDMA Channel Control (SDxCON) register provides bits that allow software to request an immediate poll of one or both of the current descriptors (transmit channel and/or receive channel). This poll does not affect the polling status of any other channel.

A poll is never performed if the SmartDMA channel does not currently need the next buffer. This is true even if software sets the bit requesting an immediate poll.

8.5.7.8 SmartDMA Channel Interrupts

SmartDMA channels can generate interrupts based on three conditions. The interrupt remains pending until software clears the associated status bit(s).

The following list shows the different interrupt types, and gives some useful information about the characteristics of the interrupt.

- TEPI and REPI (Transmit/Receive End of Packet) interrupts are asserted when the last byte of a packet is transmitted or received. If many of the packets are short or appear to be short (e.g., due to an excessively noisy communications line), then the frequency of the interrupts may be higher than desired.
- TTCI and RTCI (Transmit/Receive Terminal Count) interrupts are asserted when the last byte of a buffer is transmitted or received. The TTCE/RTCE bit in word 2 of the descriptor entry must also be set to use this interrupt.

  Because generation of this interrupt is controllable on a buffer-by-buffer basis, it can be set up so that an interrupt occurs every *n* buffers, where *n* is completely under software control. This can be a useful way to reduce frequency of interrupts while ensuring that there are always descriptors available for the hardware. However, relying solely on this interrupt for a receive descriptor chain could mean that up to the last *n–1* buffers in the ring go unprocessed.

  If RTCI is the primary interrupt for a receive descriptor ring, a timer interrupt can also be used to detect leftover buffers after the other side has stopped sending. This can be a one-shot timer that is reset on every RTC interrupt, so that the timer expires shortly after the next RTC interrupt is expected. This timer interrupt generally should not be required on transmit descriptor rings, because any system that wants to be interrupted after there are no more buffers in the ring to send out can rely on the TBUI interrupt (see the description below).
- TBUI and RBUI (Transmit/Receive Buffer Unavailable) interrupts are asserted when an attempt to load the next descriptor in the ring finds that the OWN bit is 0.

  As discussed above, TBUI can be used to determine if all buffers have been sent. You should not rely on RBUI interrupts to cause software to make more buffers available to the hardware, or on TBUI interrupts to cause software to make additional buffers within a single packet available to the hardware. The interrupt latency associated with either of these tasks could cause FIFO overflows or underruns. RBUI can be used as a type of watchdog interrupt—if RBUI interrupts are occurring, it means that, for some reason, the system is not giving the receiver buffers fast enough.

8.5.7.9 SmartDMA Channel Use Without CPU Intervention

In each SmartDMA channel, the user has the option of not clearing the OWN bit after DMA has finished accessing a buffer (with the TXS0 and RXS0 bits in the SDxCON register). This option provides the effect of a circular buffer from which data is accessed without the intervention of software.

In a typical scenario, an HDLC receive channel and a different HDLC transmit channel, or an HDLC receive channel and a USB IN endpoint, share a circular buffer. This shared buffer is implemented in software by making the transmit DMA channel read from the same memory area to which the receive DMA channel is writing. A software PLL attempts to keep the write pointer slightly ahead of the read pointer so that stale data is never read by the transmitter, and so that the receiver never overwrites data not yet transmitted. In case of an error, both the transmit and receive channel can be disabled and reprogrammed to start at any particular buffer descriptor in the ring.

The following facilities aid in SmartDMA channel circular buffer management:

- When receiving transparent HDLC data, no buffer status is transferred to the SmartDMA channel. The received data is simply a continuous stream of samples, and the SmartDMA controller keeps cycling through buffers without ever storing an EOP.
- The TXSO and RXSO bits in the SDxCON registers can be set to keep the DMA controller from returning buffer descriptors to the software. If the OWN bit is never updated, the DMA controller can run through the descriptor ring multiple times without software intervention.
- The SDxCTAD and SDxCRAD registers can be read to determine the current buffer position, to enable a software PLL to be able to control the rate of buffer filling/emptying.
- If an error occurs, the SDxCBD registers can restart the DMA at any arbitrary point in the buffer.

In the simplest instance, a circular buffer can be formed by using a ring with a single descriptor. The descriptor contains the starting address and length of the circular buffer, and the STP and OWN bits must be set so that the DMA controller uses the buffer.

If code is to allow for adjustment of the buffer pointer (e.g., in case a USB isochronous transfer has an error or is missing), then the ring should have two descriptors in it. Each descriptor points to a portion of the physical buffer, and the DMA can be started at any arbitrary point by adjusting the descriptors' starting addresses and lengths, and setting the SDxCBD registers to point to the correct descriptor.

8.5.8 DMA and USB

CC CU

The integrated USB peripheral controller is the only Am186CC and Am186CU microcontroller peripheral capable of using either general-purpose DMA or a SmartDMA channel. Each of the four USB data endpoints is connected to a single SmartDMA channel, and can be connected to any of the general-purpose DMA channels. DMA is a powerful tool when used with the USB peripheral controller. In addition to providing increased throughput and responsiveness to USB requests, it allows the use of larger packets, and it enables the USB peripheral controller's automatic rate control feature for isochronous transfers based on the PCM highway frame clock or an external frame clock source.

For more information about using DMA with the USB peripheral controller, see Chapter 18, "Universal Serial Bus (USB)."

8.5.9 Software-Related Considerations

Software must stop DMA operation before writing to the GDxCON1 register, or the results are unpredictable. Stopping the SmartDMA channel has no effect while a request is pending on the channel. Before stopping the channel, make sure the requesting peripheral (HDLC channel or USB endpoint) is stopped.

8.5.10 Comparison to Other Devices

- The general-purpose DMA channels are the same as on other Am186 controllers.
- SmartDMA channels are compatible with the Am79C90 C-LANCE DMA.

8.6 INITIALIZATION

On both an internal and external reset, the following occurs:

- All the general-purpose DMA and SmartDMA channel registers are cleared to 0.
- Any DMA transfer in progress is aborted.
- Multiplexed signal DRQ0 defaults to its PIO functionality.

CHAPTER

9 PROGRAMMABLE I/O SIGNALS

9.1 OVERVIEW

The Am186CC/CH/CU microcontrollers provide 48 user-programmable input/output signals (PIOs).

Many of these signals share a pin with at least one alternate function. If an application does not need the alternate function, the associated PIO can be programmed through the PIO registers.

If a pin is enabled to function as a PIO, the alternate function is disabled and does not affect the pin. Conversely, the value of any pin configured as a PIO does not affect the alternate function of the pin. When configured as a PIO, an appropriate default value for the signal is sent to the associated device rather than the value on the pin.

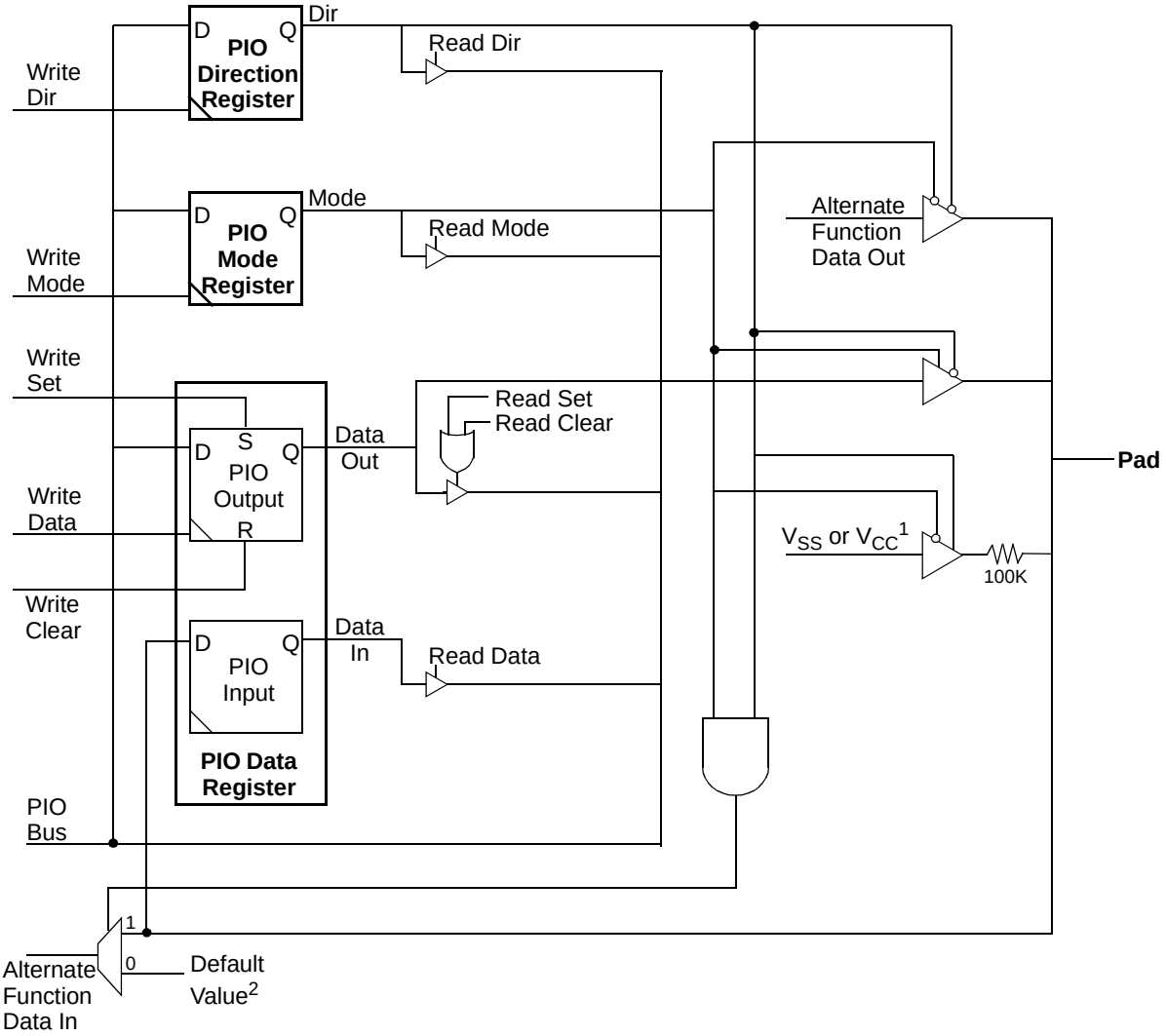
A PIO can be configured to operate as an input or output, with or without internal pullup or pulldown resistors (pullup or pulldown depends on the pin configuration and is not user-configurable), or as an open-drain output. Additionally, eight PIOs can be configured as external interrupt sources. For information on this interrupt functionality, see Chapter 7, “Interrupts.”

Associated bits in the PIO Mode, PIO Direction, PIO Data, PIO Set, and PIO Clear registers control each of the 48 PIOs. Because these registers are 16 bits wide, each PIO function requires three registers (see Table 9-2). Two additional registers are provided for ease of use.

9.2 BLOCK DIAGRAM

Figure 9-1 shows the PIO operation.

Figure 9-1 PIO Operation Block Diagram



Notes:

1. *Depends on pullup or pulldown.*
2. *When the PIO is enabled, an appropriate default value is driven on the Alternate Function Data In.*

9.3 SYSTEM DESIGN

Table 9-1 lists the PIO signals that are multiplexed with other microcontroller functions. Pinstraps are sampled only at external reset and do not affect the pin's other functions, so they are not shown in this table. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin. The table also shows which register bit programs the pin to be the PIO or alternate function.

Table 9-1 PIO Multiplexed Signals

Signal	Multiplexed Signal(s)	Default Signal	Register[Bit]
PIO0	TMRIN1	PIO0: input with pullup	PIOMODE0[0]
PIO1	TMROUT1	PIO1: input with pulldown	PIOMODE0[1]
PIO2	$\overline{\text{PCS5}}$	PIO2: input with pullup	PIOMODE0[2]
PIO3	$\overline{\text{PCS4}}$	PIO3: input with pullup	PIOMODE0[3]
PIO4	$\overline{\text{MCS0}}$	PIO4: input with pullup	PIOMODE0[4]
PIO5	$\overline{\text{MCS3}}$ / $\overline{\text{RAS1}}$	PIO5: input with pullup	PIOMODE0[5]
PIO6	INT8 / PWD	PIO6: input with pullup	PIOMODE0[6]
PIO7	INT7	PIO7: input with pullup	PIOMODE0[7]
PIO8	ARDY	ARDY: input with pullup	PIOMODE0[8]
PIO9	DRQ0	PIO9: input with pulldown	PIOMODE0[9]
PIO10	SDEN	PIO10: input with pulldown	PIOMODE0[10]
PIO11	SCLK	PIO11: input with pullup	PIOMODE0[11]
PIO12	SDATA	PIO12: input with pullup	PIOMODE0[12]
PIO13	$\overline{\text{PCS0}}$	$\overline{\text{PCS0}}$: input with pullup	PIOMODE0[13]
PIO14	$\overline{\text{PCS1}}$	$\overline{\text{PCS1}}$: input with pullup	PIOMODE0[14]
PIO15	$\overline{\text{WR}}$	$\overline{\text{WR}}$: input with pullup	PIOMODE0[15]
PIO16	RXD_HU	PIO16: input with pullup	PIOMODE1[0]
PIO17	$\overline{\text{DCE\_CTS\_A}}$ CC CH $\overline{\text{PCM\_TSC\_A}}$ CC CH	PIO17: input with pullup	PIOMODE1[1]
PIO18	$\overline{\text{DCE\_RTR\_A}}$ CC CH	PIO18: input with pullup	PIOMODE1[2]
PIO19	INT6	PIO19: input with pullup	PIOMODE1[3]
PIO20	TXD_U/ DCE_TXD_D CC / PCM_TXD_D CC	PIO20: input with pullup	PIOMODE1[4]
PIO21	UCLK USBSOF CC CU USBSCI CC CU	PIO21: input with pullup	PIOMODE1[5]
PIO22	DCE_RCLK_C CC PCM_CLK_C CC	PIO22: input with pulldown	PIOMODE1[6]
PIO23	DCE_TCLK_C CC PCM_FSC_C CC	PIO23: input with pulldown	PIOMODE1[7]
PIO24	$\overline{\text{CTS\_U}}$ DCE_TCLK_D CC PCM_FSC_D CC	PIO24: input with pullup	PIOMODE1[8]
PIO25	$\overline{\text{RTR\_U}}$ DCE_RCLK_D CC PCM_CLK_D CC	PIO25: input with pullup	PIOMODE1[9]
PIO26	RXD_U DCE_RXD_D CC PCM_RXD_D CC	PIO26: input with pullup	PIOMODE1[10]
PIO27	TMRIN0	PIO27: input with pullup	PIOMODE1[11]
PIO28	TMROUT0	PIO28: input with pulldown	PIOMODE1[12]

Table 9-1 PIO Multiplexed Signals (Continued)

Signal	Multiplexed Signal(s)	Default Signal	Register[Bit]
PIO29	DT/$\overline{R}$	DT/$\overline{R}$: three-state output with pullup	PIOMODE1[13]
PIO30	$\overline{DEN}$ / $\overline{DS}$	$\overline{DEN}$: three-state output with pullup	PIOMODE1[14]
PIO31	$\overline{PCS7}$	PIO31: input with pullup	PIOMODE1[15]
PIO32	$\overline{PCS6}$	PIO32: input with pullup	PIOMODE2[0]
PIO33	ALE	ALE: three-state output with pulldown	PIOMODE2[1]
PIO34	$\overline{BHE}$	$\overline{BHE}$: input with pullup	PIOMODE2[2]
PIO35	SRDY	SRDY: input with pullup	PIOMODE2[3]
PIO36	DCE_RXD_B CC CH PCM_RXD_B CC CH	PIO36: input with pullup	PIOMODE2[4]
PIO37	DCE_TXD_B CC CH PCM_TXD_B CC CH	PIO37: input with pullup	PIOMODE2[5]
PIO38	$\overline{DCE\_CTS\_B}$ CC CH $\overline{PCM\_TSC\_B}$ CC CH	PIO38: input with pullup	PIOMODE2[6]
PIO39	$\overline{DCE\_RTR\_B}$ CC CH	PIO39: input with pullup	PIOMODE2[7]
PIO40	DCE_RCLK_B CC CH PCM_CLK_B CC CH	PIO40: input with pullup	PIOMODE2[8]
PIO41	DCE_TCLK_B CC CH PCM_FSC_B CC CH	PIO41: input with pullup	PIOMODE2[9]
PIO42	DCE_RXD_C CC PCM_RXD_C CC	PIO42: input with pulldown	PIOMODE2[10]
PIO43	DCE_TXD_C CC PCM_TXD_C CC	PIO43: input with pulldown	PIOMODE2[11]
PIO44	$\overline{DCE\_CTS\_C}$ CC $\overline{PCM\_TSC\_C}$ CC	PIO44: input with pullup	PIOMODE2[12]
PIO45	$\overline{DCE\_RTR\_C}$ CC	PIO45: input with pullup	PIOMODE2[13]
PIO46	$\overline{CTS\_HU}$ $\overline{DCE\_CTS\_D}$ CC $\overline{PCM\_TSC\_D}$ CC	PIO46: input with pullup	PIOMODE2[14]
PIO47	$\overline{RTR\_HU}$ $\overline{DCE\_RTR\_D}$ CC	PIO47: input with pullup	PIOMODE2[15]

9.4 REGISTERS

The 16 registers listed in Table 9-2 program the PIO signals. Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 9-2 PIO Register Summary

Offset	Register Mnemonic	Register Name	Description
3C0h	PIOMODE0	PIO Mode 0	Set PIO15–PIO0 to PIO or alternate function, and as input or output (see Table 9-3).
3C2h	PIODIR0	PIO Direction 0	
3C4h	PIODATA0	PIO Data 0	Stores read or write data driven on outputs PIO15–PIO0. Reads of this register reflect the value of the pin.
3C6h	PIOSET0	PIO Set 0	Sets PIO Data register contents for PIO15–PIO0.
3C8h	PIOCLR0	PIO Clear 0	Clears PIO Data register contents for PIO15–PIO0.
3CAh	PIOMODE1	PIO Mode 1	Set PIO31–PIO16 to PIO or alternate function, and as input or output (see Table 9-3).
3CCh	PIODIR1	PIO Direction 1	
3CEh	PIODATA1	PIO Data 1	Stores read or write data driven on outputs PIO31–PIO16. Reads of this register reflect the value of the pin.
3D0h	PIOSET1	PIO Set 1	Sets PIO Data register contents for PIO31–PIO16.
3D2h	PIOCLR1	PIO Clear 1	Clears PIO Data register contents for PIO31–PIO16.
3D4h	PIOMODE2	PIO Mode 2	Set PIO47–PIO32 to PIO or alternate function, and as input or output (see Table 9-3).
3D6h	PIODIR2	PIO Direction 2	
3D8h	PIODATA2	PIO Data 2	Stores read or write data driven on outputs PIO47–PIO32. Reads of this register reflect the value of the pin.
3DAh	PIOSET2	PIO Set 2	Sets PIO Data register contents for PIO47–PIO32.
3DCh	PIOCLR2	PIO Clear 2	Clears PIO Data register contents for PIO47–PIO32.

9.5 OPERATION

9.5.1 Usage

Note: *Before using the PIOs, ensure multiplexed pins are configured to reflect the use of PIO and not other functionality (see Table 9-1 on page 9-3).*

To define a pin to be used as a PIO, use the following process:

1. Set the applicable bits in the PIO Mode and PIO Direction registers. To avoid changing system PIO functionality unintentionally, it is good programming practice to do a read-modify-write when setting these bits.
2. Manipulate data with the PIO Data, PIO Set, and PIO Clear registers.

9.5.2 Defining the PIO Signal as Input or Output

Table 9-3 shows how the bit settings for the PIO Mode and PIO Direction registers affect signal function. The internal pullup and pulldown resistors each have a value of approximately 10 KΩ.

Table 9-3 PIO Mode and PIO Direction Register Bit Settings

Mode	PIO Mode Register	PIO Direction Register	Pin Function
Alternate Operation	0	0	Alternate operation with pullup/pulldown (PIO functionality disabled)
PIO	0	1	PIO input with pullup/pulldown[1]
	1	0	PIO output with pullup/pulldown[1]
	1	1	PIO input without pullup/pulldown[1]

Notes:
1. The following PIO signals can be configured as interrupt sources in the interrupt controller's Shared Mask (SHMASK) register: PIO5, PIO15, PIO27, PIO29, PIO30, PIO33, PIO34, and PIO35. Typically, these signals should be configured as inputs when used as an interrupt source. However, if any of these signals is configured as both a PIO output and as an interrupt source, the PIO output signal generates interrupts.

9.5.3 Driving Data on the PIO

If a PIO signal is enabled as an output, the value in the corresponding bit in the PIO Data register is driven on the signal with no inversion.

Whether a PIO signal is enabled as an input or as an output, a synchronized value from the PIO signal is reflected in the value of the corresponding bit in the PIO Data register, with no inversion for PIO Data register reads.

9.5.4 Using PIOs as Open-Drain Outputs

The PIO Data registers permit the PIO signals to operate as open-drain outputs. This is accomplished by keeping the appropriate PDATA bits constant in the PIO Data and PIO Mode registers and writing the data value into its associated bit position in the PIO Direction register. The output is either driving Low or is disabled, depending on the data.

9.5.5 Setting and Clearing Data

The Am186CC/CH/CU microcontrollers offer two additional registers, which can be used to set and clear the PIO Data register. A write to the PIO Set or PIO Clear registers functions as shown in Table 9-4; a read does not change the PIO Data register contents. A read of the PIO Set or PIO Clear registers returns the last value written by software in the corresponding PIO Data register (including changes made via the PIO Set and PIO Clear registers). This enables software to read back the value that would be driven if a PIO is changed from an input to an output.

Table 9-4 PIO Set and PIO Clear Registers' Effect on PIO Data Register

PIO Set Register Function			PIO Clear Register Function		
Written to PIO Set Register Bit	Old PIO Data Register Bit	New PIO Data Register Bit	Written to PIO Clear Register Bit	Old PIO Data Register Bit	New PIO Data Register Bit
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	1	0	0
1	1	1	1	1	0

9.5.6 Hardware-Related Considerations

Choose your PIOs wisely. The following PIO signals are multiplexed with alternate signals that may be used by emulators: PIO8, PIO15, PIO33, PIO34, and PIO35. Consider any emulator requirements for the alternate signals before using these pins as PIOs. For more information, see Chapter 4, “Emulator Support.”

9.5.7 Software-Related Considerations

- The PIO Set and PIO Clear registers provide an efficient, nondestructive means to modify specific data values for the PIOs. Previous Am186 devices required the user to perform a read-mask-modify-write approach when modifying PIO data values (and preserving the values of other bits in the register). With the PIO Set and PIO Clear registers, specific bits can be set or cleared in a single write.
- When configuring PIOs, modify only the bits relevant to your application by reading the existing register value, masking the bits needed, and writing the register.

9.5.8 Comparison to Other Devices

- The PIO registers are similar to previous Am186 controller implementations. The PIO Mode, PIO Direction, and PIO Data registers behave similarly; the PIO Set and PIO Clear registers have been added.
- The Am186CC/CH/CU microcontrollers offer 48 PIOs rather than the 32 offered in other Am186 implementations, and these PIOs have different pin assignments.

9.6 INITIALIZATION

On both an external and internal reset, the following occurs:

- The PIOMODE0 register defaults to 0000h, the PIODIR0 register to 1EFFh, PIOMODE1 to 0000h, PIODIR1 to 9FFFh, PIOMODE2 to 0000h, and PIODIR2 to FFF1h. This defaults the PIOs to various configurations as shown in Table 9-1 on page 9-3. (PIO8, PIO13, PIO14, PIO15, PIO29, PIO30, PIO33, PIO34, and PIO35 default to their alternate operation. The remaining PIOs default to the PIO function.) System initialization code must reconfigure PIOs as required.
- The PIOSETx and PIOCLRx registers default to 0000h.
- The PIODATAx registers default to values dependent on the system configuration. For more information, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

CHAPTER

10 PROGRAMMABLE TIMERS

10.1 OVERVIEW

There are three 16-bit programmable timers in the Am186CC/CH/CU microcontrollers. Timers 0 and 1 are identical and may be used to generate periodic external signals or waveforms or to count or time external events. Each of these two timers has an input and an output pin. Timer 2 is an internal timer which can be used to prescale Timers 0 and 1 to provide longer time-out periods, or to generate DMA requests for the general-purpose DMA channels (see Chapter 8, “DMA Controller”). All three timers can be programmed to generate periodic interrupts.

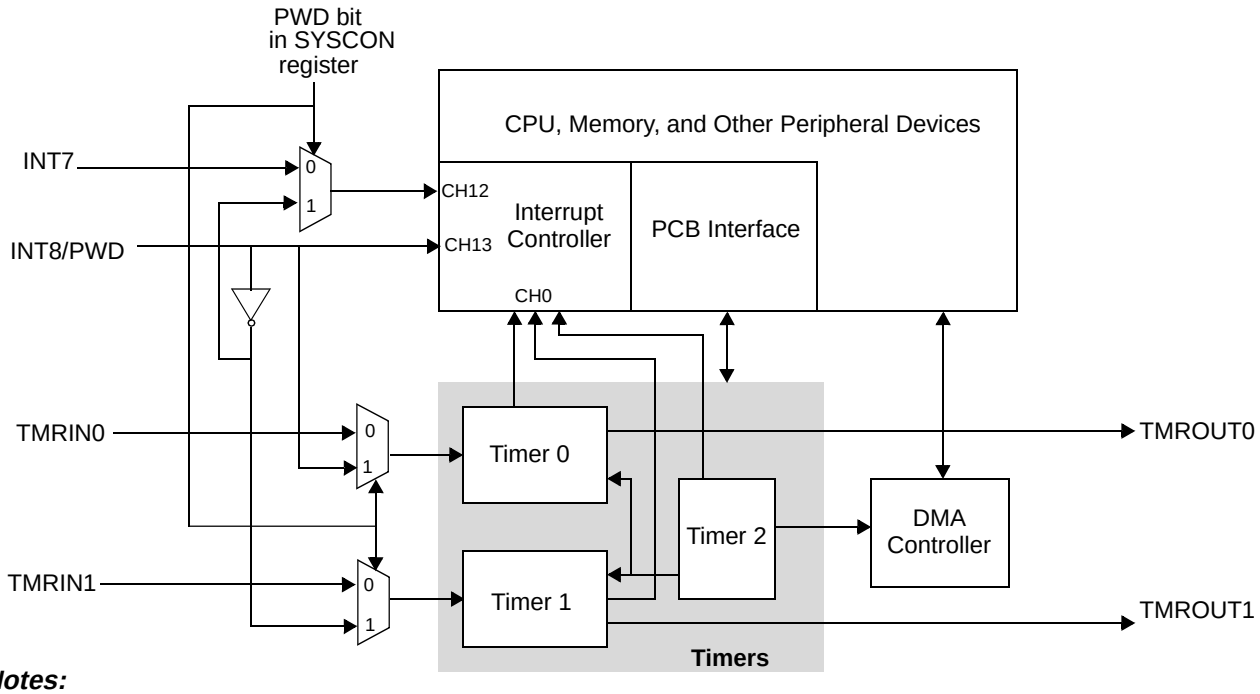
The source clock for Timer 2 is one-fourth of the CPU clock frequency (every fourth CPU clock tick). The source clock for Timers 0 and 1 can be the timer input pin, Timer 2, or one-fourth of the CPU clock,.

The microcontroller also provides a pulse width demodulation (PWD) option for measuring the Low state and High state durations of a toggling input signal.

10.2 BLOCK DIAGRAM

Figure 10-1 shows the block diagram for the programmable timers.

Figure 10-1 Programmable Timers Block Diagram



Notes:

1. *In PWD mode, the TMRIN0, TMRIN1 and INT7 pins can be used as PIOs. If INT7 is to be used as an external interrupt in PWD mode, it must be programmed to use the shared interrupt channel (channel 14).*

10.3 SYSTEM DESIGN

Table 10-1 lists the programmable timer signals that are multiplexed with other microcontroller functions. Pinstraps are sampled only at external reset and do not affect the pin's other functions, so they are not shown in this table. When TMRIN0 or TMRIN1 is programmed as a PIO, the corresponding signal is held high (asserted) internally, except in PWD mode where TMRIN0 is replaced with INT8 and TMRIN1 is replaced with the inverse of INT8. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin.

Table 10-1 Programmable Timer Multiplexed Signals

Signal	Function	Multiplexed Signal(s)	Default Signal
PWD	Pulse-width demodulator	INT8 PIO6	PIO6
TMRIN0	Timer inputs	PIO27	PIO27
TMRIN1		PIO0	PIO0
TMROUT0	Timer outputs	PIO28	PIO28
TMROUT1		PIO1	PIO1

10.4 REGISTERS

The registers listed in Table 10-2 program the timers. Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 10-2 Programmable Timers Register Summary

Offset	Register Mnemonic	Register Name	Description
340h	T0CON	Timer 0 Mode/Control	Controls the functionality of Timer 0.
342h	T0CNT	Timer 0 Count	The value of this register is incremented by 1 for each timer event until the compare value is reached.
344h	T0CMPA	Timer 0 Maxcount Compare A	This register holds a compare value for T0CNT.
346h	T0CMPB	Timer 0 Maxcount Compare B	This register holds a compare value for T0CNT.
348h	T1CON	Timer 1 Mode/Control	Controls the functionality of Timer 1.
34Ah	T1CNT	Timer 1 Count	The value of this register is incremented by 1 for each timer event until the compare value is reached.
34Ch	T1CMPA	Timer 1 Maxcount Compare A	This register holds a compare value for T1CNT.
34Eh	T1CMPB	Timer 1 Maxcount Compare B	This register holds a compare value for T1CNT.
350h	T2CON	Timer 2 Mode/Control	Controls the functionality of Timer 2.
352h	T2CNT	Timer 2 Count	This value of this register is incremented by 1 for each timer event until the compare value is reached.
354h	T2CMPA	Timer 2 Maxcount Compare A	This register holds a compare value for T2CNT.

10.5 OPERATION

10.5.1 Usage

Note: *If Timer 0 or Timer 1 is being used without the associated TMRIN pin, the pin must be held high or programmed as a PIO, otherwise the timer will not increment. Before using the programmable timers, ensure multiplexed pins are configured to reflect the use of the timers and not other functionality (see Table 10-1).*

1. Clear the current count by writing zero to the TxCNT register.
2. Specify the timer maximum count by writing to the Timer Maxcount Compare (TxCMPy) registers for the timer being used.
3. Specify the actions taken when the timer count reaches maximum by setting bits in the corresponding Timer Mode and Control (TxCON) register.
4. Enable the timer by setting both the EN and $\overline{\text{INH}}$ bits in the corresponding Timer Mode and Control (TxCON) register.

The timer count registers can be read or written at any time, regardless of whether the corresponding timer is running. The timers count from their initial value to the programmed compare value and then reset on the same clock. The value in the timer count register never equals the compare value.

If the external pins are used (Timer 0 and Timer 1), the PIO Mode and PIO Direction bits for these pins must be configured for alternate operation. These pins are configured as PIOs at external and internal reset. For more information, see Chapter 9, “Programmable I/O Signals.”

10.5.2 Timer 2

When enabled, Timer 2 increments the T2CNT register value at every fourth processor clock. After the timer increments, the microcontroller compares the T2CNT value with the value of the T2CMPA register. When the two values are equal, the microcontroller takes the following actions:

- Resets T2CNT to zero and sets the MC (Max Count reached) bit in the T2CON register.
- If the INT bit is set in T2CON, generates an interrupt request. Software must clear the MC bit.
- Sends a pulse to Timer 0 and Timer 1 which can be used to increment those timers.
- Sends a DMA request to the general-purpose DMA—the DMA may act on or ignore this request depending on how it is programmed.
- If the CONT (continuous mode) bit is zero, clears the EN (enable) bit and the timer stops counting. If CONT is one, the timer remains enabled and continues counting.

Since the comparison is done after the count is incremented, if T2CNT and T2CMPA are initially set to the same value, the comparison of T2CNT to T2CMPA will not be equal until 4 • 0FFFFh processor clocks after the counter is enabled.

10.5.3 Timer 0 and Timer 1

Timers 0 and 1 provide identical functionality. Unlike Timer 2, Timers 0 and 1 each have an input and output pin associated with the timer. They can also use Timer 2 as a prescaler providing a 32-bit time-out count.

Three bits in the control register and the external TMRIN pin control the way Timer 0 and Timer 1 count:

- When the EXT (external clock) bit is set, the TMRIN signal provides the clock for the associated timer. In this mode, the timer count increments once for each low to high transition on the TMRIN pin. The external clock speed cannot be greater than one fourth of the processor clock. The timer output can take up to six clock cycles to respond to the clock or gate input because of internal synchronization and pipelining of the timer circuitry.
- When the RTG (retrigger) bit is set, a low to high transition on TMRIN resets the value in the timer's current count register. The timer counts during both the high and low phases of the TMRIN signal.
- When the P (prescaler) bit is set, Timer 2 provides the clock for the associated timer. The timer increments once each time Timer 2 reaches its maximum count.

Table 10-3 summarizes the behavior of the timers.

Table 10-3 Timer 0 and Timer 1 Behavior

TxCON Bit Value			TMRIN		
EXT	**RTG**	**P**	**Low**	**High**	**Low -> High**
0	0	0	Hold	1/4 processor clock	No effect
0	0	1	Hold	Timer 2 time-out	No effect
0	1	0	1/4 processor clock	1/4 processor clock	Resets count
0	1	1	Timer 2 time-out	Timer 2 time-out	Resets count
1	x	x	No effect	No effect	Increments count

Timers 0 and 1 provide two maximum count compare registers, TxCMPA and TxCMPB. The setting of the ALT (alternate compare) bit determines whether one or both of these compare registers are used. When ALT is zero, only TxCMPA is used. When ALT is one, both compare registers are used.

When ALT is zero, the timer behaves as follows:

- Each time the timer increments, it compares the value in TxCNT to the value in TxCMPA.
- If the compare is not equal, the timer:
  - Holds TMROUTx High.
- If the compare is equal, the timer:
  - Pulses TMROUTx Low for a single processor clock.
  - Resets the TxCNT register to zero.
  - Sets the MC (maxcount reached) bit. Software must clear the MC bit.
  - If the INT bit is set in TxCON, the timer generates an interrupt request.
  - If the CONT (continuous mode) bit is zero, the timer clears the EN (enable) bit and the timer stops counting. If the CONT bit is one, the timer remains enabled and continues counting.

When ALT is set and the timer is using TxCMPA (initial value after reset), the timer behaves as follows:

- The RIU (Register-In-Use) bit is zero (this is a read-only bit).
- Holds the TMROUTx signal High (inverse of RIU).
- Each time the timer increments, it compares the value in TxCNT to the value in TxCMPA.
- If the compare is equal, the timer:
  - Resets the TxCNT register to zero.
  - Sets the MC (maxcount reached) bit. Software must clear the MC bit.
  - If the INT bit is set in TxCON, the timer generates an interrupt request.
  - Sets the RIU bit and performs the next compare against TxCMPB.

When ALT is set and the timer is using TxCMPB, the timer behaves as follows:

- The RIU (Register-In-Use) bit is one (this is a read-only bit).
- Holds the TMROUTx signal Low (inverse of RIU).
- Each time the timer increments, it compares the value in TxCNT to the value in TxCMPB.
- If the compare is equal, the timer:
  - Resets the TxCNT register to zero.
  - Sets the MC (maxcount reached) bit.
  - If the INT bit is set in TxCOT, the timer generates an interrupt request. Software must clear the MC bit.
  - Clears the RIU bit and TMROUTx transitions to high.
  - If the CONT (Continuous Mode) bit is zero, the timer clears the EN (Enable bit) and the timer is disabled. If the CONT bit is set, the timer remains enabled and performs the next compare against TxCMPA.

Because the comparison is done after the count is incremented, if TxCNT and TxCMPA are set to the same value, the comparison of TxCNT to TxCMPA will not be equal until the current count reaches its maximum value, wraps around through zero and counts to the TxCMPA value. Setting TxCMPB to zero provides the maximum time-out for the second phase of the timer.

Setting the ALT bit and using the two compare registers allows Timer 0 and Timer 1 to generate waveforms on the associated TMROUT pins.

10.5.4 Requesting Interrupts

The INT bits in the T0CON, T1CON, and T2CON registers control interrupt request generation when a maximum count is reached. The request remains asserted for as long as the MC bit in the TxCON register is set. Software must clear this bit.

If the maximum count and compare registers are both set to 0000h, the timer associated with that compare register counts from 0000h to FFFFh before requesting an interrupt. With a 40-MHz clock, a timer configured this way interrupts every 6.5536 ms.

When the ALT bit is set for Timer 0 or Timer 1, the MC bit is set both when the timer reaches the TxCMPA value and when the timer reaches the TxCMPB value. Software can differentiate these two conditions by examining the RIU bit. The RIU bit is 1 when the

TxCMPA value is reached (timer is now comparing against TxCMPB). The RIU bit is 0 when the TxCMPB value is reached (timer is now comparing against TxCMPA).

10.5.5 Software Polling

Software can poll the MC bit in the T0CON, T1CON, and T2CON registers to monitor timer status rather than using interrupts. This bit must be cleared by software.

10.5.6 Generating Waveforms

When programmed to use both compare values (ALT bit in TxCON is 1), Timer 0 and Timer 1 can generate waveforms on the associated TMROUT pin. The TxCMPA value determines the duration of the High phase of the output waveform. The TxCMPB value determines the duration of the Low phase of the output waveform. For more information, see the timer examples available on the AMD website at **ftp.amd.com**.

10.5.7 Pulse Width Demodulation

For many applications, such as bar-code reading, it is necessary to measure the width of a signal in both its High and Low phases. The Am186CC/CH/CU microcontrollers provide a pulse width demodulation (PWD) option to fulfill this need. The PWD bit in the System Configuration (SYSCON) register enables the PWD option. Note that the Am186CC/CH/CU microcontrollers do not support analog-to-digital conversion.

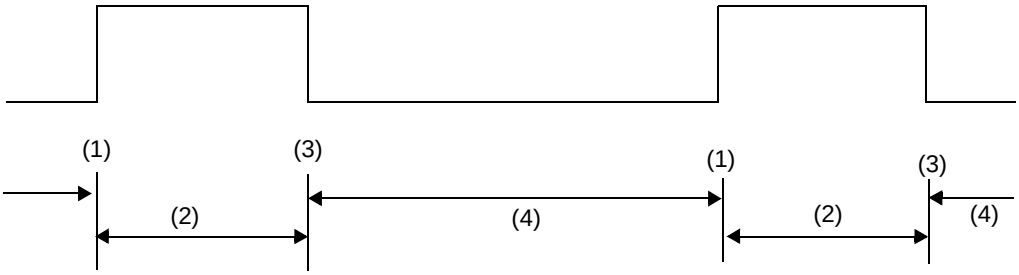
Figure 10-1 on page 10-1 shows the routing of signals when pulse width demodulation is either enabled or disabled. The waveform for PWD mode is input on the INT8/PWD pin. This pin is of type Schmitt trigger in both normal interrupt and PWD modes. Note that this pin is multiplexed with PIO6 and defaults to the PIO function at external and internal reset.

In PWD mode, software uses Timer 0 and Timer 1 to measure the High and Low pulse width of the input signal. Interrupt 8 (Channel 13, type 1dh) and interrupt 7 (Channel 12, type 1ch) notify software of the transitions of the measured input signal.

Timer 0 starts its count on the Low-to-High transition on the PWD input and counts the High signal duration. Timer 1 starts its count on the High-to-Low transition on the PWD input and counts the Low signal duration. The Low-to-High transition of the PWD input generates an interrupt request using Channel 13 (type 1dh). The High-to-Low transition of the PWD input generates an interrupt request using Channel 12 (type 1ch).

Figure 10-2 shows the behavior of the PWD function for a typical input waveform.

Figure 10-2 Pulse Width Demodulation Example



1. A Channel 13 (INT8) interrupt request is generated.
2. Timer 0 counts during the high phase of the input signal.
3. A Channel 12 (INT7) interrupt request is generated.
4. Timer 1 counts during the low phase of the input signal.

As shown in Figure 10-1 on page 10-1, entering pulse width demodulation mode by setting the PWD bit in the SYSCON register does not have any direct effect on the timer block other than to reroute the TMRIN0 and TMRIN1 signals. The timers retain their full functionality and programmability.

In the typical pulse width demodulation application, configure the T0CON and T1CON registers with a write of C001h (EN + $\overline{\text{INH}}$ + CONT). The ISR for Channel 13 reads the value in T1CNT to determine the length of the Low phase of the signal, and then resets the T1CNT register to zero. The interrupt service routine (ISR) for Channel 12 reads the value in T0CNT to determine the length of the High phase of the signal and then resets the T0CNT register to zero. Set the TxCMPA compare value high enough to ensure that the signal duration will not exceed the maximum count. The ISR should check the MC bit of the associated timer to determine if the maximum count has been exceeded. If the MC bit is set, software must then determine the appropriate response to this overflow situation. It may be sufficient to add the TxCMPA register value to the TxCNT register value to generate the correct signal duration.

10.5.7.1 Handling Short Signal Durations

In applications where the pulse width is short, it may be necessary to poll the interrupt request bits in the interrupt request (REQST) register and jump to the ISR rather than actually taking interrupts.

10.5.7.2 Handling Long Signal Durations

When the timers are configured for PWD (EN + $\overline{\text{INH}}$ + CONT), the maximum duration of each phase of the input signal should not exceed 4 • TxCMPA processor clocks because the timer increments every fourth processor clock in this configuration. To extend the maximum measurable duration using PWD, software can enable timer interrupts, use Timer 2 as a prescaler, or both.

If the INT (interrupt) bit is set in either T0CON or T1CON, the associated timer generates an interrupt request on Channel 0—type 08h for Timer 0 and type 09h for Timer 1. The ISR for these interrupts should add the programmed maxcount (the value of the TxCMPA register) to a memory location and clear the MC bit in the TxCON register each time the interrupt is taken. The ISR for channel 13 (interrupt type 1dh) must add the value of the Timer 1 memory location to the current T1CNT register to determine the duration of the Low phase of the signal. The ISR for channel 12 (interrupt type 1ch) must add the value of the Timer 0 memory location to the current T0CNT register to determine the duration of the High phase of the signal. In both cases the calculated duration must be multiplied by four to yield the total number of processor clocks.

If the P (prescaler) bit is set in either T0CON or T1CON, the associated timer increments once for each Timer 2 time-out. This increases the maximum measurable duration to 4 • T2CMPA • TxCMPA. However, the precision of the measurement falls from within four processor clocks of the actual value to within 4 • T2CMPA processor clocks of the actual value. For this reason, the value of T2CMPA should be kept as small as possible. This solution uses fewer processor cycles and has less of an effect on system performance than the use of timer interrupts.

In applications where extremely long signals need to be measured, both the P bit and the INT bit can be set either in T0CON or T1CON or both.

10.5.8 Software-Related Considerations

- Timer 2 can generate a DMA request. For more information, see Chapter 8, “DMA Controller.”
- Timer 0 and Timer 1 each have two 16-bit count compare registers. These registers can be used together to expand the time resolution for the timers. For more information, see the Timer Mode and Control registers in the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

10.5.9 Comparison to Other Devices

The programmable timers are 100% compatible with the timers in the Am186ES and Am186ED microcontrollers.

10.6 INITIALIZATION

On both an external and internal reset, the following occurs:

- The values of all the timer registers are cleared to 0000h.
- All the timer signals default to PIO operation (see Table 10-1 on page 10-2).
- The PWD bit in the SYSCON register is cleared to 0.

CHAPTER

11 WATCHDOG TIMER

11.1 OVERVIEW

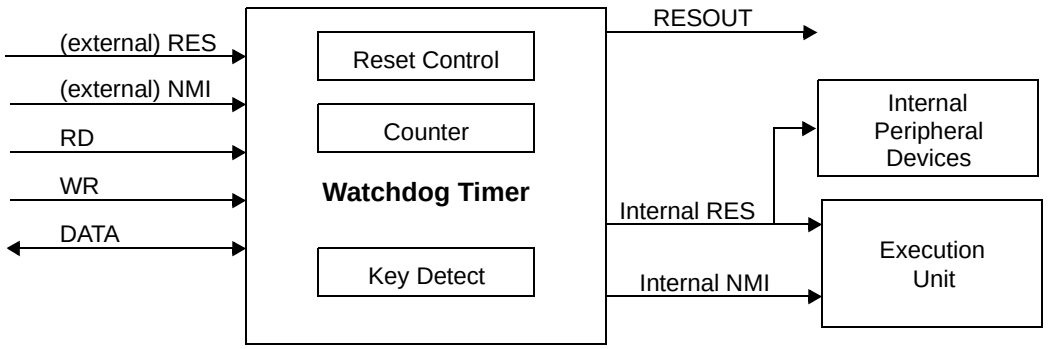
The Am186CC/CH/CU microcontrollers provide a full-featured watchdog timer that can generate nonmaskable interrupts (NMIs), internal resets, and system resets when the time-out value is reached. The time-out value is programmable and ranges from 2^{10} to 2^{26} processor clocks. Throughout this chapter, an *external reset* refers to a reset of the microcontroller as initiated by the $\overline{\text{RES}}$ signal, which resets the CPU and the internal peripherals. *Internal reset* refers to a reset initiated by the watchdog timer. *System reset* refers to a reset of the external peripherals connected to the controller as initiated by the $\overline{\text{RESOUT}}$ signal, which is pulled Low during an external reset and can be pulled Low during an internal reset.

The watchdog timer provides a method to regain control when a system has failed due to a software error or to the failure of an external device to respond in the expected way. Software errors can sometimes be resolved by recapturing control of the execution sequence through a watchdog-timer-generated NMI. When an external device fails to respond, or responds incorrectly, it may be necessary to reset the controller or the entire system, including external devices. The watchdog timer provides the flexibility to support both NMI and reset generation. The watchdog timer is enabled at reset.

11.2 BLOCK DIAGRAM

Figure 11-1 shows a block diagram of the watchdog timer.

Figure 11-1 Watchdog Timer Block Diagram



11.3 SYSTEM DESIGN

Table 11-1 lists the watchdog timer signals that are multiplexed with other microcontroller functions. Pinstraps are sampled only at external reset and do not affect the pin's other functions, so they are not shown in this table. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin.

Table 11-1 Watchdog Timer Multiplexed Signals

Signal	Function	Multiplexed Signal(s)	Default Signal
$\overline{\text{RES}}$	Controller reset	—	$\overline{\text{RES}}$
NMI	Nonmaskable interrupt	—	NMI

Systems that require a guaranteed recovery time from software or hardware errors should use the watchdog timer.

Generation of the internal NMI signal on the first watchdog timer time-out can be useful in systems where it may be possible to recover from glitches, corrupted data, or incorrect code without resetting either the controller or the board. This is especially true where potential data recovery is important. Such systems should have the NMI interrupt handler routine in ROM to ensure that it has not been corrupted by runaway code. However, in most systems, the interrupt table, which must be located at address 00000h, is located in RAM and so is subject to corruption.

Generation of the RESOUT signal should be used in systems where a system hang may be caused by incorrect behavior of an external device. The watchdog timer reset duration, and therefore the duration of the RESOUT signal on a watchdog timer reset, is 2^{16} processor clocks. This allows sufficient time for external devices to reach their reset state.

The watchdog timer must function in all cases where either the software or external devices have failed to respond appropriately. The watchdog timer has incorporated several features to ensure that this is the case.

- The watchdog timer is active after reset.
- The watchdog timer's default configuration after a power-on reset is to generate a reset on the first time-out and to assert the RESOUT signal.
- Software can disable the Watchdog Timer Control (WDTCON) register after reset and, while it is disabled, it can be written any number of times. When software enables the watchdog timer, the register becomes read-only except for two flag bits. This allows bootup or monitor code to disable the watchdog timer until the system has been configured.
- Each single write to the watchdog timer must be preceded by writes of a keyed sequence. Detection of the keyed sequence allows a single write to the WDTCON register.
- The watchdog timer time-out counter can only be reset by the initial enabling write to the WDTCON register or by writing a special key sequence to the WDTCON register.

These features guarantee that the watchdog timer is not affected by runaway code.

Software can determine whether an NMI or reset event was caused by an external source or by the watchdog timer by reading the WDTCON register. The NMIFLAG bit is set when the watchdog timer generates an NMI; the RSTFLAG bit is set when the watchdog timer generates a reset. Software can clear, but not set, these bits.

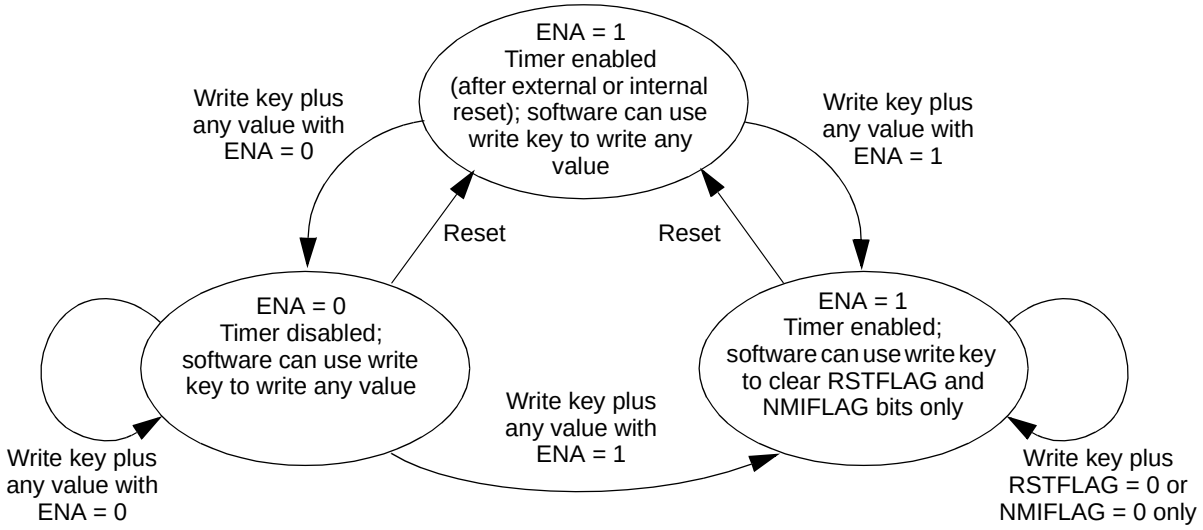
11.4 REGISTERS

The register shown in Table 11-2, WDTCON, programs the watchdog timer. Figure 11-2 illustrates the rules for accessing the WDTCON register. Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 11-2 Watchdog Timer Register Summary

Offset	Register Mnemonic	Register Name	Description
3E0h	WDTCON	Watchdog Timer Control	Controls the watchdog timer.

Figure 11-2 Access to the WDTCON Register



Notes:

Only one write is allowed to the WDTCON register after each write key sequence of 3333h followed by CCCCh.

11.5 OPERATION

11.5.1 Usage

1. Enable the watchdog timer by writing the keyed sequence of 3333h followed by CCCCh to the WDTCON register address.

   This sequence opens the WDTCON register for a single write. Any number of processor cycles, including memory and I/O reads and writes, can be inserted between the two halves of the key or between the key and the writing of data as long as they do not read or write the WDTCON register. The write key sequence must be repeated for each single write.

2. After enabling the watchdog timer, periodically reset it by writing the keyed sequence of AAAAh followed by 5555h to the WDTCON register address.

   As with the write key, any number of processor cycles, including memory and I/O reads and writes, can be inserted between the two halves of the key as long as they do not access the WDTCON register. The key itself resets the counter; no further writes are necessary. Note that the clear-count key cannot be initiated while the write key is active. This would result in the value of AAAAh being written to the WDTCON register.

11.5.2 Overview

Because the watchdog timer is enabled after reset, it is important for start-up code to program the watchdog timer before the initial time-out period expires. The time-out period after a watchdog timer reset is 2^{26} clock cycles.

All writes to the WDTCON register must be preceded by the write key sequence. The write key is a special two-write sequence to the WDTCON register address. The value of the key is not written to the WDTCON register but is used by internal logic to open the register for a single write. If a read-modify-write sequence is desired, the read must take place before the key is written because a read of WDTCON resets the keyed sequence.

The system's start-up code can either enable or disable the watchdog timer. When enabled, the watchdog timer cannot be disabled until a reset occurs. If disabled, it can be enabled later by software. The reset start-up code should check the WDTCON register to see if the RSTFLAG bit is set. If set, then the last reset was due to a watchdog timer time-out. What actions are taken is system dependent; however, possible actions include signaling another device that there is a problem, performing a more extensive test of hardware systems, or requesting reset of remote devices.

Debug monitor software (such as AMD's E86MON™ software) can disable the watchdog timer, allowing the user to interact with the monitor without having to refresh the watchdog timer. The application code can then enable or disable the watchdog timer in its own start-up routine.

In systems that program the watchdog timer to generate an NMI, the NMI service routine should check the WDTCON register to see if the NMIFLAG bit is set. If this bit is set, it indicates that an NMI due to a watchdog timer time-out occurred. Software should clear this bit so that subsequent external NMIs are not confused with watchdog timer NMIs. What actions are taken are system dependent; however, possible actions include examining the state of the DMA controller to determine whether DMA usage is preventing instruction execution, polling external devices for status, or re-execution of all or part of the system start-up code.

Code that supports the watchdog timer should be divided into two parts. The main loop of the application, or some section of code that is periodically executed but not interrupt driven, should set a flag indicating that execution has passed through this code loop. A second piece of code that is interrupt driven, typically a timer interrupt, should check the value of the flag. If the flag is set, the interrupt service routine (ISR) should write the watchdog timer clear-count key, resetting the time-out counter to zero. If the flag is not set, the ISR has several options: wait for a watchdog timer time-out to let the reset or NMI code handle the problem; attempt to determine what the problem is; or continue normal execution with the expectation that the flag may be set at some later iteration. Because transfer of control to an ISR does not require non-ISR code to be executing correctly, it is important that the ISR code not reset the time-out counter unless the flag is set.

11.5.3 Hardware-Related Considerations

- Pins that are latched on reset (pinstraps) are not resampled during a watchdog-timer reset.
- If the external reset ($\overline{\text{RES}}$) signal is asserted while the watchdog timer is performing a watchdog-timer reset, the external reset takes precedence over the watchdog-timer reset. This means that the RESOUT signal asserts as with any external reset and the WDTCON register does not have the RSTFLAG bit set. In addition, the part exits reset based on the external reset timing (i.e., 4.5 clocks after the deassertion of $\overline{\text{RES}}$ rather than 2^{16} clocks after the watchdog timer time-out occurred).

11.5.4 Software-Related Considerations

- Even if the watchdog-timer default configuration is appropriate for the application, software should always perform an enabling write to the watchdog timer. This write causes most of the fields of the WDTCON register to become read-only, preventing run-away code from disabling or otherwise modifying the watchdog timer behavior.
- If a watchdog-timer time-out occurs when the timer is programmed with WRST cleared, an NMI is generated, the time-out counter is reset, and the NMIFLAG bit is set. If the NMIFLAG bit is not cleared before a second watchdog timer time-out, a reset is generated regardless of the setting of WRST.
- The watchdog timer can generate a nonmaskable interrupt (NMI). This interrupt can be taken at any time. Unlike the maskable interrupts, the controller is not inhibited from taking a second NMI request while the NMI ISR is executing. Therefore, a watchdog timer NMI can interrupt, or be interrupted by, an externally generated NMI.

11.5.5 Comparison to Other Devices

The watchdog timer is based on the watchdog timer in the Am186ES and Am186ED microcontrollers, with the following enhancements:

- Multiple writes are allowed to the WDTCON control register following reset as long as these writes have the enable bit cleared. When a write is detected with the enable bit set, the control register becomes read-only except for the NMI Flag (NMIFLAG) bit and the Reset Flag (RSTFLAG) bit.
- The time-out counter is automatically reset by a write that enables the watchdog timer.
- A read to the WDTCON register does not clear the NMIFLAG or RSTFLAG bits. Software must write a 0 value to each of these bits to clear them. Writing a 1 to these bit positions has no effect.
- The watchdog timer can generate an external signal when a watchdog-timer reset event occurs.

11.6 INITIALIZATION

At reset, the following occurs:

- After an external reset, the watchdog timer is enabled and programmed to generate a reset including generation of the RESOUT signal on time-out, the RSTFLAG bit is cleared, and the time-out value is 2^{26} clock cycles.
- After a watchdog-timer reset, the watchdog timer is enabled and programmed to generate a reset on the time-out, the RSTFLAG bit is set, and the time-out value is 2^{26} clock cycles. The EXRST bit, which determines whether RESOUT is asserted for watchdog timer resets, retains its previously programmed value.
- A watchdog-timer reset affects the microcontroller the same as an external reset, except for the following:
  - Pinstraps are not sampled.
  - The RESCON register is not reset.
  - The RSTFLAG bit is cleared by an external reset, and set in an internal reset.
  - The EXRST bit is set by an external reset, and unchanged by an internal reset.

CHAPTER

12 SERIAL COMMUNICATIONS OVERVIEW

12.1 OVERVIEW

The Am186CC/CH/CU microcontrollers support both asynchronous and synchronous serial communications. These features are described in the chapters indicated in the bullets below. The remainder of this chapter shows some of the trade-offs of using the various serial communications features available on the microcontroller and provides a brief overview of serial communications.

- Two asynchronous serial ports, the Universal Asynchronous Receiver/Transmitter (UART) and High-Speed UART, provide full-duplex, bidirectional data transfer in RS-232 format using several industry-standard protocols: CTS/RTR and 9-Data-Bit (multidrop). The UART supports data transfer speeds of up to 115.2 Kbaud; the High-Speed UART supports speeds up to 460 Kbaud. See Chapter 13, “Asynchronous Serial Ports (UARTs).”
- One synchronous serial port provides half-duplex, bidirectional data transfer using the Synchronous Serial Interface (SSI). The microcontroller can operate as the master for multiple slave peripheral devices. The SSI supports data transfer speeds of up to 25 Mbit/s with a 50-MHz CPU clock. See Chapter 14, “Synchronous Serial Port (SSI).”

CC CH

- Four High-level Data Link Control (HDLC) channels on the Am186CC and Am186CH microcontrollers provide 8-bit element (byte or character) or frame full-duplex synchronous serial data transmission. The clock is provided by the Time Slot Assigner (TSA) for that channel. For the most part, the TSA muxing logic controls the path data takes from an HDLC to an external communication interface (or vice versa). External interfaces supported are: raw Data Communications Equipment (DCE), Pulse Code Modulation (PCM) Highway, and on the Am186CC microcontroller only, General Circuit Interface (GCI). Each TSA channel can support a burst data rate to or from an HDLC channel of up to 10 Mbit/s in both raw DCE and PCM highway modes, and up to 768 Kbit/s in GCI mode. See Chapter 15, “High-Level Data Link Control (HDLC),” Chapter 16, “HDLC External Serial Interface Configuration (TSAs),” and Chapter 17, “General Circuit Interface (GCI).”

CC CU

- The Am186CC and Am186CU microcontrollers both provide a Universal Serial Bus (USB) peripheral controller, which supports full-speed (12 Mbit/s) USB bulk, isochronous, interrupt, and control transfers as defined in the *Universal Serial Bus Specification, Revision 1.0*. The microcontroller acts as a USB peripheral device. The USB is a half-duplex, master/slave, polled bus. In other words, the microcontroller only transmits on the USB in response to a request from the USB host, usually a personal computer. There can be only one transmitter on the USB at a time. See Chapter 18, “Universal Serial Bus (USB).”

12.2 SYSTEM DESIGN

12.2.1 Multiplexed Signals

The serial interfaces in the Am186CC/CH/CU microcontrollers are multiplexed as shown in Table 12-1. Because of the multiplexing, there are some design trade-offs, as shown in the table. The figures that follow the table show how the microcontroller's serial communications features could be used in typical applications.

Table 12-1 Multiplexed Signal Trade-Offs for Serial Interfaces

Function Used			Functions Lost							
Inter-face	**Name**	**Pin**	**Inter-face**	**Name**	**Inter-face**	**Name**	**Inter-face**	**Name**	**Inter-face**	**Name**
Synchronous Communications Interfaces										
DCE Channel A CC CH	DCE_RXD_A	118	**PCM Channel A** CC CH	PCM_RXD_A	—	—	**GCI Channel A** CC	GCI_DD_A	**PIO**	—
	DCE_TXD_A	119		PCM_TXD_A	—	—		GCI_DU_A		—
	DCE_RCLK_A	117		PCM_CLK_A	—	—		GCI_DCL_A		—
	DCE_TCLK_A	116		PCM_FSC_A	—	—		GCI_FSC_A		—
	$\overline{\text{DCE\_CTS\_A}}$	123		$\overline{\text{PCM\_TSC\_A}}$	—	—		—		PIO17
	$\overline{\text{DCE\_RTR\_A}}$	122		—	—	—		—		PIO18
DCE Channel B CC CH	DCE_RXD_B	138	**PCM Channel B** CC CH	PCM_RXD_B	—	—	—	—	**PIO**	PIO36
	DCE_TXD_B	139		PCM_TXD_B	—	—	—	—		PIO37
	DCE_RCLK_B	135		PCM_CLK_B	—	—	—	—		PIO40
	DCE_TCLK_B	134		PCM_FSC_B	—	—	—	—		PIO41
	$\overline{\text{DCE\_CTS\_B}}$	137		$\overline{\text{PCM\_TSC\_B}}$	—	—	—	—		PIO38
	$\overline{\text{DCE\_RTR\_B}}$	136		—	—	—	—	—		PIO39
DCE Channel C CC	DCE_RXD_C	153	**PCM Channel C** CC	PCM_RXD_C	—	—	**GCI to PCM Con-version** CC	—	**PIO**	PIO42
	DCE_TXD_C	154		PCM_TXD_C	—	—		—		PIO43
	DCE_RCLK_C	150		PCM_CLK_C	—	—		PCM_CLK_C		PIO22
	DCE_TCLK_C	149		PCM_FSC_C	—	—		PCM_FSC_C		PIO23
	$\overline{\text{DCE\_CTS\_C}}$	152		$\overline{\text{PCM\_TSC\_C}}$	—	—		—		PIO44
	$\overline{\text{DCE\_RTR\_C}}$	151		—	—	—		—		PIO45
DCE Channel D CC	DCE_RXD_D	158	**PCM Channel D** CC	PCM_RXD_D	**Low-Speed UART**	RXD_U	**High-Speed UART** (Flow Control)		**PIO**	PIO26
	DCE_TXD_D	159		PCM_TXD_D		TXD_U				PIO20
	DCE_RCLK_D	156		PCM_CLK_D		$\overline{\text{RTR\_U}}$				PIO25
	DCE_TCLK_D	157		PCM_FSC_D		$\overline{\text{CTS\_U}}$				PIO24
	$\overline{\text{DCE\_CTS\_D}}$	24		$\overline{\text{PCM\_TSC\_D}}$		—		$\overline{\text{CTS\_HU}}$		PIO46
	$\overline{\text{DCE\_RTR\_D}}$	23		—		—		$\overline{\text{RTR\_HU}}$		PIO47
PCM Channel A CC CH	PCM_RXD_A	118	**DCE Channel A** CC CH	DCE_RXD_A	—	—	**GCI Channel A** CC	GCI_DD_A	**PIO**	—
	PCM_TXD_A	119		DCE_TXD_A	—	—		GCI_DU_A		—
	PCM_CLK_A	117		DCE_RCLK_A	—	—		GCI_DCL_A		—
	PCM_FSC_A	116		DCE_TCLK_A	—	—		GCI_FSC_A		—
	$\overline{\text{PCM\_TSC\_A}}$	123		$\overline{\text{DCE\_CTS\_A}}$	—	—		—		PIO17
PCM Channel B CC CH	PCM_RXD_B	138	**DCE Channel B** CC CH	DCE_RXD_B	—	—	—	—	**PIO**	PIO36
	PCM_TXD_B	139		DCE_TXD_B	—	—	—	—		PIO37
	PCM_CLK_B	135		DCE_RCLK_B	—	—	—	—		PIO40
	PCM_FSC_B	134		DCE_TCLK_B	—	—	—	—		PIO41
	$\overline{\text{PCM\_TSC\_B}}$	137		$\overline{\text{DCE\_CTS\_B}}$	—	—	—	—		PIO38
PCM Channel C CC	PCM_RXD_C	153	**DCE Channel C** CC	DCE_RXD_C	—	—	**GCI to PCM Con-version** CC	—	**PIO**	PIO42
	PCM_TXD_C	154		DCE_TXD_C	—	—		—		PIO43
	PCM_CLK_C	150		DCE_RCLK_C	—	—		PCM_CLK_C		PIO22
	PCM_FSC_C	149		DCE_TCLK_C	—	—		PCM_FSC_C		PIO23
	$\overline{\text{PCM\_TSC\_C}}$	152		$\overline{\text{DCE\_CTS\_C}}$	—	—		—		PIO44

Table 12-1 Multiplexed Signal Trade-Offs for Serial Interfaces (Continued)

Function Used			Functions Lost							
Inter-face	Name	Pin	Inter-face	Name	Inter-face	Name	Inter-face	Name	Inter-face	Name
PCM Channel D CC	PCM_RXD_D	158	DCE Channel D CC	DCE_RXD_D	Low-Speed UART	RXD_U	High-Speed UART	—	PIO	PIO26
	PCM_TXD_D	159		DCE_TXD_D		TXD_U		—		PIO20
	PCM_CLK_D	156		DCE_RCLK_D		$\overline{\text{RTR\_U}}$		—		PIO25
	PCM_FSC_D	157		DCE_TCLK_D		$\overline{\text{CTS\_U}}$		—		PIO24
	$\overline{\text{PCM\_TSC\_D}}$	24		$\overline{\text{DCE\_CTS\_D}}$		—		$\overline{\text{CTS\_HU}}$		PIO46
Low-Speed UART	RXD_U	158	DCE Channel D CC	DCE_RXD_D	PCM Channel D CC	PCM_RXD_D	—	—	PIO	PIO26
	TXD_U	159		DCE_TXD_D		PCM_TXD_D	—	—		PIO20
	$\overline{\text{RTR\_U}}$	156		DCE_RCLK_D		PCM_CLK_D	—	—		PIO25
	$\overline{\text{CTS\_U}}$	157		DCE_TCLK_D		PCM_FSC_D	—	—		PIO24
High-Speed UART	RXD_HU	25	DCE Channel D CC	—	PCM Channel D CC	—	—	—	PIO	PIO16
	TXD_HU	26		—		—	—	—		—
	$\overline{\text{RTR\_HU}}$	23		$\overline{\text{DCE\_RTR\_D}}$		—	—	—		PIO47
	$\overline{\text{CTS\_HU}}$	24		$\overline{\text{DCE\_CTS\_D}}$		$\overline{\text{PCM\_TSC\_D}}$	—	—		PIO46
GCI Channel A CC	GCI_DD_A	118	DCE Channel A CC CH	DCE_RXD_A	PCM Channel A CC CH	PCM_RXD_A	—	—	PIO	—
	GCI_DU_A	119		DCE_TXD_A		PCM_TXD_A	—	—		—
	GCI_DCL_A	117		DCE_RCLK_A		PCM_CLK_A	—	—		—
	GCI_FSC_A	116		DCE_TCLK_A		PCM_FSC_A	—	—		—
GCI to PCM Con-version CC	PCM_CLK_C	150	DCE Channel C CC	DCE_RCLK_C	PCM Channel C CC	PCM_CLK_C	—	—	PIO	PIO22
	PCM_FSC_C	149		DCE_TCLK_C		PCM_FSC_C	—	—		PIO23

12.2.2 Sample Applications for the Am186CC Communications Controller

CC

Figure 12-1 on page 12-4 shows an HDLC control application that uses all four HDLC channels of the Am186CC microcontroller configured as nonmultiplexed raw DCE external interfaces, in addition to the High-Speed UART port for debugging.

Figure 12-2 on page 12-4 shows a POTS linecard application that uses all four HDLC channels of the Am186CC microcontroller multiplexed to external interface A for linecard control. In addition, this application uses the UART interfaced to external interface D for debugging, and the SSI port to control the linecard voice integrated circuits. Unused external interfaces B and C are used as PIOs.

Figure 12-3 on page 12-5 shows an ISDN application for the Am186CC microcontroller that uses three HDLC channels multiplexed off external interface A, the High-Speed UART port for a serial AT modem connection to the host PC (or a USB connection to the PC can be used instead), the UART port on external interface D for debugging, and the SSI port to control the POTS interface integrated circuits. (Flow control on the High-Speed UART is also multiplexed to external interface D.)

Figure 12-4 on page 12-5 shows an ISDN application that uses the GCI-to-PCM Highway conversion feature of the Am186CC microcontroller. This application uses three HDLC channels multiplexed off external interface A, the High-Speed UART port for a serial AT modem connection to the host PC, and the debug UART port on external interface D (flow control on the High-Speed UART is also multiplexed to external interface D). External interface C is used for the converted GCI-to-PCM highway frame sync and clock required by PCM Highway codecs. External interface B is unused.

Figure 12-1 HDLC Control Application

CC

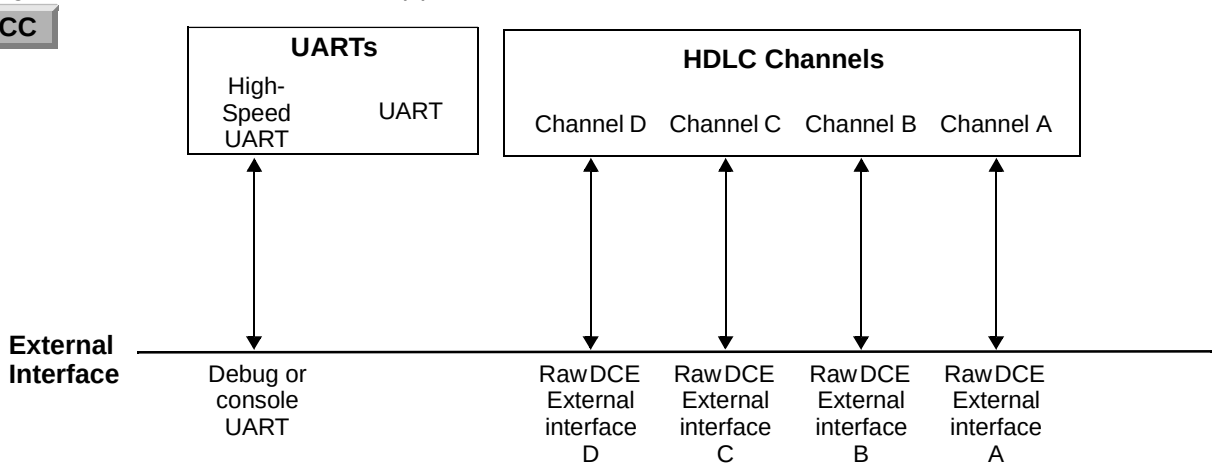


Figure 12-2 POTS Linecard

CC

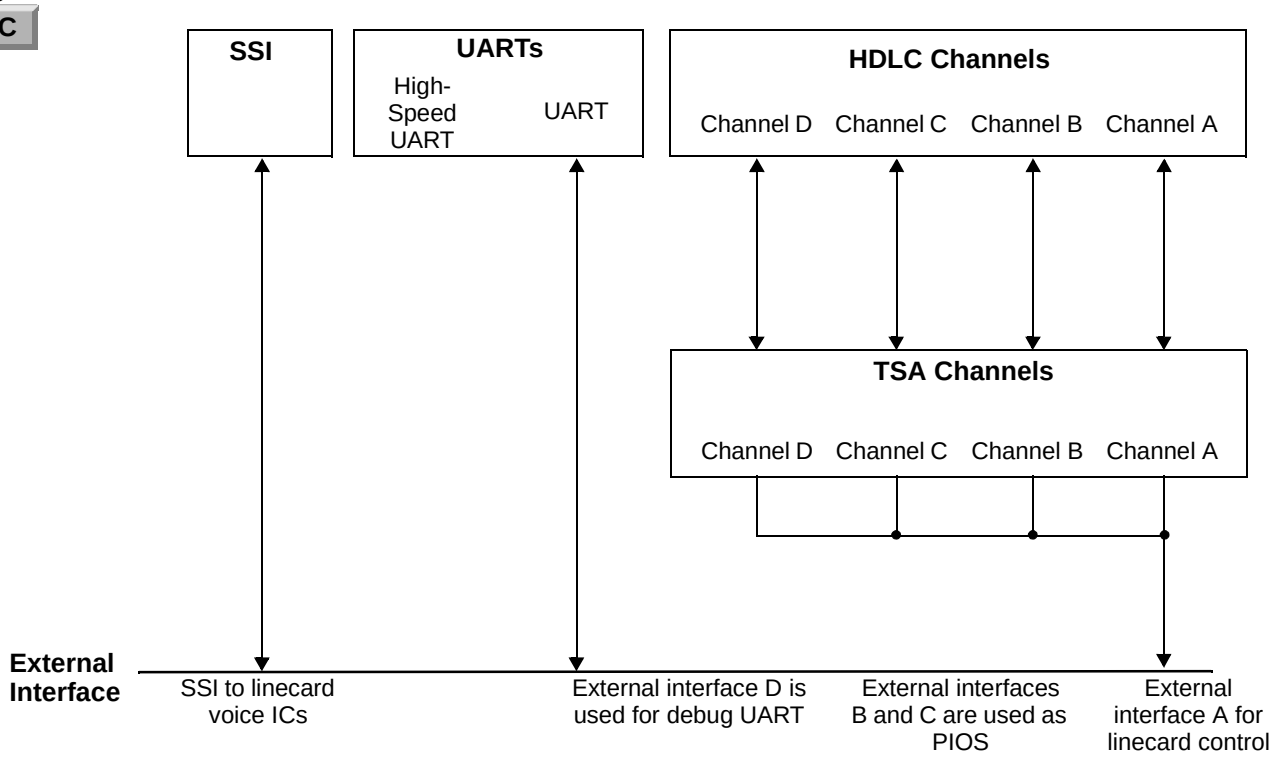


Figure 12-3 ISDN Application

CC

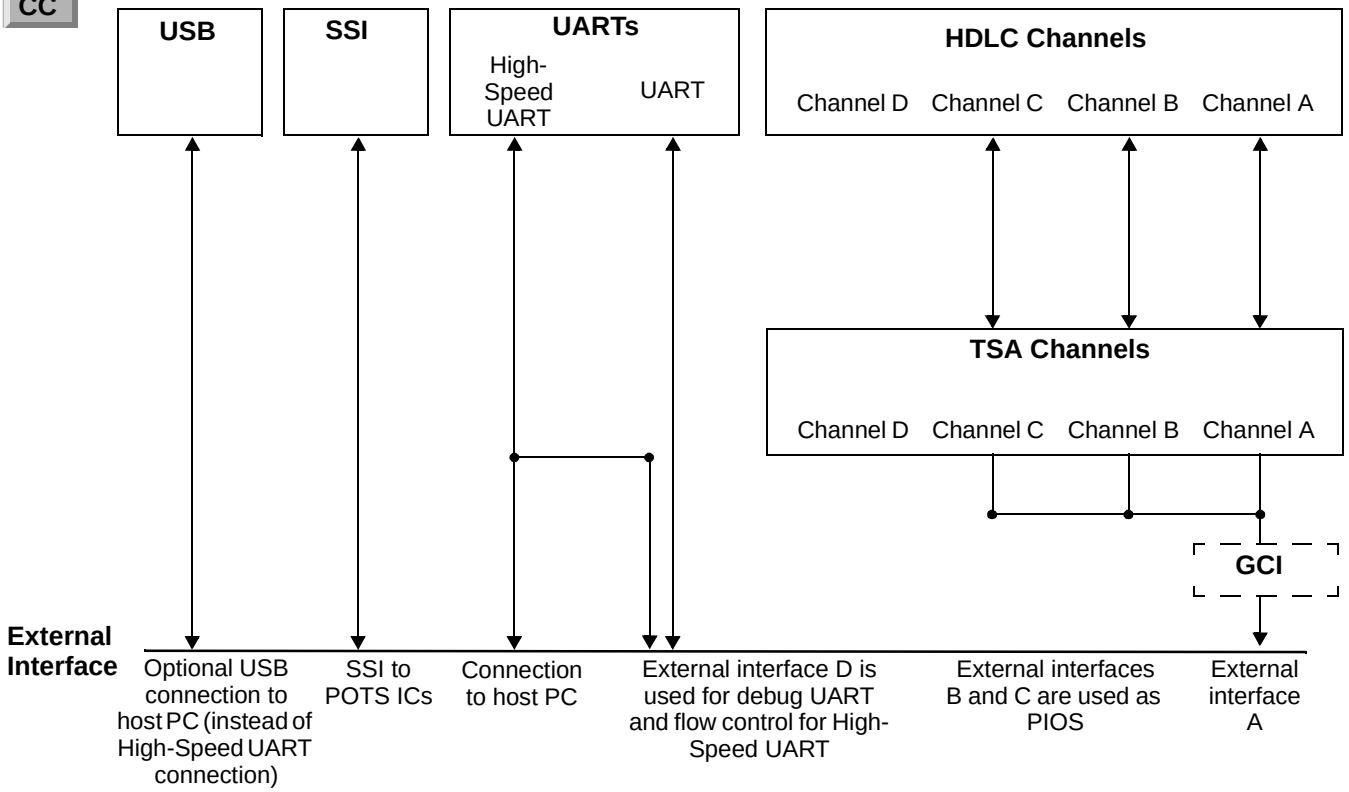
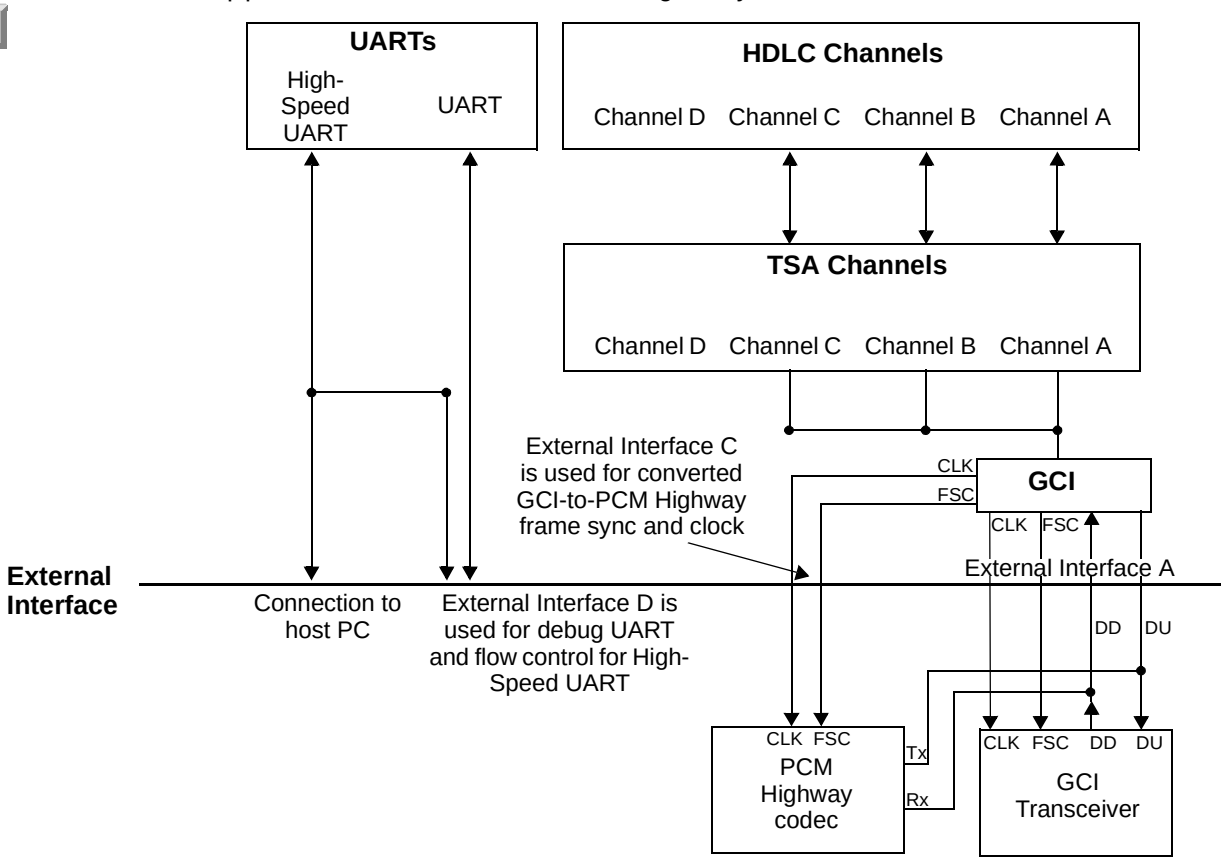


Figure 12-4 ISDN Application with GCI-to-PCM Highway Conversion

CC



12.3 SERIAL COMMUNICATIONS INTRODUCTION

In *serial* communications, one data bit at a time is transmitted through a single wire or communication channel. Because a processor data bus uses *parallel* communications (more than one data bit is transmitted at the same time through more than one wire), the communications channel must convert data to serial at the transmitter and then back to parallel at the receiver. In addition, the timing of long streams of bits must be the same for the transmitter and the receiver, or errors result.

12.3.1 Asynchronous and Synchronous Communications

In *asynchronous* serial communications, the receiver and transmitter have independent clocks. Synchronization problems are avoided by not sending long, uninterrupted streams of bits. Instead, data bits are transmitted one character at a time. Each *character* can be from four to nine data bits, and is preceded by a start bit and followed by a stop bit. The start bit, character, and stop bit together are called a *frame*. Asynchronous communication does not require continuous data so timing must only be maintained within each character; the receiver can resynchronize between frames. In addition, frames can be sequenced to form packets of data.

In *synchronous* serial communications, timing is determined by transmitting a clock signal along with the data. The channel transmits blocks of bits or characters without a start or a stop bit; the clock ensures the transmitter and receiver are synchronized. While this addresses the timing problem, the receiver must also be able to determine the beginning and end of a block of data. To achieve this, each block of data has some start and end bits. Other control information may be included. The data plus the control information is called a *frame*. The start-bit and end-bit patterns vary based on the protocol, and are sometimes called *preamble* and *postamble* bits, or *flags*.

Asynchronous communications is simpler to use but has a higher overhead than synchronous communications, and as such is better suited for small blocks of data that transmit in bursts (e.g., a keyboard or terminal). Synchronous communications is better suited for continuous large data blocks and higher speeds (e.g., HDLC frames), as many bytes of data are sent without overhead.

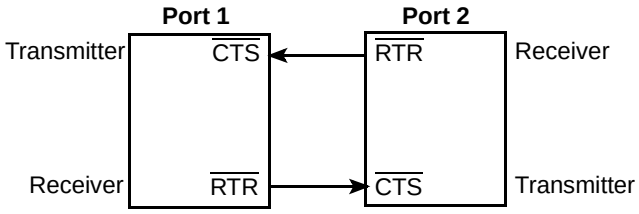
12.3.2 Hardware Flow Control

Both synchronous and asynchronous communications can have problems when a transmitter sends the data faster than the receiver is ready for it. Typically, a receiver allocates a data buffer with a certain length. After the data is processed, the receiver clears the buffer so it can receive more data. However, the receiver buffer can overflow if new data is received before the last received data is read. *Hardware flow control* is a method to eliminate the possibility of overrun errors. The Am186CC/CH/CU microcontrollers support the clear-to-send/ready-to-receive (CTS/RTR) protocol on the UART and High-Speed UART.

CC **CH** The Am186CC and Am186CH microcontrollers also support the clear-to-send/ready-to-receive (CTS/RTR) protocol on the HDLC ports.

The CTS/RTR protocol is a symmetrical interface between two serial ports, and provides flow control when both ports are sending and receiving data, as shown in Figure 12-5.

Figure 12-5 $\overline{CTS}/\overline{RTR}$ Protocol



$\overline{CTS}$ = Input signal to the transmitter; clear to send input

$\overline{RTR}$ = Output signal from the receiver; ready to receive output

12.3.3 FIFOs

Another way to reduce data overflow is to use a hardware *FIFO* (First In First Out data buffer). A hardware FIFO queues up the bytes until the receiver is ready for them. A FIFO is classified by its width and depth. The width specifies the number of bits in a word; the depth, the number of those words that can be queued. So, a 9x16 FIFO can queue 16 9-bit words before overflowing. FIFOs can also be useful when data arrives during an interrupt.

CC In the Am186CC microcontroller, FIFOs are available for the High-Speed UART, HDLC, and USB ports.

CH In the Am186CH HDLC microcontroller, FIFOs are available for the High-Speed UART and HDLC ports.

CU In the Am186CU USB microcontroller, FIFOs are available for the High-Speed UART and USB ports.

12.3.4 Polled, Interrupt, and DMA Modes

Serial communications can occur in polled, interrupt, or DMA modes. *Polled mode* disables interrupts and the DMA controller. The software loops on a status register, reading in all wait situations. In *interrupt mode*, interrupts are enabled. Software does other tasks while waiting for the interrupt. In *DMA mode*, hardware performs the entire transfer, with no software intervention except for errors. For information about interrupts, see Chapter 7, “Interrupts.” For information about DMA, see Chapter 8, “DMA Controller.”

In the Am186CC/CH/CU microcontrollers, the serial communications peripherals support the three modes as follows:

- The UART and High-Speed UART support polled, interrupt, and DMA modes.
- The SSI only supports polled mode.
- **CC** **CH** In the Am186CC and Am186CH microcontrollers, the HDLC channels support polled, interrupt, and DMA modes.
- **CC** In the Am186CC microcontroller, GCI supports polled and interrupt modes but not DMA mode.
- **CC** **CU** In the Am186CC and Am186CU microcontrollers, USB supports polled, interrupt, and DMA modes.

12.3.5 Simplex, Half-Duplex, and Full-Duplex Systems

In serial communications, a *simplex* system can transmit data in only one direction; a *half-duplex* system can send data in either direction, but not both at the same time; a *full-duplex* system can send data in both directions simultaneously.

In the Am186CC/CH/CU microcontrollers, the SSI supports half-duplex transfers, and the UART and High-Speed UART support full-duplex transfers.

CC CH In the Am186CC and Am186CH microcontrollers, the HDLC channels support full-duplex transfers.

CC CU In the Am186CC and Am186CU microcontrollers, USB supports half-duplex transfers.

CHAPTER

13 ASYNCHRONOUS SERIAL PORTS (UARTS)

13.1 OVERVIEW

The Am186CC/CH/CU microcontrollers each provide two independent asynchronous serial ports: a Universal Asynchronous Receiver/Transmitter (UART) and a High-Speed UART. The UARTs support the following features:

- Up to 115.2 Kbaud rate (UART) or up to 460 Kbaud rate (High-Speed UART)
- Automatic baudrate detection with enhancements to compensate for distortion of start bit (High-Speed UART only)
- 32-byte receive and 16-byte transmit FIFOs with threshold at half full (High-Speed UART only)
- Receive-character-matching for up to six characters including address-bit matching (High-Speed UART only)
- 7-, 8-, or 9-bit data transfers
- Address bit generation and detection in 7- and 8-data-bit frames
- Extended read and write modes that allow word-wide DMA transfers
- Multidrop protocol (9-data-bit) support
- Use of processor clock or external clock signal for generation of baud clock
- Full-duplex operation
- One or two stop bits
- Even, odd, or no parity
- Break generation and detection
- Programmable to drive either High or Low on TXD line during break
- Automatic hardware flow control using the clear-to-send/ready-to-receive (CTS/RTR) protocol
- DMA to and/or from the serial ports using the general-purpose DMA channels
- Double-buffered transmit and receive
- Individually maskable interrupt requests for the following conditions:
  - Receive FIFO threshold reached (High-Speed UART only)
  - Transmit FIFO threshold reached (High-Speed UART only)
  - Receive FIFO overflow (High-Speed UART only)
  - Receive data character match (High-Speed UART only)
  - Transmit FIFO empty (High-Speed UART only)
  - Break detected
  - Received character with address bit set
  - Receive data available
  - Transmitter able to accept new data
  - Framing error detected

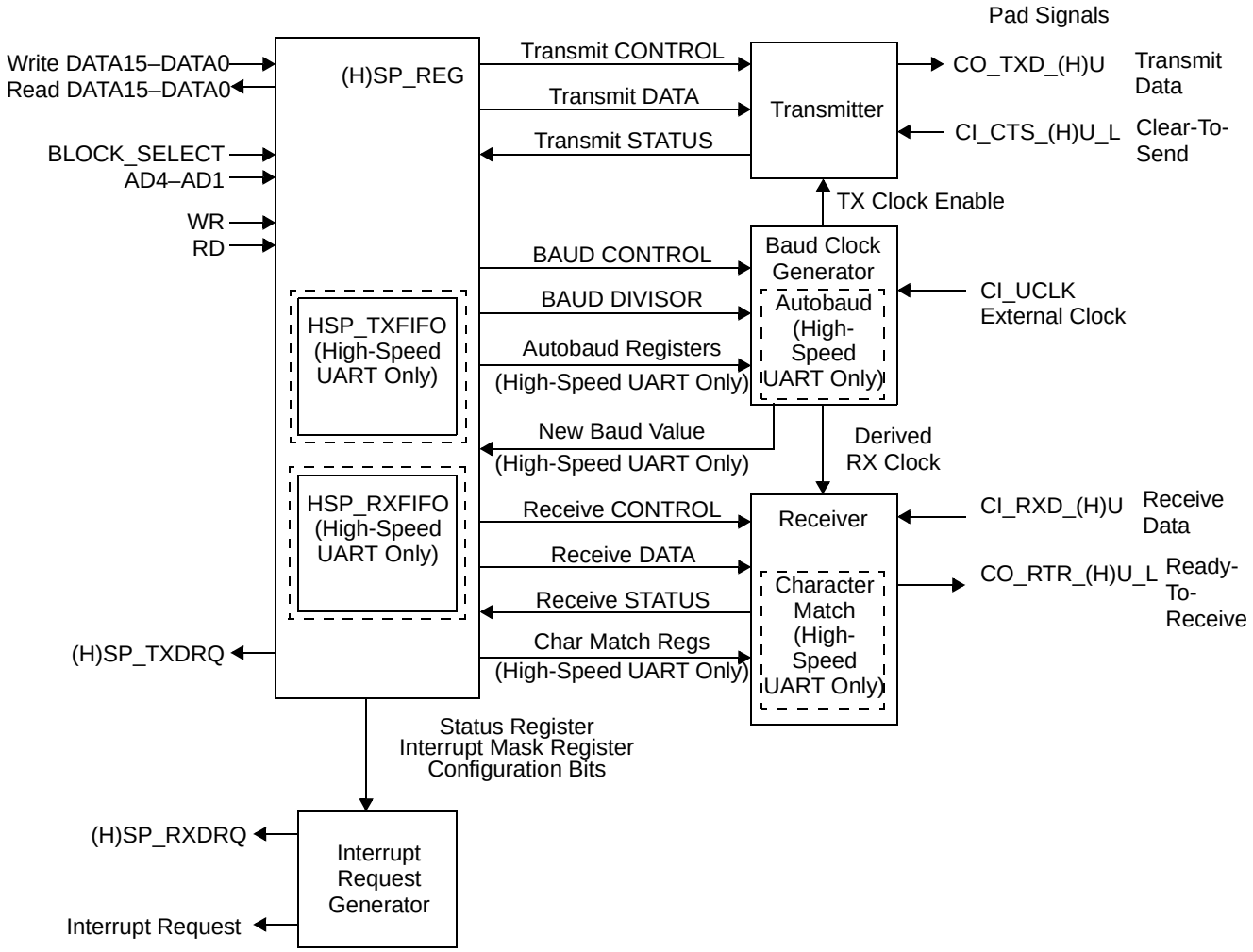
– Receive overflow error detected
– Parity error detected
– Transmitter empty
– Receive line idle

The High-Speed UART interface has been designed so that code written to run on the UART runs on the High-Speed UART with no modification other than adjusting the register offsets. The High-Speed UART interface maintains all bits and bit positions in the UART interface. The High-Speed Serial Port Status (HSPTAT) register contains additional status bits, but these bits read as zeros unless the associated function is enabled. Code written for the UART that writes zeros to reserved bits should run identically on the High-Speed UART.

13.2 BLOCK DIAGRAM

Figure 13-1 shows the UART and High-Speed UART block diagram. Features specific to the High-Speed UART are marked “High-Speed UART Only”. UART signal names begin with “SP”; High-Speed UART signal names begin with “HSP”. Signals for both start with “(H)SP.”

Figure 13-1 UARTs Block Diagram



13.3 SYSTEM DESIGN

Table 13-1 lists the UART signals that are multiplexed with other microcontroller functions. Pinstraps are sampled only at external reset and do not affect the pin's other functions, so they are not shown in this table. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin.

Table 13-1 UARTs Multiplexed Signals

Signal	Function	Multiplexed Signal(s)	Default Signal
UART			
RXD_U	Receive data UART	DCE_RXD_D PCM_RXD_D PIO26	PIO26
TXD_U	Transmit data UART	DCE_TXD_D PCM_TXD_D PIO20	PIO20
$\overline{\text{CTS\_U}}$	Clear-to-send UART	DCE_TCLK_D PCM_FSC_D PIO24	PIO24
$\overline{\text{RTR\_U}}$	Read-to-receive UART	DCE_RCLK_D PCM_CLK_D PIO25	PIO25
High-Speed UART			
RXD_HU	Receive data High-Speed UART	PIO16	PIO16
TXD_HU	Transmit data High-Speed UART	—	TXD_HU
$\overline{\text{CTS\_HU}}$	Clear-to-send High-Speed UART	$\overline{\text{DCE\_CTS\_D}}$ PCM_TSC_D PIO46	PIO46
$\overline{\text{RTR\_HU}}$	Read-to-receive High-Speed UART	$\overline{\text{DCE\_RTR\_D}}$ PIO47	PIO47

13.4 REGISTERS

The registers listed in Table 13-2 program the UARTs: 8 for the UART and 15 for the High-Speed UART. Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

CC In addition to these registers, the ITF4 bit field in the System Configuration (SYSCON) register of the Am186CC microcontroller configures external interface 4 for the HDLC or the UARTs.

CH **CU** In the Am186CH and Am186CU microcontrollers, the ITF4 bit field should be set to 10b.

Table 13-2 UARTs Register Summary

Offset	Register Mnemonic	Register Name	Description
High-Speed UART			
260h	HSPCON0	High-Speed Serial Port Control 0	Configures and enables serial port.
262h	HSPCON1	High-Speed Serial Port Control 1	Configures serial port.
264h	HSPSTAT	High-Speed Serial Port Status	Provides information about the current status of the serial port.
266h	HSPIMSK	High-Speed Serial Port Interrupt Mask	Enables interrupts based on condition of status bits.
268h	HSPTXD	High-Speed Serial Port Transmit Data	Provides data to transmitter.
26Ah	HSPRXD	High-Speed Serial Port Receive Data	Contains data read over serial line.
26Ch	HSPRXDP	High-Speed Serial Port Receive Data Peek	Reads data in Receive Data register without changing condition of serial port.
26Eh	HSPBDV	High-Speed Serial Port Baud Rate Divisor	Specifies a clock divisor for generation of the serial clock.
270h	HSPM0	High-Speed Serial Port Character Match 0	Each register can be programmed with two characters for use with automatic character matching.
272h	HSPM1	High-Speed Serial Port Character Match 1	
274h	HSPM2	High-Speed Serial Port Character Match 2	
276h	HSPAB0	High-Speed Serial Autobaud 0	Each register contains values used as baud divisors during autobaud.
278h	HSPAB1	High-Speed Serial Autobaud 1	
27Ah	HSPAB2	High-Speed Serial Autobaud 2	
27Ch	HSPAB3	High-Speed Serial Autobaud 3	
UART			
280h	SPCON0	Serial Port Control 0	Behaves the same as the High-Speed UART registers but for the UART port.
282h	SPCON1	Serial Port Control 1	
284h	SPSTAT	Serial Port Status	
286h	SPIMSK	Serial Port Interrupt Mask	
288h	SPTXD	Serial Port Transmit Data	
28Ah	SPRXD	Serial Port Receive Data	
28Ch	SPRXDP	Serial Port Receive Data Peek	
28Eh	SPBDV	Serial Port Baud Rate Divisor	

13.5 OPERATION

13.5.1 Usage

Note: *Before using the UARTs, ensure multiplexed pins are configured to reflect the use of the UARTs and not other functionality (see Table 13-1 on page 13-3).*

To use the UART and the High-Speed UART, software must program the bits described in the following procedures. The procedures include transmit, receive, and autobaud mode (High-Speed UART only). The High-Speed UART has the same basic registers as the UART (plus some additional ones). These registers are named the same except for an H in front of the High-Speed UART register name. Throughout this chapter, an “(H)” in front of the

register name indicates that both the UART and the High-Speed UART registers are being described.

13.5.1.1 Transmit

This section describes the procedure for programming a transmit. To program a receive, see page 13-6. To use autobaud mode, see page 13-7. Transfers can be done in Polled, Interrupt, or DMA modes.

13.5.1.1.1 Initializing the Transmitter

Initialize the transmitter with the following steps:

1. Disable the UART by clearing the TMODE bit of the (H)SPCON0 register to 0.

   Software can change interrupt masks without disabling the TMODE bit.

2. Set the baud rate with the (H)SPBDV register.

   For information about detecting the baud rate automatically, see "Autobaud Mode (High-Speed UART Only)" on page 13-7.

3. Set the applicable configuration options in the (H)SPCON1 register: break value, extended write, external/internal clock, and FIFOs (High-Speed UART only). If software enables FIFOs with the TFEN bit, it should also set the TFLUSH bit at the same time (or before) to flush the FIFO.

4. Set the interrupts to be taken with the (H)SPIMSK register. Bits in this register are second-level interrupt enables based on status bits in the (H)SPSTAT register. Set first-level interrupt enables in the (H)SPCON0 register (see step 5). Note that corresponding bits must be set in both registers for the interrupt to be taken. If software disables an interrupt in (H)SPIMSK, it can still read the status from the (H)SPSTAT register.

5. Set the applicable configuration options in the (H)SPCON0 register—interrupts, breaks, CTS/RTR hardware flow control, parity (odd, even, or none), address bit enable, number of data bits in serial frame (7 or 8), and stop bit length (one or two)—and enable the transmit by setting the TMODE bit to 1. All of these bits can be set simultaneously, but the TMODE bit cannot be set before any of the other bits described in steps 2–5.

13.5.1.1.2 Transmitting Data

When the transmitter is initialized, to send data:

1. Verify that the THRE bit in the (H)SPSTAT register is set to 1 to ensure the transmit register can be written without loss of data.

2. If FIFOs are being used (High-Speed UART only), instead of polling the THRE bit, verify that the FIFO is not yet full (TTHRSH bit in the HSPSTAT register is set to 1).

3. To send an address bit with a particular frame w*hen extended writes are disabled* (EXDWR in (H)SPCON1 is 0):

   a. Verify that TEMT = 1 to ensure any other transmissions have completed (the transmitter and the transmit shift register are both empty).

   b. Set the transmit AB bit in the (H)SPCON0 register to 1 if this address bit should be sent as the MSB of TDATA for this frame.

   When extended writes are enabled, write the value of the address bit with the data. In this situation, the value of TEMT does not matter.

4. Write data to the (H)SPTXD register (this sets the THRE bit to 0).

When extended writes and address bits are enabled, 9-bit data should be written to the (H)SPTXD register. The first word should include the AB bit value (set the transmit AB (AB) bit in the (H)SPTXD register to 1 followed by the address bits in the TDATA field). Then the following words should contain AB = 0 and the data for the designated address point.

Hardware stops writing to the (H)SPTXD register when TXDRQ goes to 0; when TXDRQ = 1, hardware continues writing the data. In the case of DMA, the hardware handles the data flow from the DMA unit to the SPTXD register automatically using an internal DMA signal.

5. Wait for the TEMT bit in the (H)SPSTAT register to go to 1 to indicate the on-line transfer has completed.

13.5.1.2 Receive

The procedure to program a receive is described below. To program a transmit, see page 13-5. To use autobaud, see page 13-7. Transfers can be done in Polled, Interrupt, or DMA modes.

13.5.1.2.1 Initializing the Receiver

The following procedure initializes the receiver. Before reconfiguring any options (including the baud rate), disable any receives. To do this, check that RDR=0 and IDLED=1 in the (H)SPSTAT register to ensure no data is in the receiver, and then clear the RMODE bit to 0. Interrupt masks can be changed without disabling the receiver (clearing RMODE to 0).

13.5.1.2.2 Setting the Baud Rate with the (H)SPBDV Register

1. If character matching is desired (High-Speed UART only), load the High Speed Serial Port Character Match (HSPM0, HSPM1, and HSPM2) registers with the characters to be matched. Each match register contains two character fields. Note that 00h is a valid value so if you do not want the 00h character to be matched, all six character fields should be initialized, even if it is with the same value.
2. Set the applicable configuration options in the (H)SPCON1 register: break value, extended read, external/internal clock, and, on the High-Speed UART only, FIFOs and character matching. If FIFOs are enabled with the RFEN bit, the RFLUSH bit should also be set at the same time (or before) to flush the FIFO. If comparing characters in frames with address bits, the three MAB bits should be configured. Each MAB bit setting (1 or 0) is used as the address bit for both characters in the corresponding match register. As with the character match registers, all three bits should be set.
3. Set the interrupts to be taken with the (H)SPIMSK register. Bits in this register enable interrupts based on interrupts set in the (H)SPCON0 register. Note that corresponding bits must be set in both registers for the interrupt to be taken. If an interrupt is disabled in (H)SPIMSK, the status can still be read.
4. Write all the bits in the (H)SPSTAT register to 0 to clear any status.
5. Set the applicable configuration options in the (H)SPCON0 register—interrupts, breaks, CTS/RTR hardware flow control, parity (odd, even, or none), address bit enable, number of data bits in serial frame (7 or 8), and stop bit length (one or two)—and enable the receive by setting the RMODE bit in the (H)SPCON0 register to 1. All of these bits can be set simultaneously but RMODE cannot be set before any of the other bits described in steps 1–5.

13.5.1.2.3 Receiving Data

When the receiver is initialized, to receive data:

1. Read the (H)SPSTAT register:
   a. Verify that the RDR bit is set to 1 to ensure the RDATA field contains valid data.
   b. Read the other status bits to check the status on the last byte received and clear any status bits that were set.
2. Read data from the (H)SPRXD register (this clears the RDR bit to 0). The (H)SPRXDP register is also available to peek at the data. This register is a duplicate of the receive register; however, reading it does not clear the RDR bit.

13.5.1.3 Autobaud Mode (High-Speed UART Only)

The procedure to program autobaud mode is described here. To program a transmit, see page 13-5; to program a receive, see page 13-6.

1. Verify that the TMODE and RMODE bits in the HSPCON0 register are cleared to ensure there are no transfers in progress.
2. Set the baud rate with the HSPBDV register. The register value cannot be 0; a value must be written into this register before enabling a transmit or receive.
3. Set the autobaud enable (ABAUD) bit in the HSPCON1 register to 1.
4. Optionally, to address errors in computing autobaud, program the High Speed Serial Port Autobaud (HSPAB0, HSPAB1, HSPAB2, and HSPAB3) registers with thresholds and divisors. This method compensates for distortion of start bit width resulting from external effects.
5. Clear all the bits in the HSPSTAT register to 0.
6. Set the applicable configuration options in the HSPCON0 register—interrupts, breaks, CTS/RTR hardware flow control, parity (odd, even, or none), address bit enable, number of data bits in serial frame (7 or 8), and stop bit length (one or two)—and enable the receive by setting the RMODE bit in the HSPCON0 register to 1. These bits can be set simultaneously, but RMODE cannot be set before any of the other bits described in steps 2–6. If no information is known about the data to be received, set the options to 8 bits, no parity, and no address bit. The TMODE bit must be 0; a transmit cannot be occurring while the receiver is computing autobaud.
7. Wait for the ABAUD bit in the HSPCON1 register to go to 0 to indicate that the autobaud operation is complete. The computed baud divisor is automatically copied into the HSPBDV register, and the autobaud (ABAUD) bit in the HSPCON1 register is cleared.
8. Read the new baud divisor value from the HSPBDV register and check that it is a valid divisor value or is acceptable.
9. Wait for the RDR bit in the HSPSTAT register to be set to 1 to ensure the RDATA field contains valid data.
10. Read data from the HSPRXD register (this clears the RDR bit to 0). In autobaud mode, the receiver is expecting a 1 in the least significant bit of the data (i.e., a valid autobaud character such as an ascii a). Software must check that this value is valid.

This procedure sets the autobaud rate for both the transmitter and the receiver. The receiver can continue to receive data unless there is a need to reconfigure the options (see “Receive” on page 13-6). The transmitter can now be enabled by setting the TMODE bit to 1.

13.5.2 Data

In asynchronous serial port communication, data is transmitted in *frames*. Each frame begins with a start bit (Low) and ends with one or two stop bits (High). After the start bit is transmitted, the data bits are transmitted serially, least significant bit first. Data can be 7 or 8 bits, plus an optional address bit. For more information, see “Address Bits” on page 13-9.

The data bits may be followed by an optional parity bit. A parity bit ensures there is an even or odd number of bits in the transmission. The UARTs support even, odd, or no parity. *Even parity* forces an even number of 1s in the data field by changing the parity bit as needed; *odd parity* forces an odd number of 1s. Parity checking allows the detection of single bit errors in each frame.

The UARTs also support transmission of either one or two stop bits. A second stop bit increases the spacing between back-to-back serial frames and can be useful in reducing frame errors due to clock frequency inconsistencies between devices. All these options are configured by bits in the (H)SPCON0 register. The TXD line is always held High between frames.

In asynchronous serial communication, an idle line can be differentiated from an active receive line by the absence of start bits in the data stream. In other words, a transmission of a data stream of FFh in N-8-1 mode (no parity, eight data bits, one stop bit) results in a Low bit, the start bit, every tenth bit time. When the line is truly idle, there are no Low bits. The UARTs set the IDLED bit in the (H)SPSTAT register when 40 bit times have elapsed without a Low bit and there is unread data in the receiver.

Figure 13-2 shows the frame configuration and the bit stream sequence for the UARTs. Figure 13-3 shows the timing for a transmission or a receive.

Figure 13-2 UARTs Frame

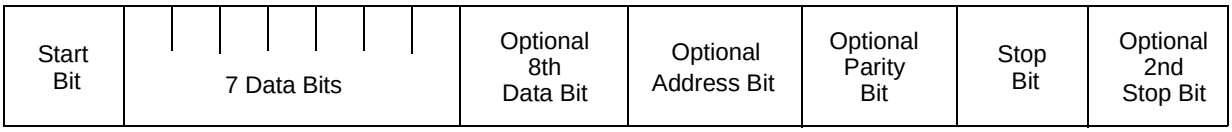
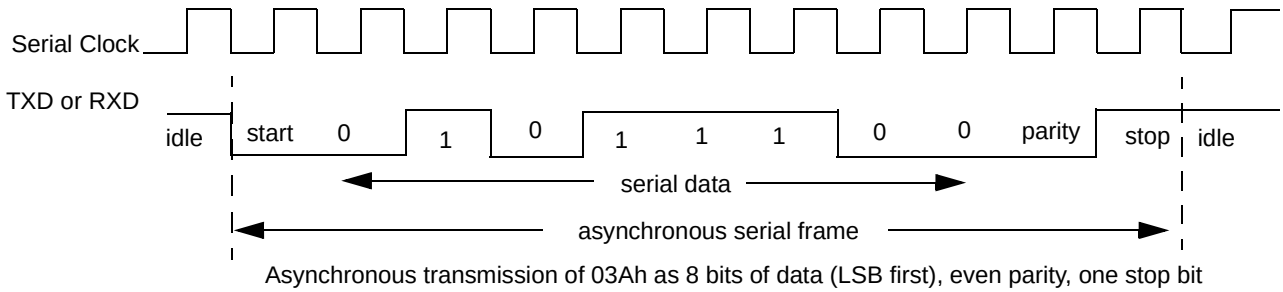


Figure 13-3 UARTs Timing



Asynchronous transmission of 03Ah as 8 bits of data (LSB first), even parity, one stop bit

13.5.2.1 Data Overflow

The UART registers contain two bits to handle receiver overrun errors: OER and OERIM. An *overrun error* occurs when the serial port overwrites valid, unread data in the receive data register or receive FIFO, resulting in a loss of data.

The OER bit in the (H)SPSTAT status register is set to 1 when the receiver has a data overrun error. When extended reads are enabled, the OER bit in the (H)SPRXD receive data register is set to 1, and then a 1 is written to the OER bit in the status register.

When FIFOs are enabled, the OER bit in the receive data register is passed through the FIFO with the last character of overrun data (i.e., the first data character after the overrun loss). The OERIM bit was added to provide an immediate interrupt. This bit bypasses the FIFO; OERIM is set immediately after the overflow occurs.

Both the OER and OERIM bits must be cleared by software.

13.5.2.2 Address Bits

When set, an *address bit* indicates that the present frame is a special frame. On the microcontroller, address bit generation and detection are supported in 7- or 8-data-bit frames. When the address bit is set, the other 7 or 8 bits of data in the frame are interpreted as a code; which type of code depends on the configuration. All transmissions that follow this address frame are directed as specified until another frame is received with the address bit enabled and with a different code.

One possible use for the code following the address bit (resulting in its name) is for the address of a slave peripheral device on a *multidrop* (also called *multipoint*) serial line, where one master device is talking to multiple slave devices. Although named an address bit, this bit actually behaves as an extended bit that may set an interrupt; the data bits that follow the bit do not need to be used as an address. Another possible use is for encoded discrete commands (e.g., sending a hang-up command to a modem). What the code is used for, and how, is determined by software.

To use the address bit in the microcontroller, the ABEN and D7 bits in the (H)SPCON0 register must be configured. In addition, the transmit AB bit must be set for a transmit; the received AB bit is set by hardware for a receive.

When the ABEN bit is set to 1, address bits are enabled. If the D7 bit is 0, serial frames contain a low start bit, eight data bits, an optional address bit, then one or two High stop bits (transmitted least significant bit first). If the D7 bit is set to 1, serial frames contain a low start bit, seven data bits, an optional address bit, and one or two stop bits.

13.5.2.2.1 Transmitting with Address Bit Set

When ABEN is set in a transmit, the transmit AB field of the (H)SPCON0 register is sent as the MSB bit after the 7 or 8 data bits in TDATA.

When extended writes are enabled (EXDWR = 1), the transmit AB bit in the (H)SPTXD register is used instead. Because this register also contains the data to be transmitted, this allows a single write to replace the two writes needed when extended writes are not enabled.

When extended writes are enabled, the (H)SPTXD register supports word-wide DMA transfers.

When extended writes are not enabled, the value of the transmit AB field is sampled during the transmission of the final data bit and is used to determine the value of the TXD signal for one bit time following the last data bit and before the transmission of the stop bit. The transmit AB field is cleared by the UARTs after reading its value. Because the transmit AB bit is not double buffered, software that intends to send a frame with the address bit set must wait until the transmitter is empty (TEMT=1) before setting the transmit AB bit and writing the data for the next frame into the transmit data register.

13.5.2.2.2 Receiving with Address Bit Set

In a receive, when ABEN is set in the HSPCON0 register, the most significant bit of received data can be read in the AB bit in the (H)SPSTAT register. The received AB bit can optionally generate an interrupt (if the AB bit is set in the HSPIMSK register and the RSIE bit is set in the HSPCON0 register). The received AB field must be cleared by software after reception of a frame for which the address bit was set.

When using extended reads (the EXDRD bit is set in the (H)SPCON1 register), the AB bit in the (H)SPRXD register is used instead. This allows reads from a single register, rather than requiring an additional read of the (H)SPSTAT register. In addition, the AB bit in the (H)SPRXD register is updated automatically; however, software must clear the AB bit in the (H)SPSTAT register.

13.5.2.3 Receive Status and Data

The EXDWR and EXDRD bits in the (H)SPCON1 register enable the programmer to use the upper bytes of the Transmit Data ((H)SPTXD) and Receive Data ((H)SPRXD) registers for transmitted data (the transmit and receive address bits and receive data status).

Receive status bits are set in the High-Speed Serial Port Status (HSPSTAT) register when the associated data byte is available to be read from the receive data register. When FIFOs are enabled, this occurs when the byte reaches the top of the FIFO. The read-only Receive Data Ready (RDR) bit in the (High-Speed) Serial Port Status (H)SPSTAT register reports when received data is available. This bit is cleared by hardware when there is no valid data waiting to be read from the (H)SPRXD register. If FIFOs are enabled, the FIFO is advanced and the next byte reaches the top of the FIFO when the previous data is read.

Under some conditions, such as when the DMA interface is being used, it may be useful for software to be able to examine the received character without affecting the status register or removing the data from the FIFO. The UARTs support this through the use of an alternate address for the receive data register (in the (H)SPRXDP register). This address allows software to peek at the value of the receive data register.

13.5.2.4 Extended Reads and Writes

Both serial ports on the microcontroller support extended reads of the receive data register and extended writes of the transmit data register.

When extended reads are enabled, by setting the EXDRD bit in the (H)SPCON1 register to 1, the serial port receive register supports 16-bit reads. The low byte of the register contains the normal receive data while the high byte contains status associated with the current frame, including the value of the address bit. See the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916, for a full description. Unlike the serial port status register, the high byte of the receive data register in extended reads reflects only the current frame. The accumulated status can be read from the status register normally and bits set in the status register must be cleared by software.

When extended writes are enabled, by setting the EXDWR bit in the (H)SPCON1 register to 1, the serial port transmit register supports 16-bit writes. Unlike extended reads which have a broad application, extended writes are useful only for applications that are using the address bit. When extended writes are enabled, the value of the address bit is written directly to the transmit register. This eliminates the need to write the address bit value to the (H)SPCON0 register.

Both extended reads and extended writes support word-wide DMA transfers. This allows full automation of the transmission of data streams containing address bits. If DMA is enabled with extended reads, status for each frame is stored along with the data for that

frame. At the end of the receipt of a sequence of frames, software can examine the value of the (H)SPSTAT register to determine if any significant status bits have been set. If the accumulated status does not reflect action required by software, the status bytes can be ignored, otherwise software can traverse the buffer searching for the status of interest and its associated data byte.

The microcontroller's general-purpose DMA channels support compression and decompression of data streams in part to support the extended read and write features of the serial ports. Status can be removed from a data stream through data compression using the DMA. For more information, see Chapter 8, "DMA Controller."

13.5.3 FIFOs (High-Speed UART Only)

The High-Speed UART provides a 32-byte FIFO for receive data and a 16-byte FIFO for transmit data. The use of the FIFOs can be enabled or disabled by software. The FIFOs can be operated in Polled or Interrupt mode, or they can be serviced using the general-purpose DMA channels. The High-Speed UART status register provides the RTHRSH and TTHRSH bits, which reflect the state of the receive and transmit FIFOs, respectively. When the RTHRSH bit is set, the receive FIFO has reached its threshold value (i.e., the receive FIFO is at least half full). When the THRSH bit is set, the transmit FIFO has reached its threshold value (i.e., the transmit FIFO is at least half empty). The HSPIMSK register contains bits that enable or disable interrupt generation based on the RTHRSH and TTHRSH bits. In this case, interrupt generation on the RDR (Receive Data Ready) and THRE (Transmit Holding Register Empty) bits should be disabled.

FIFOs are initialized to the empty condition on reset. For subsequent transfers, the transmit FIFO and the receive FIFO should be flushed by software by setting the TFLUSH and RFLUSH bits in the HSPCON1 register.

All transmit data is written to a single address, which is addressable as the transmit data register (HSPTXD) in both FIFO and non-FIFO modes. When the FIFO is not full, the High-Speed UART status register has the THRE bit set, indicating that data can be written to the FIFO without overwriting previously written data.

Receive data is read from a single address, which is addressable as the High-Speed UART Receive Data (HSPRXD) register in both FIFO and non-FIFO modes. When the FIFO is not empty, valid data can be read from HSPRXD; this is indicated by the Receive Data Ready (RDR) bit in the HSPSTAT status register. When the last bit of data has been removed from the FIFO, the RDR bit reads 0.

The status associated with each of the FIFO entries can be determined by reading the Serial Port Status (HSPSTAT) register before the associated data is read from the FIFO. When a byte is read from the FIFO, the next received character and its status move to the top of the FIFO and can be read from the receive data and status registers. The status must be read before the data is read. Alternatively, extended reads can be enabled. Extended reads allow status to be read with data as it moves to the top of the FIFO. Reading the status using extended reads differs from what is shown in the serial port status register in that it reflects the current frame only. The serial port status register functions normally during extended reads and continues to reflect accumulated status and to generate interrupts based on that status as configured.

13.5.3.1 Transmit FIFO

The transmit FIFO provides for up to 16 bytes of transmit data plus the value of the associated address bit, if applicable.

When both the FIFO and a transmit have been enabled (with the TFEN and TMODE bits), hardware immediately checks the Transmit FIFO Threshold Reached (TTHRSH) bit in the status register. The transmit FIFO threshold value is set to half-empty (FIFO contains eight bytes of data) and is not programmable. The High-Speed UART can be programmed to generate a maskable interrupt when the transmit FIFO reaches the threshold value. Software must clear the TTHRSH bit, but hardware can set it again immediately if the FIFO contains less than eight entries.

FIFO underflow occurs when the transmit FIFO becomes completely empty. The High-Speed UART can be programmed with the TEMT bit to generate an interrupt on FIFO underflow.

13.5.3.2 Receive FIFO

The receive FIFO provides for up to 32 bytes of receive data along with status associated for each byte, including special-character matching, framing and parity error flags, and the value of the address bit, if applicable.

When both the receiver and the receive FIFO have been enabled, through the RMODE bit for the receiver and the RFEN bit for the receive FIFO, the serial port hardware immediately checks for a receive FIFO threshold reached condition. The receive FIFO threshold value is not programmable and is set at half full or 16 bytes of data present in the FIFO. The High-Speed UART can be programmed to generate a maskable interrupt when the receive FIFO reaches the threshold value. Software must clear the RTHRSH bit, but hardware sets it again immediately if the FIFO contains 17 or more entries.

As data moves to the top of the FIFO, the associated status is placed in the Serial Port Status (HSPSTAT) register. Status bits are set by the hardware and must be cleared by software. The serial port can be configured to generate an interrupt based on serial port status. Each status bit is individually maskable. If an interrupting status condition is detected in the serial port, DMA requests from the receiver are disabled. This allows the interrupt service routine to read the data from the receiver (or to peek at the data through the HSPRXDP register) and take appropriate action. Enabling extended reads allows the status to be read in a word-wide read from the Serial Port Receive Data (HSPRXD) register. The status data in the upper byte of an extended read reflects only the current frame. However, the HSPSTAT register continues to be updated normally and set bits must be cleared by software.

The IDLED bit in the HSPSTAT register indicates instances where the threshold is never reached because less than 16 bytes of data were sent. The IDLED bit is set (and can be used to generate an interrupt with the HSPIMSK register) when the receive data line has been idle for 40 bit times and receive data is available. This bit must be cleared by software.

FIFO overflow occurs when the receive FIFO is completely full and another character is received, resulting in the loss of data. In a FIFO overflow condition, the last location of the FIFO is overwritten with the last byte received. The High-Speed UART can be programmed to generate a maskable interrupt on FIFO overflow with the OERIM bit.

13.5.3.3 Using the FIFOs in Polled, Interrupt, or DMA Mode

The High-Speed UART FIFOs can be used in Polled, Interrupt, or DMA modes.

Interrupt and DMA modes are described in “Interrupt Sources” on page 13-19 and “Interface to General-Purpose DMA Channels” on page 13-21. When in Polled mode, the High-Speed UART behavior is similar to the non-FIFO mode.

In Polled mode, software reads the received data by reading the HSPRXD register. The HSPSTAT register is updated with the status of the next frame after each read of the receive

data register. The new status is ORed with the previous status, possibly accumulating status over multiple frames. For this reason, the status register must be read before the receive data register to ensure that the status being read is for the current frame. Set status bits must be cleared by software. When extended reads are enabled, the high byte of the HSPRXD register contains the status associated with the current frame; however, status continues to accumulate in the HSPSTAT register. The RDR bit is set when data is available in the receive FIFO (the value of this bit can be read from the HSPSTAT register or from the high-byte of an extended read). When the RDR bit is set, valid data is present in the FIFO.

If receive status interrupts are enabled, an interrupt can be generated at the time an error is detected. The interrupt service routine must read data out of the FIFO until the data which generated the interrupt reaches the top of the FIFO. At this point, the status register reports the error condition.

Data to be transmitted is written to the Transmit Data register as in non-FIFO mode. However, the transmit status reflects the disposition of the FIFO. When the FIFO is not full, the Transmit Holding Register Empty (THRE) bit is set, indicating that the transmitter can accept more data. When the FIFO is completely full, the THRE bit is 0. When the transmit FIFO and the transmit shift register are completely empty, the Transmitter Empty (TEMT) bit is set. At this point, the transmitter or FIFO can be disabled without loss of data.

13.5.4 CTS/RTR Hardware Flow Control

The microcontroller supports CTS/RTR hardware flow control. Each UART port is provided with two data signals (TXD_U and RXD_U for the UART, and TXD_HU and RXD_HU for the High-Speed UART) and two flow control signals ($\overline{\text{CTS\_U}}$ and $\overline{\text{RTR\_U}}$ for the UART, and $\overline{\text{CTS\_HU}}$ and $\overline{\text{RTR\_HU}}$ for the High-Speed UART). Hardware flow control is enabled when the FC bit in the (H)SPCON0 register is set to 1.

In the CTS/RTR protocol, the receiver asserts clear-to-send (CTS) whenever there is room in the receiver for more data. The transmitting device should sample CTS before beginning transmission of each frame. CTS is deasserted when the start bit is detected for the last frame that can be read without data loss. When FIFOs are disabled, CTS is deasserted after the start bit for each frame is detected and remains deasserted until the data is read from the receive data register. When the receive FIFO is enabled, CTS is deasserted after the start bit is received for the last frame that will fit in the FIFO.

The transmitter samples ready-to-receive (RTR) before transmitting the start bit of each frame. The RTR signal is not sampled during frame transmission. This allows the receiving device to deassert RTR any time before the end of the stop bit. The transmitter does not begin transmitting the start bit for the next frame while RTR is deasserted.

The use of hardware flow control can eliminate the possibility of overrun errors—data loss due to reception of new data before the last received data has been read. However, there can be an adverse effect on data throughput. For example, in the case where there is no receive FIFO, transmission of a second data frame cannot begin until the previous frame's data has been read. Without hardware flow control, transmission of the next frame may begin immediately, providing the receiver with one frame time to read the previous frame's data without data loss. Use of FIFOs or DMA reduces the impact of hardware flow control on data throughput.

In multidrop systems, typically using the address bit feature of the microcontroller's serial ports, hardware flow control should not be enabled, or must be restricted to a single pair of active UARTs.

Figure 13-4 illustrates the behavior of the $\overline{\text{RTR\_U}}$ signal. Figure 13-5 illustrates the behavior of the $\overline{\text{RTR\_U}}$ signal with the FIFO. Note that the $\overline{\text{RTR\_U}}$ signal is deasserted as soon as the serial port begins receiving a character and is reasserted when the data is read from the receive data register.

Figure 13-4 $\overline{\text{RTR\_U}}$ Signal Behavior

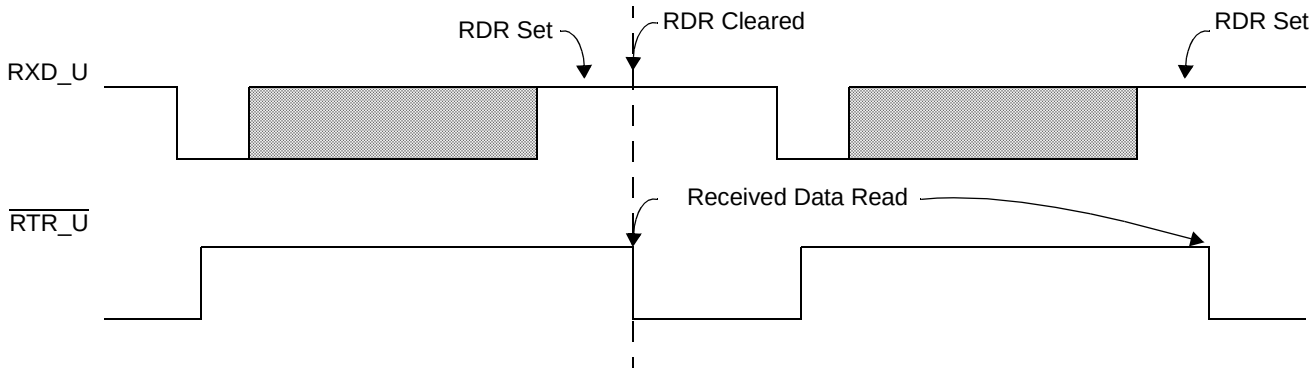
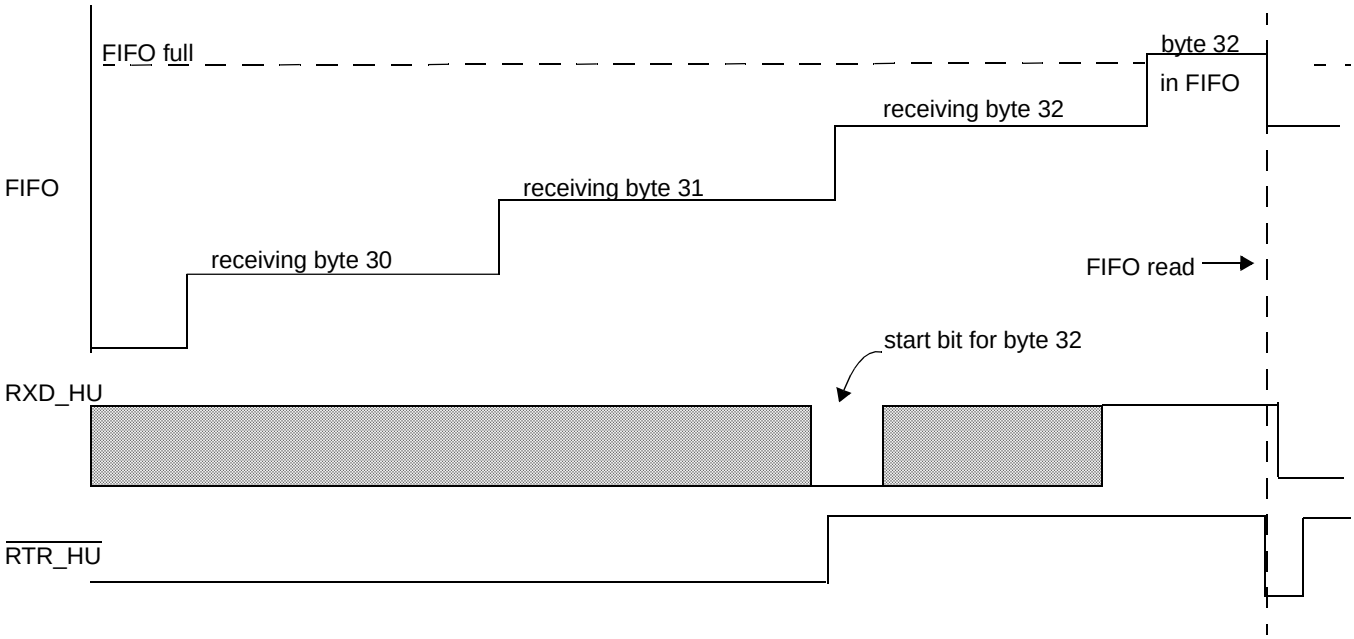


Figure 13-5 $\overline{\text{RTR\_HU}}$ Signal Behavior with Receive FIFOs



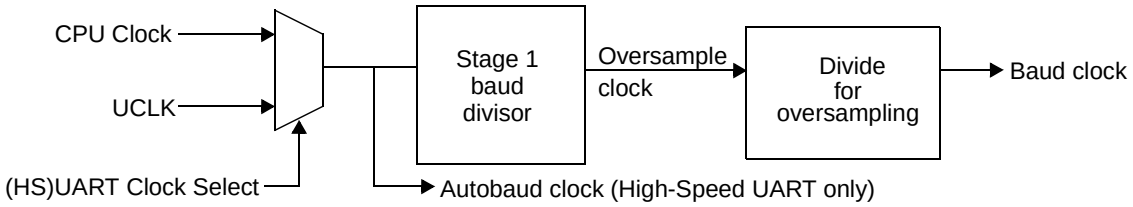
13.5.5 Clock Sources and Baud Rate

The UARTs have two possible clock sources, the processor clock or the UCLK input signal. The possible clock configurations are shown graphically in Figure 13-6. The XTRN bit in the (H)SPCON1 register selects the clock source.

The baud clock is generated by dividing the clock source by the value of the Baud Rate Divisor ((H)SPBDV) register. In addition, the High-Speed UART supports automatic baud rate detection. The UARTs select the clock from either the CPU clock or the UCLK signal, independent of any other settings.

CC **CU** In the Am186CC and Am186CU microcontrollers, the USBSOF signal must not be enabled at the same time as UCLK.

Figure 13-6 UARTs Clock



13.5.5.1 Programming the Baud Rate

The formula for determining the baud divisor register value is:

BAUDDIV = (clock frequency / (16 • baud rate))

The maximum baud rate is 1/16th of the CPU clock. The processor clock frequency is the maximum clock frequency supported on the UCLK input. Table 13-3 shows the baud divisors for various baud rates and clock speeds. Note that the baud divisor value (BDV) is shown in both decimal and hexadecimal.

Table 13-3 Baud Rate Table for UARTs

	Serial Port Clock Frequency (Processor Frequency or UCLK Frequency)											
	24 MHz		25 MHz		40 MHz		44.2 MHz		48 MHz		50 MHz	
Baud Rate	Divisor	% Error	Divisor	% Error	Divisor	% Error	Divisor	% Error	Divisor	% Error	Divisor	% Error
300	5000d 1388h	0	5208d 1458h	0	8333d 208Dh	0	9208d 23F8h	0	10000d 2710h	0	10417d 28B1h	0
600	2500d 09C4h	0	2604d 0A2Ch	0	4167d 1047h	0	4604d 11FCh	0	5000d 1388h	0	5208d 1458h	0
1200	1250d 04E2h	0	1302d 0516h	0	2083d 0823h	0	2302d 08FEh	0	2500d 09C4h	0	2604d 0A2Ch	0
2400	625d 0271h	0	651d 028Bh	0	1042d 0412h	0	1151d 047Fh	0	1250d 04E2h	0	1302d 0516h	0
9600	156d 9Ch	0.2	163d A3h	-0.1	260d 0104h	0.2	288d 0120h	-0.1	313d 0139h	-0.2	326d 0146h	-0.1
19200	78d 4Eh	0.2	81d 51h	0.5	130d 82h	0.2	144d 90h	-0.1	156d 9Ch	0.2	163d A3h	-0.1
38400	39d 27h	0.2	41d 29h	-0.8	65d 41h	0.2	72d 48h	-0.1	78d 4Eh	0.2	81d 51h	0.5
57600	26d 1Ah	0.2	27d 1Bh	0.5	43d 2Bh	0.9	48d 30h	-0.1	52d 34h	0.2	54d 36h	0.5
115200	13d 0Dh	0.2	14d 0Eh	-3.2	22d 16h	-1.4	24d 18h	-0.1	26d 1Ah	0.2	27d 1Bh	0.5
230400	7d 07h	-7.5	7d 07h	-3.2	11d 0Bh	-1.4	12d 0Ch	-0.1	13d 0Dh	0.2	14d 0Eh	-3.2
460800	3d 03h	7.8	3d 03h	11.5	5d 05h	7.8	6d 06h	-0.1	7d 07h	-7.5	7d 07h	-3.2

13.5.5.2 Receiver Bit Sampling

Whenever the receiver is enabled and is not in-frame (i.e., no data frame is currently being received), it remains in a Search-for-start-bit mode. In this mode, the receiver looks for a High to Low transition of the RXD input. This sampling is done based on the divided Baud Clock. When a transition is detected, the receiver waits one-half bit-time, until the expected midpoint of the start bit, then resamples the RXD input. If RXD is sampled Low, a valid start bit has been detected and the receiver enters In-Frame mode. If RXD is sampled High, it is assumed that the initial transition was a glitch and the receiver remains in Search-For-Start-Bit mode. While in In-Frame mode, the receiver samples each bit one time at the expected midpoint for that bit.

When the High-Speed UART is configured in Autobaud mode, the High-Speed UART uses the undivided autobaud clock during the search for the High-to-Low transition while in Search-For-Start-Bit mode. When the transition is detected, the receiver enters Start-Bit-Calibration mode. In this mode, the undivided autobaud clock is used to time the duration from the detection of the initial falling edge to the next rising edge of RXD. The number of autobaud clocks is divided by 16 and written to the baud divisor register. If the autobaud registers (HSPAB3–HSPAB0) are active, the value written to the baud divisor register may be provided by one of these registers. In either case, the receiver enters normal In-Frame mode, as with frames that do not use autobaud, and the new value of the baud divisor is used to generate the baud clock for the next and subsequent bits.

13.5.5.3 Detecting the Baud Rate Automatically (High-Speed UART Only)

The High-Speed UART supports automatic baud rate detection (autobaud) by setting the ABAUD bit in the HSPCON1 register to 1. When in autobaud mode, the detection of a transmission from High to Low on the receive data signal causes the baud rate timer to begin counting. A transition from Low to High stops the baud rate timer and causes the serial port to exit autobaud mode. The baud rate timer runs at the processor clock or at the external serial port clock rate if enabled (via the XTRN bit in the HSPCON1 register). The closest possible baud-rate divisor is determined (rounded) and automatically programmed into the Baud Rate Divisor (HSPBDV) register. The data is reported in the Receive Data (HSPRXD) register and can be read by software.

For automatic baud rate detection to function correctly, the first data sent must be a Low bit (the start bit) followed by a High bit (the least significant bit of the data). This allows the use of the Hayes AT interface, which requires the initial character to be an ascii character a. Although a framing error *may* result, software should check the value of the initial character to verify that it matches the expected character.

There is no automatic determination of parity use or sense, word length, or number of stop bits.

Figure 13-7 shows the possible error ranges that exist over 167 implied baud divisors. The worst-case % error occurs when the baud rate expected requires a real number baud divisor that lies halfway between two integer baud divisors and is rounded up. Assuming 0.3% error is an acceptable error per bit, autobaud mode that selects a divisor above 166 should always yield acceptable results. For baud rates that require a baud divisor of less than 166, the end user must determine whether the High-Speed UART autobaud capability is appropriate for the application. Figure 13-8 illustrates autobaud detection under various frequencies.

Figure 13-7 Worst Case % Error Per Bit vs. Baud Divisor Without Autobaud Enhancement

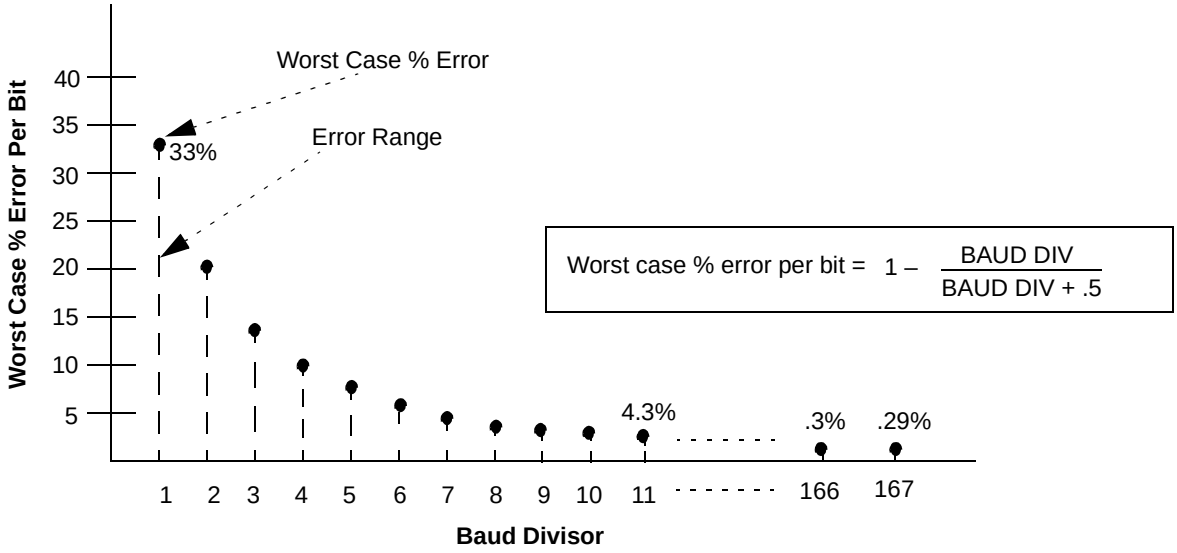
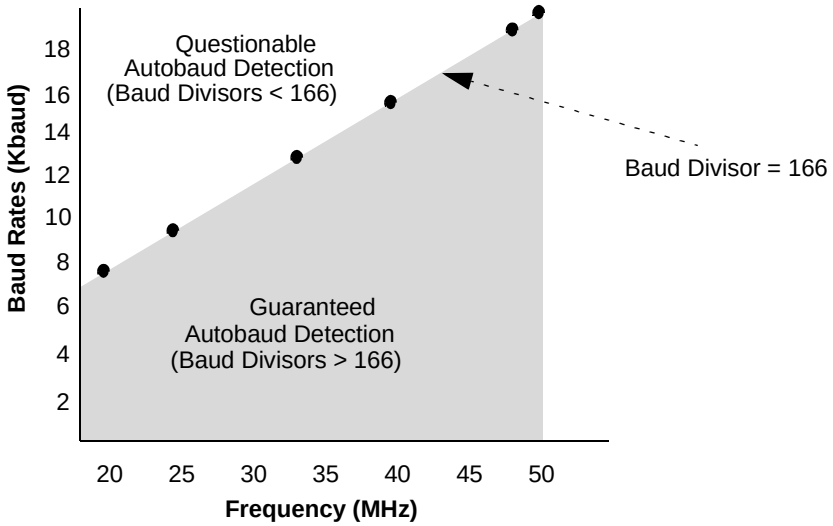


Figure 13-8 Detectable Baud Ranges for Various Frequencies



The microcontroller also offers an enhancement to autobaud detection. Start bit width distortion can result in calculation errors. In instances where there is some system knowledge, baud divisor and threshold values can be programmed to allow for a 25–30% distortion in the width of the start bit.This method greatly increases the probability that the correct valid baud rate divisor is selected for higher baud rates, which are at highest risk.

The microcontroller method makes use of the fact that, in most cases, valid baud divisor values used in a particular application are separated by several integers. Four High-Speed Serial Port Autobaud registers are provided for this enhancement: HSPAB0, HSPAB1, HSPAB2 and HSPAB3. Each register contains a divisor value and a threshold value. The HSPAB3 register must contain the largest programmed divisor value and threshold value, then HSPAB2, etc. When using fewer than four valid divisor values, software must clear the unused HSPABx registers or leave them at their default values (00h).

When the registers have been programmed, the High-Speed UART compares the autobaud calculated baud rate divisor to the threshold values and selects one of the programmed valid baud rate divisors to use in subsequent data transfers. A calculated value less than

or equal to threshold 1 and greater than threshold 0 selects the divisor 1 value, and so on. A value greater than threshold 3 uses the calculated divisor value. If the registers are not programmed (are in reset state), the High-Speed UART uses the autobaud calculated baud divisor value. Figure 13-9 illustrates this concept.

Table 13-4 shows two examples of using the autobaud registers to enhance autobaud detection.

Figure 13-9 Autobaud Enhancement

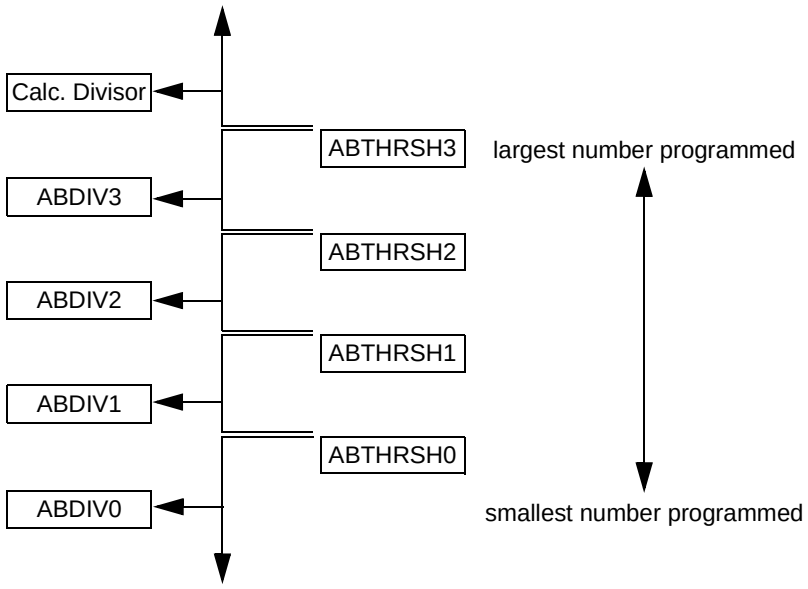


Table 13-4 Examples of Autobaud Enhancement

Example	Register	ABDIV	ABTHRSH	Baud Rate	Range	
					# of Clocks in Start Bit	Baud Divisors
A) @24MHz	HSPAB3	9Ch	A0h	9600	810h–A0Fh	81h–A0h
	HSPAB2	4Eh	80h	19200	350h–80Fh	35h–80h
	HSPAB1	27h	35h	38400	190h–34Fh	19h–34h
	HSPAB0	0Dh	18h	115200	010h–18Fh	01h–18h
B) @48MHz	HSPAB3	34h	32h	57600	310h–32Fh	31h–32h
	HSPAB2	1Ah	30h	115200	190h–30Fh	19h–30h
	HSPAB1	0Dh	18h	230400	010h–18Fh	01h–18h
	HSPAB0	00h	00h			

In autobaud mode, the receiver determines a value for the baud divisor register based on the sampled duration of the start bit. The start bit duration in clocks is converted to a value to be written to the baud divisor register by dividing by 16, as shown in “Programming the Baud Rate” on page 13-15.

The configuration in example A does not support a baud rate of 57600 since this baud rate is not represented in the table and requires a baud divisor less than that programmed in HSPAB3. In general, any baud divisor below the maximum divisor programmed in the

HSPABx registers, which is not programmed in the ABDIV field for one of the HSPABx registers, is unattainable for that system.

In example B, the HSPAB0 register is not being used. The value of the ABDIV field for HSPAB3 is greater than the ABTHRSH field for that register. Although this is not the case for most systems, it is possible for the replacement divisor to fall outside of the range of sampled baud rates that generate that divisor.

13.5.6 Interrupt Sources

All UART and High-Speed UART interrupt sources require two interrupt enable bits to be set before that source is enabled to generate interrupts.

The first level consists of three main interrupt enable bits in the (H)SPCON0 control register. The Receive Interrupt Enable (RXIE) bit enables interrupts that indicate receive data is available (the RDR bit in the status register is 1). The Receive Status Interrupt Enable (RSIE) bit enables interrupts on the condition or status of the received data. The Transmit Interrupt Enable (TXIE) bit enables interrupts based on the status of the transmit data (whether the TEMT bit in the status register is 0 or 1).

The (H)SPIMSK register contains the second-level interrupt bits. Even if these bits are set to 1, interrupts are disabled if the corresponding first-level enable bit is not also set to 1.

Table 13-5 shows the interrupt sources for the UARTs. All first-level enable bits default to Off (0). The defaults for the second-level bits vary and are listed.

Note that when a receive status bit has generated an interrupt condition and extended reads are disabled, receive DMA requests are inhibited.

Table 13-5 UARTs Interrupt Sources

Interrupt Enable[1]	**(H)SPCON0 1st-Level Enable Bit (Default)**	**(H)SPIMSK 2nd-Level Enable Bit (Default)**
Receive data ready	RXIE (Off)	RDR (On)
Receive FIFO threshold reached	RXIE (Off)	RTHRSH (Off)
Overrun error on receive FIFO	RSIE (Off)	OERIM (Off)
Transmit holding register empty	TXIE (Off)	THRE (On)
Transmit FIFO threshold reached	TXIE (Off)	TTHRSH (Off)
Transmitter empty, transmit FIFO empty on High-Speed UART	TXIE (Off)	TEMT (Off)
Parity error	RSIE (Off)	PER (On)
Overrun error	RSIE (Off)	OER (On)
Framing error	RSIE (Off)	FER (On)
Break detected	RSIE (Off)	BRK (On)
Address bit set on receive	RSIE (Off)	AB (Off)
Character match on receive	RSIE (Off)	MATCH (Off)
Receive line idle detected	RXIE (Off)	IDLED (Off)
Receive line idle	RSIE (Off)	IDLE (Off)

Notes:

1. When the FIFOs are in use, High-Speed UART RDR and THRE should be disabled from generating interrupts. When using the FIFOs, High-Speed UART RTHRSH is the logical replacement for RDR, and TTHRSH is the logical replacement for THRE.

13.5.7 Break Detection and Generation

The UARTs support detection of break characters. A *break* is defined as a constant Low signal on the receive data line for one frame time or greater. This is reported as a zero character with the framing error (FER) and break (BRK) status bits set in the (H)SPSTAT register. (A *framing error* is the detection of a Low signal during the stop bit time.)

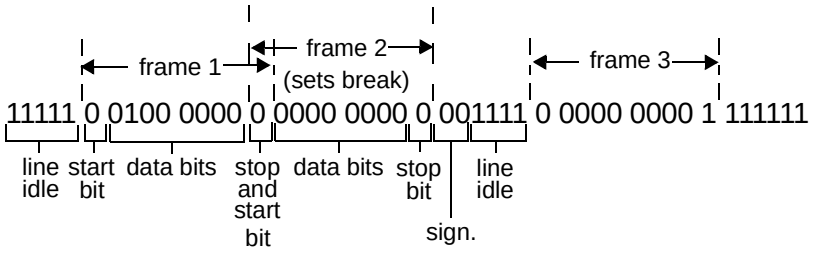
When a break is being transmitted, it only affects the output on the TXD signal; it does not affect the timing of the transmit section of the serial port. In other words, the transmitter can be used to time the break by setting the BRK bit when the transmitter is empty (TEMT=1), writing the transmit register with data, then waiting until the TEMT bit is set again before resetting the BRK bit.

Note: *The transmitter can only be used to time the break if hardware flow control is disabled. If flow control is enabled, setting the BRK bit will still force the TXD line Low, but the receiving device may deassert the CTS input, inhibiting the clocking out of the character in the transmit data register.*

The microcontroller also supports timing of idle frames (TXD signal High) through use of a software configurable bit (BRKVAL in the HSPCON1 register), which controls whether to hold the TXD line High or Low when a break is being transmitted.

Receive status information is reported at the end of the frame time. Figure 13-10 provides an example data stream (assuming no parity, 8 data bits, one stop bit).

Figure 13-10 Break Character Example



In this stream, the leading 1's are assumed to be an idle indication on the line. The first Low bit is interpreted as the start bit of a frame, resulting in the first frame consisting of the stream "0 0100 0000 0", including start and stop bits. This is reported as a 02h character with a framing error. The stop bit for the first frame also acts as the start bit for the next frame, which is "0 0000 0000 0". This is reported as a 00h character with a framing error and a break character (the BRK bit is set to 1). Low bits following a break are ignored until the line returns to the High state. Therefore, the next frame would be "0 0000 0000 1", which is a 00h character with no error status.

13.5.8 Receive Special-Character Matching (High-Speed UART Only)

The High-Speed UART provides a method of generating interrupts on special characters. Up to six special characters can be matched. Special-character matching is enabled by the MEN bit in the HSPCON1 register.

The special characters are written by software into three 16-bit character match registers (HSPM0, HSPM1, and HSPM2). When character matching is enabled with the MEN bit, the incoming character is compared against all six special characters. Applications using fewer than six special characters should program the extra compare registers with duplicates of valid special characters, as the default 0 is considered a valid character.

If address bits are used, three bits (MAB0, MAB1, and MAB2) in the HSPCON1 register must also be used, one for each special-character register. When the MAB bit and the address enable (ABEN) bit are both set, a received character must have the address bit set in order to match characters in the corresponding character match register. Both the character and the address bit must match in order for a special character to be detected. If address bit detection and generation is not enabled, the value of the address-match bit is ignored during the comparison.

For 7-bit character matching, the High bit of each byte in the character match registers should be cleared to zero. If address matching is enabled for 7-bit characters, the three match-address bits in the control register are used to determine the match, not the High bit of each byte in the character match register.

When a special character is detected, the Address Match Detected (MATCH) bit is set in the status register. A maskable interrupt can be generated on this condition, using the HSPIMSK register.

Special-character matching has several possible applications, including the following:

- Use special character matching to implement software flow control using the XON/XOFF protocol.
- In a multidrop system, use special character matching to determine if a device's address has been broadcast.
- Once the address has been enabled in a multidrop system and a data stream is being received, use special-character matching to detect the flag that signals the end of a data packet.

13.5.9 Interface to General-Purpose DMA Channels

The general-purpose DMA channels can access either UART's receivers and transmitters. The receiver generates a DMA request when the Receive Data Ready (RDR) bit is set. The transmitter generates a DMA request when the Transmit Holding register (THRE) bit is set. This behavior is independent of the use of the High-Speed UART FIFOs and the FIFO thresholds.

When a receive status condition that is configured to generate an interrupt request is detected, DMA requests from the receiver are disabled. This leaves the data with the interrupting condition in the (H)SPRXD register for easy examination by software in the interrupt service routine. When the receive FIFO is enabled on the High-Speed UART, the interrupt request is generated, and the DMA requests are suspended when the data reaches the top of the FIFO. DMA requests are resumed when the interrupting condition is cleared by software.

When extended reads are enabled, DMA requests from the receiver are not disabled when an interrupting status condition is detected. In this situation, the status is maintained with

data in memory so that the association of status and data is not lost. This behavior is not affected by enabling or disabling the receive FIFO on the High-Speed UART.

Unlike the other status interrupts that move through the receive FIFO with their associated data, the OERIM bit provides immediate notification of an overrun error condition. Software can determine where the overrun occurred since the OER status bit travels through the FIFO with the associated data. The OERIM error bypasses the FIFO and does an immediate interrupt.

When extended writes are enabled, the value of the address bit is written to the high byte of the (H)SPTXD register. When extended writes are not enabled, the value of the address bit is taken from the value of the transmit Address Bit (AB) field in the serial port control register at the time that the data is written to the transmit holding register. This bit is cleared after each transfer of the data out of the holding register into either the transmit shift register or the High-Speed UART FIFO.

When extended reads are enabled, the value of the address bit is read from the high byte of the (H)SPRXD register, and also sets the AB bit in the status register. When extended reads are not enabled, the value of the received address bit is placed in the received Address Bit (AB) field of the (H)SPSTAT status register and must be cleared by software. This means that applications needing to send or receive a string of characters with the address bit cleared can use DMA to transfer the data to or from the serial port. Applications that require the address bit set, or a mixture of the bit set and cleared, may use the DMA but must take an interrupt each time the address bit is set. This is true regardless of the use of the FIFO.

For information about the use of the CTS/RTR protocol with DMA, see “CTS/RTR Hardware Flow Control” on page 13-13.

For more information about using the UARTs and DMA, see Chapter 8, “DMA Controller.”

13.5.10 Hardware-Related Considerations

The signals for the UART and flow-control for the High-Speed UART are multiplexed with HDLC Channel D. For more information, see Table 13-1 on page 13-3.

13.5.11 Software-Related Considerations

- Always program the configuration registers *before* setting the TMODE or RMODE bit to 1.
- The most efficient data transfer operation (least software intervention, highest average data transfer rate, and least opportunity for FIFO overrun) is when using FIFOs, DMA, and CTS/RTR flow control.
- In a multidrop system, hardware flow control must be enabled for only a single pair of devices at any one time.
- Always flush FIFOs before new data transfers.
- The UARTs are multiplexed with HDLC. The Interface 4 Select (ITF4) bits in the System Configuration (SYSCON) register must be configured for the UART interface.

13.5.12 Comparison to Other Devices

The UARTs are similar to those of the other Am186 family microcontrollers and are most closely related to those of the Am186ES and Am186ED microcontrollers. However, the functionality provided by the serial port modes of those devices have been replaced by individual enables that allow for more flexibility on the UARTs and the configuration of the DMA interface has been modified. In addition, both the UART and the High-Speed UART of the microcontroller provide significant enhancements over previous UARTs, including extended reads and writes, support for address bits on 7-data bit frames, and two stop bits. The High-Speed UART's enhancements additionally include autobaud detection, receive and transmit FIFOs, and special character matching.

13.6 INITIALIZATION

On both external and internal reset, the following occurs:

- All register bits are cleared to 0 except for the following second-level interrupt enable bits in the (H)SPIMSK registers: BRK, RDR, THRE, FER, OER and PER. This provides compatibility with the Am186ES and Am186ED microcontroller serial ports.
- All UART and High-Speed UART signals, except for TXD_HU, default to that signal's PIO function. See Table 13-1 on page 13-3.
- The ITF4 bit field in the SYSCON register is cleared, which defaults external interface D to HDLC with flow control.

CHAPTER

14 SYNCHRONOUS SERIAL PORT (SSI)

14.1 OVERVIEW

The Am186CC/CH/CU microcontrollers each include one synchronous serial port, which uses the SSI to provide a half-duplex, bidirectional communications interface between the microcontroller and other system components (i.e., integrated circuits). This interface is typically used by the microcontroller to monitor the status of other system devices or to configure these devices under software control. In a communications application, these devices could be system components such as transceivers or audio coder-decoders (codecs). The SSI supports data transfer speeds of up to 25 Mbit/s.

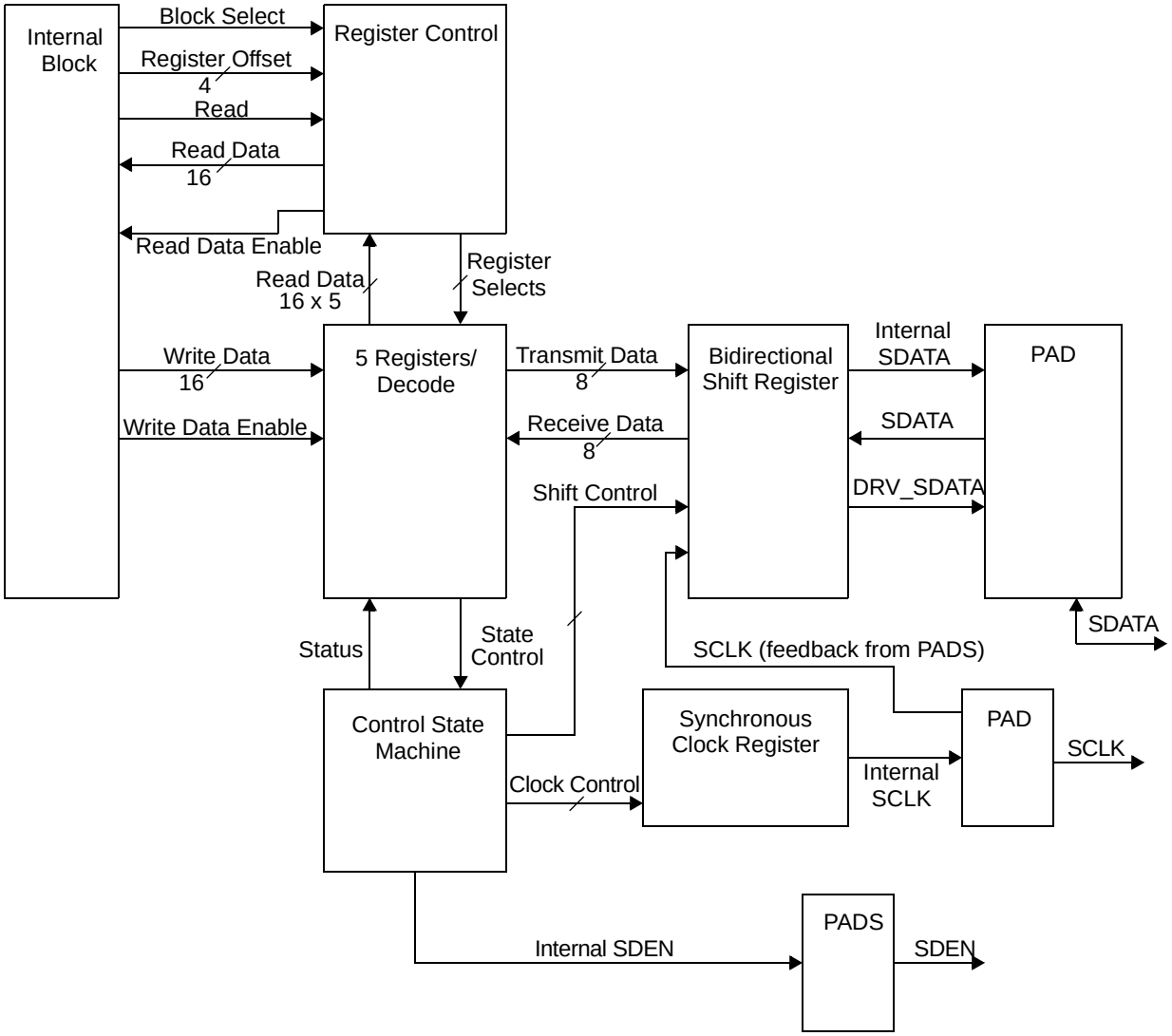
The SSI provides the following features:

- Three I/O signals: SCLK, SDATA, and SDEN, multiplexed with PIOs
- Programmable data order: Normal (least-significant bit first) or Reverse (most-significant-bit first)
- Programmable SSI clock divisor: Divides the CPU clock from 2 to 256 in power of 2 increments
- Programmable polarity of the SCLK and SDEN signals
- Bidirectional transmit/receive shift register

14.2 BLOCK DIAGRAM

Figure 14-1 shows the block diagram for the SSI.

Figure 14-1 SSI Block Diagram



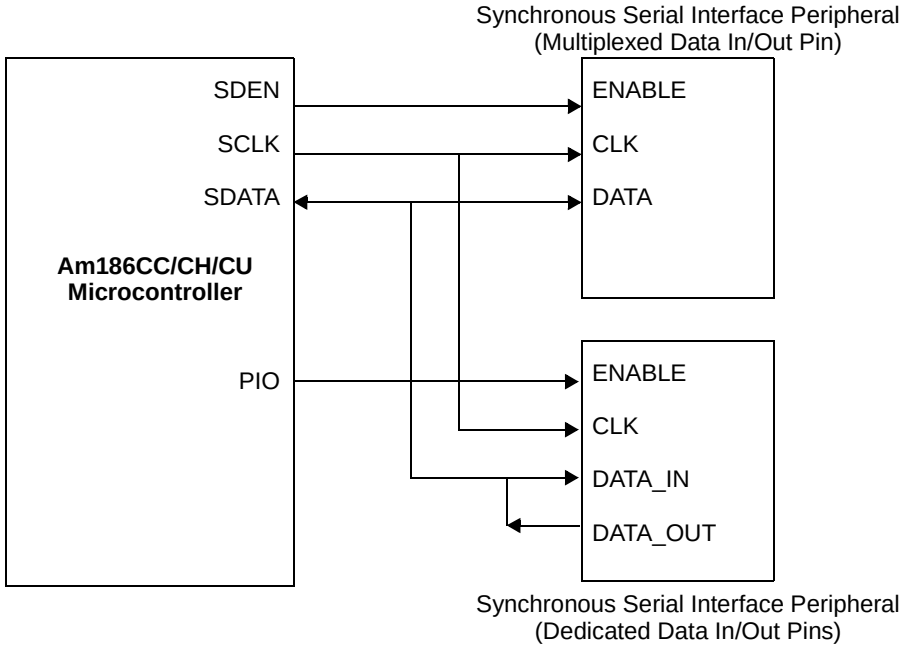
14.3 SYSTEM DESIGN

Table 14-1 lists the SSI signals that are multiplexed with other microcontroller functions. Pinstraps are sampled only at external reset and do not affect the pin's other functions, so they are not shown in this table. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin. Figure 14-2 illustrates a sample SSI application for the Am186CC/CH/CU microcontrollers.

Table 14-1 SSI Multiplexed Signals

Signal	Function	Multiplexed Signal(s)	Default Signal
SDEN	Serial data enable	PIO10	PIO10
SCLK	Serial clock	PIO11	PIO11
SDATA	Serial data	PIO12	PIO12

Figure 14-2 Synchronous Serial Interface System Application Example



14.4 REGISTERS

The registers listed in Table 14-2 program the SSI. Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 14-2 SSI Register Summary

Offset	Register Mnemonic	Register Name	Description
2F0h	SSSTAT	SSI Mode/Status	Defines SSI modes of operation and reports the port status.
2F2h	SSCON	SSI Control	Enables SSI and programs the data order, clock divisor, and polarity.
2F4h	SSTXD1	SSI Transmit 1	Contain the data to be transmitted. A write to either register initiates a transmit transaction if either of the DE bits in SSCON is set to 1.
2F6h	SSTXD0	SSI Transmit 0	
2F8h	SSRXD	SSI Receive	Contains the data received over the SSI. A read from this register initiates a receive transaction if either of the DE bits in SSCON is set to 1.

14.5 OPERATION

14.5.1 Usage

Note: *Before using the SSI port, ensure multiplexed pins are configured to reflect the use of SSI and not other functionality (see Table 14-1 on page 14-2).*

1. Set the ENHCTL bit in the SSSTAT register to 1 so that all bits in the SSCON register are operational (unless Am186EM-backwards compatibility is required).
2. If using the SDEN signal, initialize the SSI port with the SSCON register: clock polarity (CLKP bit), device polarity (DENP bit), transmit bit order—LSB or MSB first (MSBF bit), and the CPU clock divisor (CLKEXP bit field).

   If not using SDEN (but using a PIO output as an external enable), use the PIO Set and PIO Clear registers to provide the external signal while using the DE1 bit of the SSCON register to provide internal enable. Be sure the corresponding mask bit in the SHMASK register is set to disable the interrupt.
3. Enable transmit or receive by setting the DE0 or DE1 bit in the SSCON register to 1.
4. Write data with the SSTXD1/SSTXD0 registers or read data with the SSRXD register (this sets the Port Busy (PB) bit in the SSSTAT register to 1).
5. Wait for the DR/DT bit in the SSSTAT register to go to 0 to indicate the transmit or receive has completed.
6. Disable the transmit or receive by clearing the DE0 or DE1 bit in the SSCON register to 0.

14.5.2 Master/Slave Configuration

Unlike the asynchronous serial ports described in Chapter 13, “Asynchronous Serial Ports (UARTs),” the SSI port operates in a master/slave configuration, where the microcontroller operates as the master port. All other devices that communicate with the microcontroller through this interface are slave devices. The master initiates a transaction by transmitting a single byte. This byte tells the slave device whether the transaction is a read or a write and contains the device address. The microcontroller always drives the interface clock when an active communication transaction is present on the interface. Slave devices cannot drive this clock. Because PIOs can be used as external device enables, the microcontroller can support a number of peripheral devices.

14.5.3 Signal Interface

The SSI port consists of three I/O signals: data (SDATA), clock (SCLK), and enable (SDEN). The three SSI signals are multiplexed with three programmable I/O signals (PIO12–PIO10). These pins are PIOs by default, and can be individually reconfigured as SSI pins with the PIO Mode and PIO Direction registers.

14.5.3.1 SCLK

The SCLK output synchronizes transmit and receive operations between the master (microcontroller) and slave (peripheral). Based on the selected polarity of the SCLK signal in the SSCON register, SCLK is at a constant High (default) or Low (when inverted polarity is selected) level when a transmit or receive operation is *not* active on the interface. SCLK derives from the internal CPU clock divided by 2, 4, 8, 16, 32, 64, 128, or 256. Software specifies the divisor with the CLKEXP bit field of the SSCON register. When a transfer is started, the microcontroller toggles this clock for the entire transaction. Each individual transaction transfers eight data bits.

The clock edge on which data is transmitted and received is programmable with the CLKP bit in the SSCON register. In the default condition, data is transmitted on the SDATA pin on

the falling edge of the SCLK signal and is received (latched into the microcontroller) on the rising edge of the SCLK signal. In the Inverted Clock mode, data is transmitted on the SDATA pin on the rising edge of the SCLK signal and is received (latched into the microcontroller) on the falling edge of the SCLK signal.

When no transmit or receive transaction is active on the SSI, the SDATA signal is three-stated (although it has a weak keeper to hold the last value driven on the SDATA signal). When data is transmitted on the SSI from the microcontroller to another device, the SDATA signal is driven and is stable after the falling edge (rising edge if in the Inverted Clock mode) of the SCLK signal, providing the appropriate setup and hold time for the receiving device if that device latches this data on the rising edge (falling edge in the Inverted Clock mode) of SCLK.

14.5.3.2 SDATA

When the microcontroller receives data from another device on the SSI, it latches the level driven onto the SDATA signal by the transmitting device on the rising edge (falling edge if in the inverted clock mode) of the SCLK signal. The transmitting device must meet the required setup and hold times relative to this SCLK rising edge (falling edge if in the inverted clock mode).

Software writes data to be transmitted on the SSI by the microcontroller to either of the two SSI transmit registers (SSTXD0 or SSTXD1). The transmit registers are 16-bit registers but only the lower eight bits can be written and the upper eight bits are ignored. When a new value is written to one of the transmit registers, and software has previously enabled SSI and the external device (see the description of the SDEN signal below), the SSI shifts out the data written to the transmit register on SDATA.

To receive data from an external device, the microcontroller must initiate the receive transaction by toggling the SCLK signal and sampling the SDATA input. A receive transaction is initiated if the external device has been enabled (see the description of the SDEN signal below) and software reads the SSI Receive Data (SSRXD) register. The SSRXD register is a 16-bit register but only the lower eight bits contain valid data. If the external device is enabled, reading the SSRXD register causes the SCLK signal to be toggled, generating eight Low-to-High transitions (High-to-Low transitions if in the Inverted Clock mode), and the level on the SDATA signal is latched eight times and stored in the receive register bits. Note that the initial data read (activating the read cycle) should be discarded.

The SSI data order is configurable with the MSBF bit in the SSCON register. Two modes are available: Normal (LSB first) and Reverse (MSB first). A single configuration bit selects the mode and the selected mode is common for transmit and receive operations.

In Normal mode, the least significant bit (LSB) of the transmit data byte is shifted out first. For a receive operation, the SSI stores the first data bit received in the LSB of the receive register and stores the last data bit received in the most significant bit (MSB) of the receive register.

In Reverse mode, the most significant bit (MSB) of the transmit data byte is shifted out first. For a receive operation, the SSI stores the first data bit received in the MSB of the receive register and stores the last data bit received in the LSB of the receive register.

14.5.3.3 SDEN

The SDEN signal enables an external device for communication on the SSI bus. The microcontroller asserts this signal, under software control, before it initiates the transmit or receive operation to or from a device on the SSI. The DE0 bit controls the state of this

signal. Setting DE0 asserts the SDEN signal. Asserting SDEN enables the external device to which this signal is connected for communication on the SSI. Writing the transmit register or reading the receive register initiates a data transfer on the SSI.

Software can configure the SDEN signal to be active High or Low with the DENP bit in the SSCON register.

For a receive operation, reading the SSRXD register when the synchronous serial data enable bits DE0 and DE1 of the SSCON register are cleared returns the data in the register to the CPU without generating a receive transaction on the SSI.

It is possible to support multiple devices connected to the SSI bus simultaneously. In one scenario, it may be possible to connect all the devices to the SDEN signal and develop a software protocol to manage individual device communication.

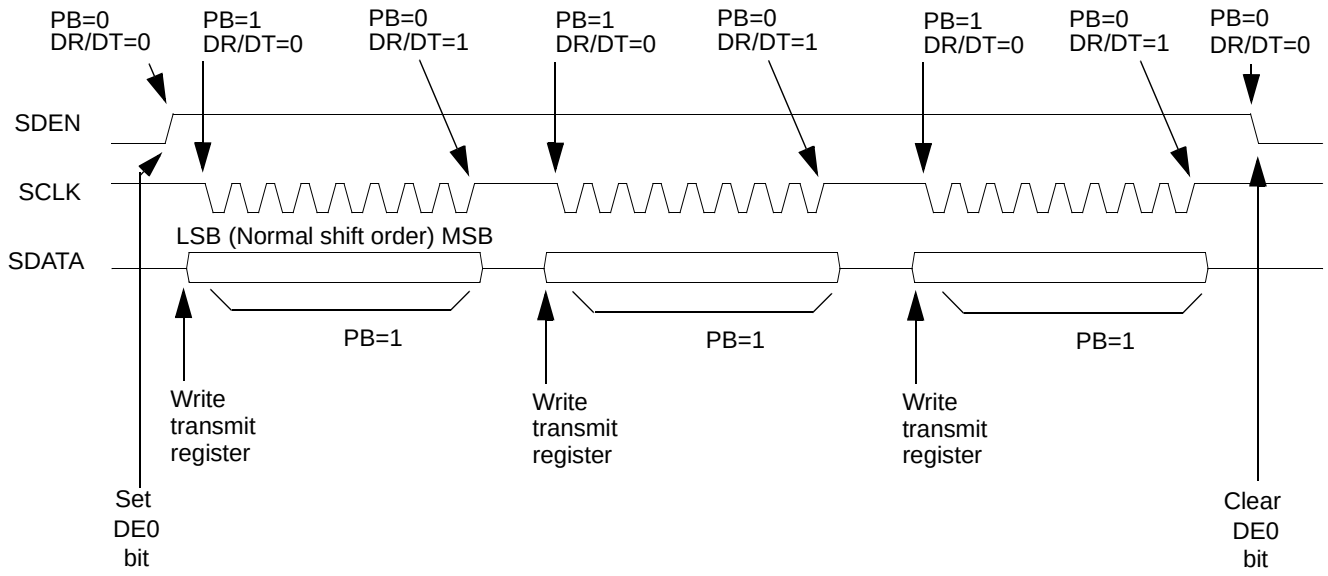
Alternatively, PIO signals can serve as external device enables in addition to the provided SDEN signal. In this scenario, to communicate to one of the devices using a PIO as an SSI enable signal, software must configure the pin as a PIO output, force the PIO to be asserted, and then set the synchronous serial data enable bit (DE1) in the SSCON register. Setting this bit and asserting the PIO enables the external device to which this PIO signal is connected for communication on the SSI and writing the SSTXDx register or reading the register initiates a data transfer. For a receive operation, reading the SSRXD register when the DE1 and the DE0 bits are cleared returns the data in the receive register to the CPU without causing a receive transaction to be generated on the SSI. Note that the DE0 and DE1 bits can be set simultaneously to achieve proper receive/transmit operation.

14.5.3.4 SSI Transactions

In general, the SSI hardware provides software with a polled I/O mechanism to control its operation. In addition to the transmit register, the receive register, and the control register, one status register is provided. The SSI Mode/Status (SSSTAT) register provides software with “busy”, “receive/transmit end”, and “error” status. These bits are called PB (busy), DR/DT (receive/transmit end), and RE/TE (error). A write to either SSTXD1 or SSTXD0, or a read to SSRXD while PB=1, sets the RE/TE bit and does not generate additional data transfers.

For SSI transmit and receive transaction examples, see Figure 14-3–Figure 14-5.

Figure 14-3 SSI Multiple Transmit with SDEN as External Device Enable



Notes:

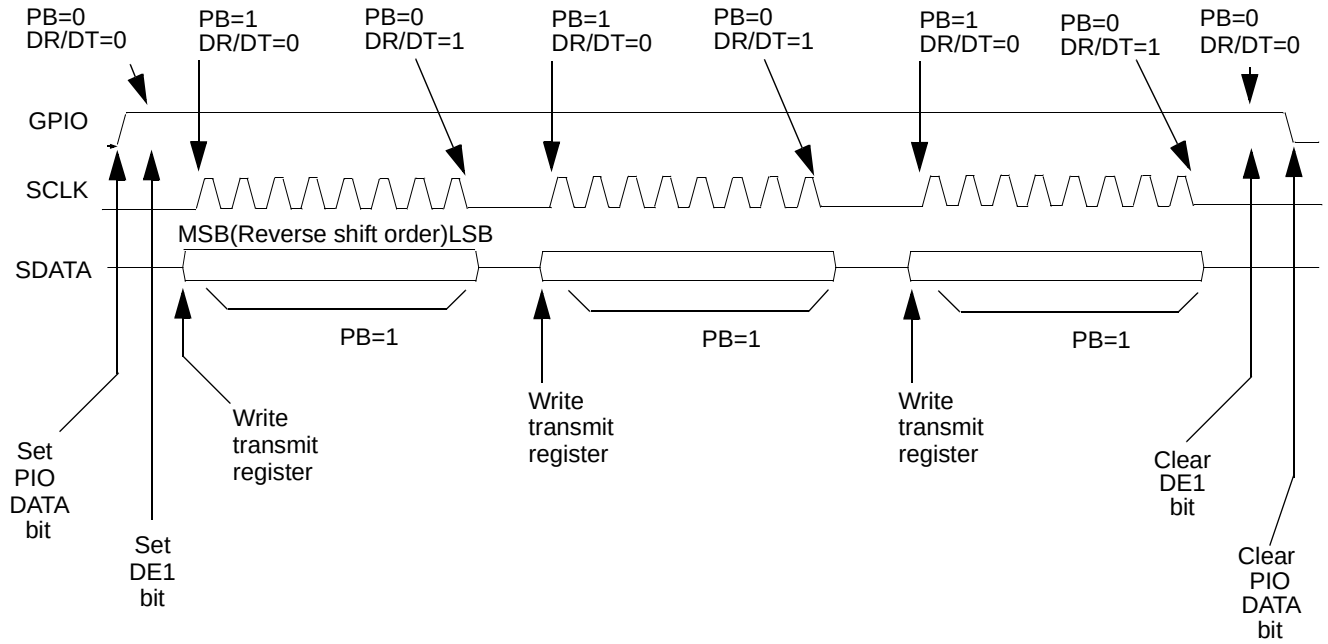
SDEN is configured to be active High in the scenario shown above.

Any PIOs used as SSI enables should be inactive while SDEN is active.

The SSI data order is configured to be in Normal mode (LSB first).

The SSI clock is configured to be in Normal mode.

Figure 14-4 SSI Multiple Transmit with PIO as External Device Enable



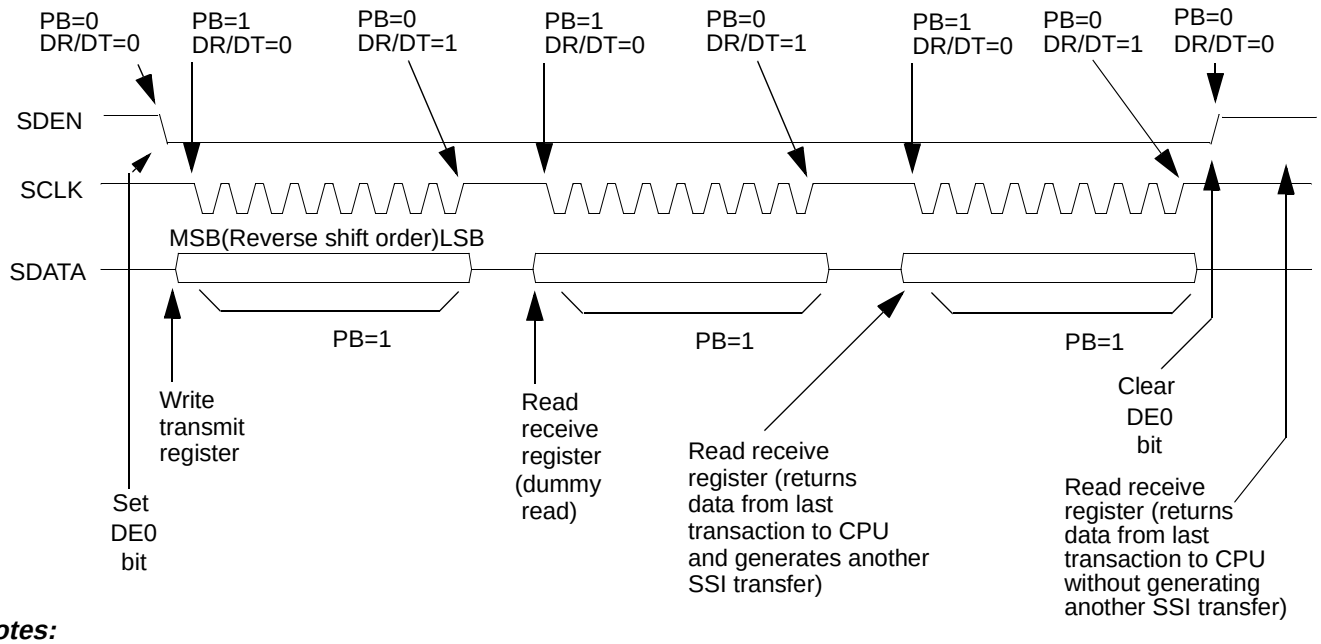
Notes:

The SDEN signal should be inactive (DE0=0) while the PIO has enabled the receiving device.

The SSI data order is configured to be in Reverse mode (MSB first).

The SSI clock is configured to be in Inverted Clock mode.

Figure 14-5 SSI Single-Transmit, Multiple-Receive with SDEN as External Device Enable



Notes:

SDEN is configured to be active Low in the scenario shown above.

Any PIOs used as SSI enables should be inactive while SDEN is active.

The SSI data order is configured to be in Reverse mode (MSB first).

The SSI clock is configured to be in Normal mode.

14.5.4 Software-Related Considerations

The SSI interface allows for a variety of software and hardware protocols:

- **Signaling a read/write:** In general, software uses the first write to the SSI to transmit an address or count to the peripheral. This value can include a read/write flag in the case where the device supports both reads and writes.
- **Using SSTXD1 as an address register:** The SSTXD1 register can be an address register that holds the value of the last address accessed, and the SSTXD0 register can be the data transmit register. In this case, the current value in the SSTXD1 register can be used by software to generate the next address or to determine if the last transaction was a read or a write.
- **Using SSTXD1 and SSTXD0 as transmit registers for two peripheral devices:** In some systems, it may clarify the code and aid in debugging to view the two data transmit registers as unique to different peripheral devices. This allows the last value transmitted to each device to be examined by debug code.

14.5.5 Comparison to Other Devices

The SSI is mostly backward-compatible with software written for the Am186EM SSI. Additional features have been added to the SSI implementation. In its default mode, the SSI on the Am186CC/CH/CU microcontrollers is backward-compatible with the Am186EM with the following exceptions:

- The SSI status and configuration register locations in the address map are different.

- Only one dedicated SSI enable pin is available. PIOs can be used for additional device enables if they are required.
- Software written for the Am186EM SSI that writes to the SSI status register does not work on the Am186CC/CH/CU microcontrollers.
- Only the /2 and /4 clock modes are available unless software sets the ENHCTL bit in the SSSTAT register.

Features added to the SSI are:

- Programmable data order: Normal (least-significant bit first) or Reverse (most-significant-bit first)
- Programmable clock divisor: Divide the clock from 2 to 256 in power of 2 increments
- Programmable polarity: SCLK and SDEN

14.6 INITIALIZATION

On both external and internal reset, the following occurs:

- SSSTAT is set to 0000h, which clears status and disables Enhanced Control mode.
- SSCON is set to 0400h, which sets SCLK to active Low, SDEN to active High, the LSB transmitted and received first, the clock divisor to 2, and disables SSI operation.
- The SSTXD0, SSTDX1, and SSRXD registers are set to 0000h, which clears all data.
- The multiplexed serial pins default to PIO functionality (see Table 14-1 on page 14-2).

CHAPTER

15 HIGH-LEVEL DATA LINK CONTROL (HDLC)

CC CH ***Note:*** *Only the Am186CC and Am186CH microcontrollers support HDLC.*

15.1 OVERVIEW

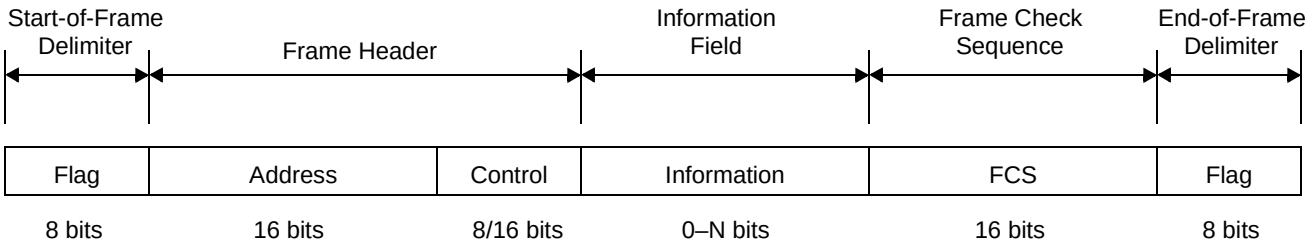
In the Open Systems Interconnection (OSI) model, layer two is the data link layer. This layer provides control between physical nodes: link initialization, flow control, and error control. One protocol that performs this function is High-level Data Link Control (HDLC). In HDLC, all transmissions are in frames. The ISO/IEC 3309 standard specifies this frame structure.

The Am186CC and Am186CH microcontrollers provide HDLC channels, which are used to transmit and receive frames based on HDLC formats. As a layer 2 function, these channels only transmit or receive the data; upper layers in the OSI model actually look at the data.

An HDLC frame uses flags to determine the start and end of a frame.These flags provide frame synchronization. One flag may be used as both an end flag for one frame and the start flag for the next frame. Although the Am186CC and Am186CH microcontrollers do not transmit such shared flags, they can receive and properly handle a shared flag.

As illustrated in Figure 15-1, an HDLC frame typically consists of a start flag, followed by an address field, a control field, an information field, a frame checking sequencing (FCS) field, and, finally, a closing flag. Frames maintain *data transparency*—a flag, mark, or abort embedded in the data is not recognized—by *bit stuffing* and *bit unstuffing.* Bit stuffing (also called *zero-bit insertion*) occurs when transmitting data; the transmitter inserts a 0 after five consecutive 1s. *Bit unstuffing* (also called *zero-bit deletion*) occurs when receiving data; between opening and closing flags, the receiver deletes any 0 received after five consecutive 1s.

Figure 15-1 HDLC Frame



In the transmit direction (data is leaving the microcontroller), you can program the HDLC controller to add the required frame checking sequencing field (CRC error detection bytes) at the end of the frame, bit stuff the data as needed, and surround it with flags. (Cyclic Redundancy Check (CRC) is a method for checking errors in transmitted data.)

In the receive direction (data is coming into the microcontroller), the HDLC controller searches for flags to determine the start and stop of the frame, removes any bit stuffing, and checks the CRC error detection bytes. The HDLC controller can also check the address of the incoming frame and reject it if it has an incorrect address.

The microcontroller uses FIFOs in both directions (16-byte transmit and 32-byte receive) to isolate the data requests from the system bus. The controller supports SmartDMA and programmed I/O for filling or emptying the FIFOs.

Each HDLC channel can connect to an external serial interface directly (nonmultiplexed mode) or can pass through a time slot assigner (multiplexed mode). An HDLC channel can connect to a raw Data Communications Equipment (DCE) interface in nonmultiplexed mode, to a Pulse Code Modulation (PCM) highway interface in multiplexed mode, or to a General Circuit Interface (GCI) in multiplexed mode. Each HDLC channel has the same feature set but separate connections to its associated time slot assigner. For more information about how the HDLC channels can be connected externally, see Chapter 16, “HDLC External Serial Interface Configuration (TSAs)”.

The HDLC channels support full-duplex data transfer at a rate of up to 10 Mbit/s in raw DCE and PCM Highway modes, and up to 768 Kbit/s in GCI mode (system performance may limit total throughput). The microcontroller contains internal PCB registers for configuring the modes of operation, controlling the HDLC channels, monitoring and reporting status, and moving data. Each HDLC channel consists of a transmitter, a receiver, and the interface (programmed I/O or SmartDMA).

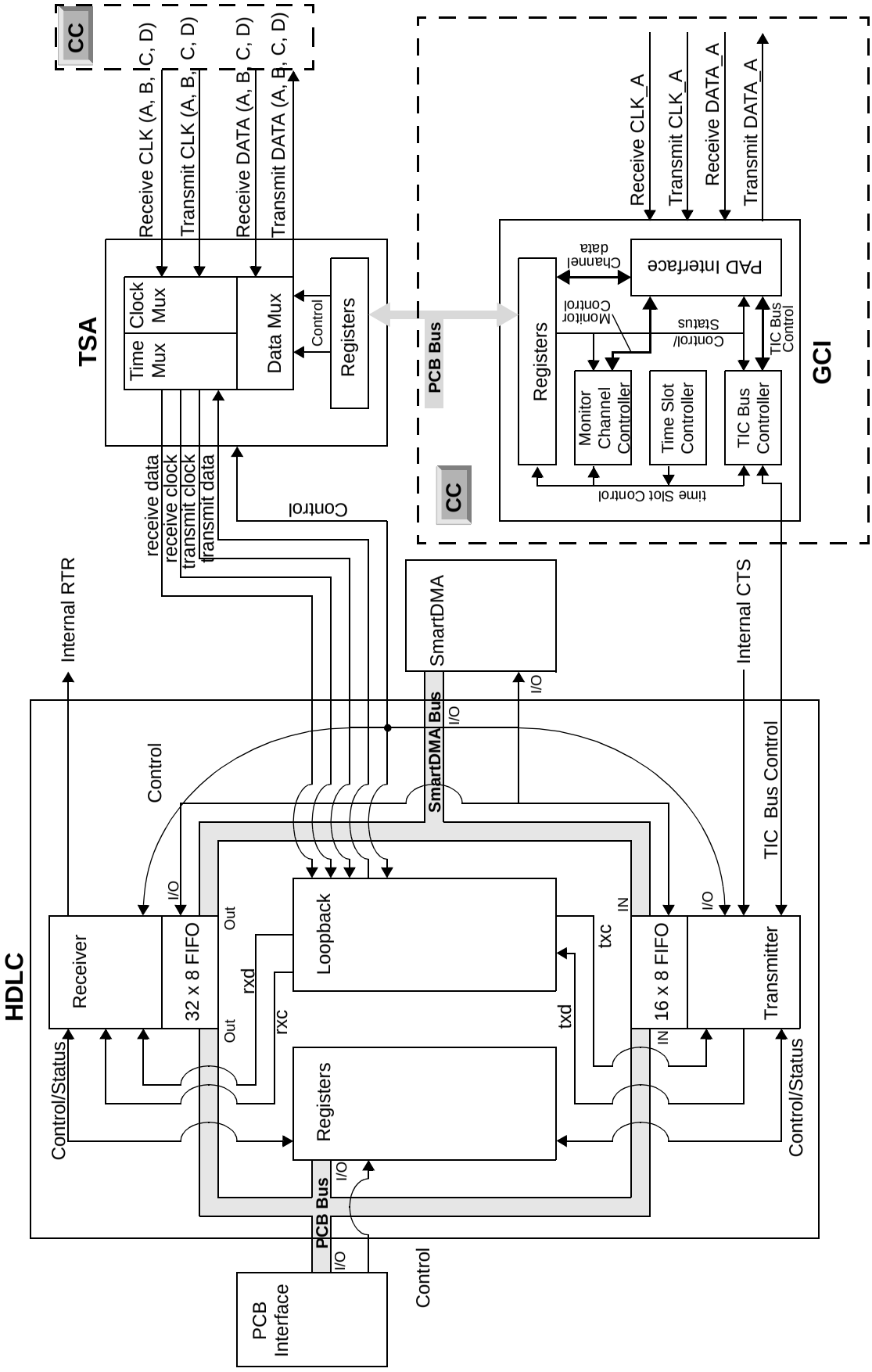
CC The Am186CC microcontroller provides four HDLC channels, A through D, which support raw DCE, PCM highway, and GCI external interfaces.

CH The Am186CH HDLC microcontroller provides two HDLC channels, A and B, which support raw DCE and PCM highway external interfaces.

15.2 BLOCK DIAGRAM

Figure 15-2 shows the block diagram for a single HDLC channel, including connections with the TSA and GCI.

Figure 15-2 HDLC, TSA, and GCI Block Diagram



15.3 SYSTEM DESIGN

Table 15-1 lists the HDLC/TSA/GCI signals that are multiplexed with other microcontroller functions. Pinstraps are sampled only at external reset and do not affect the pin's other functions, so they are not shown in this table. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin.

Table 15-1 HDLC/TSA/GCI Multiplexed Signals

Multiplexed Signals						Function	Default Signal
Ch	External Interface				PIOs		
	DCE	PCM	GCI CC	UART			
A	DCE_RXD_A	PCM_RXD_A	GCI_DD_A	—	—	DCE and PCM data input/ GCI downstream pin	DCE_RXD_A
	DCE_TXD_A	PCM_TXD_A	GCI_DU_A	—	—	DCE and PCM data output/ GCI upstream pin	DCE_TXD_A
	DCE_RCLK_A	PCM_CLK_A	GCI_DCL_A	—	—	DCE receive clock/PCM receive and transmit clock/GCI receive and transmit clock	DCE_RCLK_A
	DCE_TCLK_A	PCM_FSC_A	GCI_FSC_A	—	—	DCE transmit clock/PCM frame sync clock/GCI frame sync clock	DCE_TCLK_A
	$\overline{\text{DCE\_CTS\_A}}$	$\overline{\text{PCM\_TSC\_A}}$	—	—	PIO17	DCE clear to send/PCM external buffer enable	PIO17
	$\overline{\text{DCE\_RTR\_A}}$	—	—	—	PIO18	DCE ready to receive	PIO18
B	DCE_RXD_B	PCM_RXD_B	—	—	PIO36	DCE and PCM data input pin	PIO36
	DCE_TXD_B	PCM_TXD_B	—	—	PIO37	DCE and PCM data output pin	PIO37
	DCE_RCLK_B	PCM_CLK_B	—	—	PIO40	DCE receive clock/PCM receive and transmit clock	PIO40
	DCE_TCLK_B	PCM_FSC_B	—	—	PIO41	DCE transmit clock/PCM frame sync clock	PIO41
	$\overline{\text{DCE\_CTS\_B}}$	$\overline{\text{PCM\_TSC\_B}}$	—	—	PIO38	DCE clear to send/PCM external buffer enable	PIO38
	$\overline{\text{DCE\_RTR\_B}}$	—	—	—	PIO39	DCE ready to receive	PIO39
C CC	DCE_RXD_C	PCM_RXD_C	—	—	PIO42	DCE and PCM data input pin	PIO42
	DCE_TXD_C	PCM_TXD_C	—	—	PIO43	DCE and PCM data output pin	PIO43
	DCE_RCLK_C	PCM_CLK_C	PCM_CLK_C	—	PIO22	DCE receive clock/PCM receive and transmit clock input/GCI-to-PCM conversion clock output	PIO22
	DCE_TCLK_C	PCM_FSC_C	PCM_FSC_C	—	PIO23	DCE transmit clock/PCM frame sync clock input/GCI-to-PCM conversion frame sync output	PIO23
	$\overline{\text{DCE\_CTS\_C}}$	$\overline{\text{PCM\_TSC\_C}}$	—	—	PIO44	DCE clear to send/PCM external buffer enable	PIO44
	$\overline{\text{DCE\_RTR\_C}}$	—	—	—	PIO45	DCE ready to receive	PIO45

Table 15-1 HDLC/TSA/GCI Multiplexed Signals (Continued)

Multiplexed Signals						Function	Default Signal
Ch	External Interface				PIOs		
	DCE	PCM	GCI CC	UART			
D CC	DCE_RXD_D	PCM_RXD_D	—	RXD_U	PIO26	DCE and PCM data input/ UART data receive	PIO26
	DCE_TXD_D	PCM_TXD_D	—	TXD_U	PIO20	DCE and PCM data output/ UART data transmit	PIO20
	DCE_RCLK_D	PCM_CLK_D	—	RTR_U	PIO25	DCE receive clock/PCM receive and transmit clock input/UART ready-to-receive	PIO25
	DCE_TCLK_D	PCM_FSC_D	—	CTS_U	PIO24	DCE transmit clock/PCM frame sync clock input/UART clear-to-send	PIO24
	$\overline{\text{DCE\_CTS\_D}}$	$\overline{\text{PCM\_TSC\_D}}$	—	CTS_HU	PIO46	DCE clear to send/PCM external buffer enable/High-Speed UART clear-to-send	PIO46
	$\overline{\text{DCE\_RTR\_D}}$	—	—	RTR_HU	PIO47	DCE ready to receive/High-Speed UART ready-to-receive	PIO47

15.4 REGISTERS

Table 15-2 lists the 25 unique registers that control a single HDLC channel. The x shown in the register name can be A, B, C, or D, depending on the channel selected. The table shows the offset for Channel A; for Channel B, add 40h to the offset shown. Both the Am186CC and Am186CH microcontrollers support Channels A and B.

CC The Am186CC microcontroller also supports Channel C and D. Add 80h to the offset shown for Channel C, and add C0h for Channel D.

In addition to the registers shown in Table 15-2, the System Configuration (SYSCON) register, offset 03F0h, has two bit fields that configure HDLC: ITF4 (bits 9–8) and EXSYNC (bit 7).

CC In the Am186CC microcontroller, the IFT4 bit field configures the interface of HDLC Channel D. Setting the EXSYNC bit causes the clock and frame information to be driven out of HDLC Channel C.

CH In the Am186CH HDLC microcontroller, there is no HDLC Channel D, but the ITF4 bit field default value is 00b, specifying full HDLC with flow control. Therefore, software must change the value of the ITF4 bit field to 10b before using the UART interface or High-Speed UART with flow control.

Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 15-2 HDLC Register Summary

Offset[1]	Register Mnemonic[2]	Register Name	Description
00h	HxCON	HDLC Channel Control	Sets operating modes for both the transmitter and receiver.
02h	HxTCON0	HDLC Channel Transmit Control 0	Sets operating modes for transmitter.
04h	HxTCON1	HDLC Channel Transmit Control 1	
06h	HxRCON0	HDLC Channel Receive Control	Sets operating modes for receiver.
08h	HxRCON1	HDLC Channel Receive Max Length	Sets maximum length for received frame. Should never be set to 2 or less.
0Ah	HxSTATE	HDLC Channel Status	Contains read-only status for transmitter and receiver.
0Ch	HxISTAT0	HDLC Channel Interrupt Status 0	Contains status for transmitter and receiver. All bits can generate an interrupt if not masked off in HxIMSK0.
0Eh	HxIMSK0	HDLC Channel Interrupt 0 Mask	Mask register for HxISTAT0. When a mask bit is 0 (the reset value), the corresponding interrupt is masked off.
10h	HxISTAT1	HDLC Channel Interrupt Status 1	Contains status for receiver. All bits can generate an interrupt if not masked off in HxIMSK1.
12h	HxIMSK1	HDLC Channel Interrupt 1 Mask	Mask register for HxISTAT1. When a mask bit is 0 (the reset value), the corresponding interrupt is masked off.
14h	HxTD	HDLC Channel Transmit FIFO Data	Contains data for transmission.
16h	HxRD	HDLC Channel Receive FIFO Data	Contains data from receive. After all the data has been read, this register contains the three bytes of status for the frame, as described below.
	H*x*RFS1	HDLC Channel Receive Frame Status 1	Contains the first byte of status: the least significant byte of the length of the received frame.
	HxRFS2	HDLC Channel Receive Frame Status 2	Contains the second byte of status: the most significant byte of the length of the received frame.
	HxRFS3	HDLC Channel Receive Frame Status 3	Contains the third byte of status: the status for the frame and which address was matched.
18h	HxRDP	HDLC Channel Receive FIFO Data Peek	Copy of HxRD register that does not change when read.
1Ah	HxSFCNT	HDLC Channel Short Frame Counter	Contains the count of the number of frames that were discarded because they were smaller than the minimum receive length. Reading this register resets it to 0.
1Ch	HxSFCNTP	HDLC Channel Short Frame Counter Peek	Copy of HxSFCNT register that does not change when read.
1Eh	HxMACNT	HDLC Channel Mismatch Address Counter	Contains the count of the number of frames that were discarded because they did not match any of the address registers. Reading this register resets it to 0.

Table 15-2 HDLC Register Summary (Continued)

Offset[1]	Register Mnemonic[2]	Register Name	Description
20h	HxMACNTP	HDLC Channel Mismatch Address Counter Peek	Copy of HxMACNT register that does not change when read.
22h	HxA0	HDLC Channel Address 0	Contains the value to compare to the address in the received frames. The address bits can be masked with HxA0MSK.
24h	HxA0MSK	HDLC Channel Address Mask 0	Mask register for HxA0. When a mask bit is 0 (the reset value), the bit is always matched. This means all frames are accepted when none of the mask bits are set.
26h	HxA1	HDLC Channel Address 1	Address match and mask registers. See descriptions for HxA0MSK and HxA0.
28h	HxA1MSK	HDLC Channel Address Mask 1	
2Ah	HxA2	HDLC Channel Address 2	
2Ch	HxA2MSK	HDLC Channel Address Mask 2	
2Eh	HxA3	HDLC Channel Address 3	
30h	HxA3MSK	HDLC Channel Address Mask 3	

Notes:

1. The x *shown in the register name can be A, B, C, or D, depending on the channel selected. The offset shown is for Channel A; for Channel B, add 40h to the offset shown; for Channel C, add 80h; and for Channel D, add C0h.*

2. The Am186CC and Am186CH microcontrollers support Channels A and B; the Am186CC microcontroller also supports Channels C and D.

15.5 OPERATION

15.5.1 Usage

Note: *Before using the HDLC channels, configure the multiplexed pins for HDLC use (see Table 15-1 on page 15-4). When using HDLC Channel D on the Am186CC microcontroller, be sure to configure the ITF4 bit in the SYSCON register correctly.*

The HDLC portion of the microcontroller is an extremely flexible serial communications block that can be configured to support data movement in a variety of applications. When initializing the HDLC channels for a particular operation, it is best to establish the Time Slot Assigner and general HDLC functionality before beginning data reception or transmission. Configure HDLC functionality through the HxCON, HxTCON0, HxTCON1, and HxRCON1 registers. Establish address matching through the address match registers and their associated masks. Finally, enable the desired interrupts by setting the corresponding bits in the appropriate mask registers.

To configure the HDLC channels, use the following procedure:

1. Configure the Time Slot Assigners (TSAs) as described in “Usage” on page 16-7.
2. Configure any required HDLC channel operating modes:
   a. Configure the NRZI encoding, transparent mode, loop remote, loop local, or CRC type by programming the HxCON register.
   b. For transmissions, configure the flag idle, multidrop mode, automatic CTS, bit order, clock invert, GCI (on the Am186CC microcontroller only), output drive, or transmit delay by programming the HxTCON1 register.

3. Set the necessary transmit enables (HxTXON0 register) and receive enables (HxRCON0 register) for each HDLC channel.
4. Do an HDLC reset. A reset flushes the FIFOs and clears all R/0 status bits, but does *not* clear the R/W0 interrupt status registers.
5. Clear all pending interrupts by writing 0s to the INTSTS register.

15.5.2 Interface

The HDLC channels operate in one of two modes: SmartDMA data transfer or programmed I/O. SmartDMA data transfer provides automated data movement to the transmit FIFO or from the receive FIFO. Programmed I/O is intended for low data rates where processor intervention is possible on a byte-by-byte basis.

15.5.2.1 SmartDMA Interface

Using the SmartDMA interface bypasses the HDLC status registers associated with data handling (HxSTATE, HxISTAT0, HxISTAT1, HxRFS1, HxRFS2, HxRFS3, HxASBMSB, HxASBLSB) because the SmartDMA interface automatically places all data and status to and from the data buffers and buffer descriptors residing in memory. Some applications still require additional status such as the link status. For more information about the SmartDMA interface, see Chapter 8, “DMA Controller.”

15.5.2.2 Programmed I/O Interface

15.5.2.2.1 Transmit Programmed I/O Interface

To transmit a frame using programmed I/O, first program the control registers with the appropriate mode(s), then enable the transmitter. Then, either poll a status bit indicating transmit space is available or use the transmit space available interrupt to determine when space is available in the transmit FIFO. Just after writing the last byte of a frame to the transmit FIFO, set the “last byte” bit in the control register. When the last byte in the frame is written to the transmit FIFO, the HDLC controller knows to append the CRC (if enabled) and closing flag. When the last byte of the frame has been transmitted, the transmitter generates a maskable interrupt and sets a status bit.

The interrupt register indicates if there was an underflow of the transmit FIFO, if $\overline{\text{CTS}}$ was lost during transmission, or if the transmit frame was aborted during transmission. When one of these conditions occurs, the HDLC controller flushes the transmit FIFO and stops the transmitter until the appropriate status bit is cleared.

15.5.2.2.2 Receive Programmed I/O Interface

To receive a frame using programmed I/O, first program the control registers with the appropriate mode(s) and then enable the receiver. Then, either poll a status bit (one receive data byte available) or take an interrupt (receive threshold reached or data byte available) to determine that data is available in the receive FIFO. This data can then be read from the receive FIFO interface register. A different status bit (received end of frame) is active to indicate that frame status is now available in the receive FIFO. A maskable interrupt is also generated. The status consists of three bytes: the first two bytes are the frame byte count and the last byte is the general frame status. When performing a word (16-bit) read of the FIFO, the lower byte contains the data and the upper byte indicates whether the lower byte is data or status byte one, two, or three. The upper byte also indicates whether the programmed FIFO threshold has been reached and if any interrupts are pending.

The receive status indicates the following information:

- If there was an overflow of the receive FIFO
- If a non-integer number of bytes were received
- If a CRC error was detected
- Which address was matched
- If the frame was too short or too long
- If the receiver was turned off during the frame
- If the frame ended with an abort (one zero followed by seven to 14 consecutive 1s).

At the end of frame 1, software must read the status of frame 1 from the receive FIFO before it can read any data from frame 2.

15.5.3 General HDLC Options

These options involve both the transmitter and the receiver. For transmitter-specific options, see “HDLC Transmitter” on page 15-10; for receiver-specific options, see “HDLC Receiver” on page 15-14.

- **Data Clocks:** Each HDLC channel requires two clock sources: a transmit clock for the transmit data, and a receive clock for the receive data.
- **HDLC Reset:** To initiate HDLC reset, set the HRESET bit in the HxCON register to 1. HDLC reset clears the HDLC channels and FIFOs and restores all status registers to their default values. HDLC reset does not affect the user-programmed control bits.
- **NRZ/NRZI Data Encoding:** The microcontroller supports both non-return to zero (NRZ) and non-return to zero, invert on zero (NRZI) data formats. Specify the encoding format with the NRZI bit of the HxCON register.
- **Transparent Mode:** Transparent mode disables zero-bit insertion and deletion, CRC generation and checking, abort generation, and opening/closing flag generation. The HDLC controller transmits data exactly as it is loaded in the transmit FIFO. When the FIFO is empty, the controller generates idles (mark or flag) and does not set the abort bit. If $\overline{\text{CTS}}$ is deasserted in Transparent mode, the transmitter goes to the idle state. Transparent mode also disables the receive byte counter; therefore, short frame and long frame errors are not reported. Byte alignment is possible in all modes except raw DCE. Additionally, alignment is not possible when the entire time-multiplexed bus is allocated to a single TSA/HDLC channel. To enable Transparent mode, set the TRANSM bit in the HxCON register to 1.

  To use byte alignment when using Transparent mode with a time-multiplexed data format, set, then clear, the HRESET bit in the HxCON register after configuring the TSA and HDLC channels and establishing operation. The first byte received or transmitted may be corrupted while the HDLC channel is performing the alignment. To mask this effect on the transmit side, configure the transmitter to use mark idles and make the first transmitted byte all 1s (FFh).

  To maintain byte alignment, all time slot widths used must be a multiple of eight bits and there must be at least one empty time slot. HDLC requires the unused time slot to properly locate the byte boundary. If the transmit FIFO underflows, the transmitter loses byte alignment.

- **Remote Loopback Mode:** To enable Remote Loopback mode, set the LOOPR bit in the HxCON register to 1. Remote Loopback disables the transmitter and echoes the data received at the serial input out to the serial output. The receiver operates normally in this mode.
- **Local Loopback Mode:** To enable Local Loopback mode, set the LOOPL bit in the HxCON register to 1. Local Loopback mode disconnects the serial input and connects the serial output to the receiver input. The serial output can operate in three-state, open drain, or totem pole mode.
- **CRC Type:** The algorithm for CRC generation and checking can be CRC-CCIT, CRC-16, or CRC-32. Specify the CRC type in the CRCTYPE field of the HxCON register.
- **Time Slot Assigner (TSA):** Each HDLC channel is tightly coupled with a TSA, which can operate in either multiplexed or nonmultiplexed (pass-through) mode. In multiplexed mode, the TXCLK input becomes the synchronization input and the TSA connects the receive clock to the transmit clock. In multiplexed mode, the TSA controller determines when to enable and disable the HDLC clock. It also allows the user to reduce the number of bits transmitted in a single 8-bit time slot. This reduction allows the transmission of data from 64 Kbit/s down to 8 Kbit/s in 8 Kbit/s decrements. This feature allows the HDLC channel to be used for LAP-D and reduced data mode LAP-B transmissions such as 56 Kbit/s.

15.5.4 HDLC Transmitter

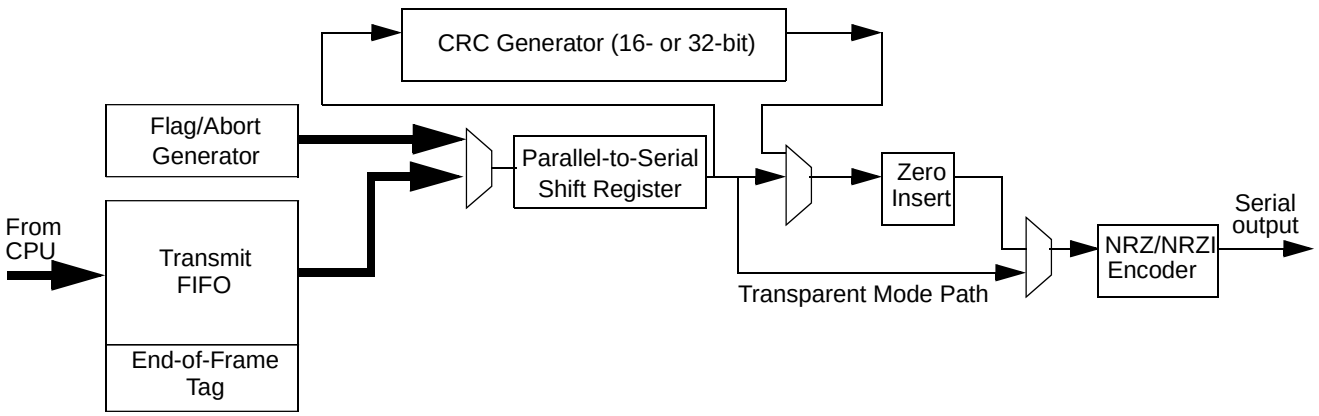
The transmitter functions include:

- Opening flag transmission
- Data transparency (via zero insertion)
- Generation and transmission of the CRC frame-check-sequence characters (if enabled)
- Transmission of the closing flag.

Figure 15-3 illustrates the block diagram for the transmitter.

Note: *The HDLC transmitter requires at least one byte of data surrounded by flags: the start flag, one byte of data, and the end flag. A 2-byte CRC with no data also constitutes a valid transmission. The HDLC receiver can receive only frames two bytes or longer.*

Figure 15-3 HDLC Transmitter Block Diagram



The HDLC transmitters have the following features:

- **Transmit FIFO:** The transmit FIFO consists of a 16-byte FIFO buffer, end-of-frame logic, and DMA-request logic. When using programmed I/O to fill the transmit FIFO, a bit must be set after the last byte in a frame is written to the FIFO. The SmartDMA interface uses the terminal count signal from the DMA controller. For more information, see Chapter 8, “DMA Controller.” Read the transmit FIFO with the HxTD register.
- **Transmit-FIFO Interface:** When the transmit FIFO requests data, it either generates an internal DMA request or sets the TDATA1 bit in the HxISTAT0 register indicating transmit space is available. This status bit being set can generate a maskable interrupt.
- **Transmit-FIFO Threshold:** The transmit FIFO has three options for the level at which it requests data, specified in the TTHRSH field of the HxTCON0 register:
  - When there is space available in the transmit FIFO (TTHRSH = 00)
  - When there are 9 bytes of FIFO space available (TTHRSH = 01)
  - When there are 16 bytes of FIFO space available (TTHRSH = 10)

  Reaching the transmit threshold also generates a maskable interrupt (indicated in the TTHRES bit of the HxISTAT0 register).
- **Transmit-Space Available:** For programmed I/O, the TDATA1 bit in the HxISTAT0 register indicates when there is space available in the transmit FIFO. This indication is independent of the threshold selected. The space available status can also generate a maskable interrupt.
- **Transmit-FIFO Underflow:** When the transmit FIFO underflows, it generates a maskable interrupt, enters the abort state, and reports a TUFLO error status.
- **Transmit-Clock Polarity:** The transmit clock polarity is specified in the TXCINV bit of the HxTCON1 register, independent of the receive clock polarity. This feature is recommended for use only in DCE mode.
- **Immediate-Transmit Start:** When immediate transmit start is enabled, the transmitter begins transmitting as soon as data is available in the transmit FIFO. When immediate transmit start is disabled, the transmitter does not start transmitting until the FIFO is half full or the complete frame is in the FIFO, whichever comes first. To enable immediate transmit start, set the IMSTART bit in the HxTCON0 register to 1.
- **Flag- or Mark-Idle Generation:** The HDLC transmitter can transmit either flag- or mark-idles when the transmitter is enabled and is not actively sending a data frame (including the opening and closing flags) or an abort sequence. Specify a flag idle by setting the FLAGIDL bit in the HxTCON1 register to 1; specify a mark idle by clearing the FLAGIDL bit to 0. A flag is 7Eh (the sequence of one 0, six 1s, and one 0); a mark idle sequence is fifteen 1s; an abort sequence is one 0 followed by from seven to 14 consecutive 1s. To properly support multidrop configurations with collision detection, the HDLC transmitter should be configured to generate mark-idles. When transmitting flag- or mark-idles, the transmitter is in the idle condition.
- **Flag Generation with Back-to-Back Frames**: The minimum number of flags between frames transmitted is two. At least one closing flag is always generated at the end of a frame, and at least one opening frame is generated at the beginning of a frame (except in Transparent mode). Back-to-back flags are sent without sharing 0s (i.e., 0111111001111110, not 011111101111110).

- **Abort Generation:** The HDLC transmitter sends an abort sequence (one 0 followed by seven to 14 1s) whenever the FORABR bit of the HxTCON0 register is set to 1. The transmitter continues sending an abort sequence as long as this bit is set; however, if the send abort bit is set and cleared on two successive writes to the HDLC Command/Control register, at least one abort character is sent. An abort is also sent if $\overline{\text{CTS}}$ is lost while the transmitter is in-frame (and $\overline{\text{CTS}}$ is enabled) or if a transmit FIFO underflow occurs (unless in Transparent mode). When in GCI (Am186CC microcontroller only) or multidrop mode, only one abort is sent and then the transmitter is turned off.
- **Parallel-to-Serial Shift Register:** The HDLC transmitter loads the output of the transmit FIFO or the flag/abort generator one byte at a time into the parallel-to-serial shift register and then shifts it out. Transmission of a flag or abort sequence bypasses the zero-bit-insertion logic.
- **CRC Generator:** The CRC or Frame Check Sequence (FCS) contains the generated CRC code for the frame being transmitted. All data transmitted between the opening and closing flags (excluding inserted 0s) is included in the CRC calculation. The transmitter appends the calculated CRC to the end of the frame just before the closing flag. The transmitter supports the CRC-CCITT, CRC-16, and CRC-32 algorithms, selected in the CRCTYPE field of the HxCON register. You can disable CRC generation by setting the CRCDIS bit of the HxTCON0 register to 1. When CRC is disabled, the transmitter does not append the CRC bytes to the end of the frame. The disable option may be changed at any time before the last byte is to be transmitted. This ability allows programmed I/O to generate some frames with CRC and some without CRC.
- **Zero-Bit Insertion:** The zero-bit-insertion logic, also referred to as data transparency, ensures that the remote receiver does not recognize a flag, mark-idle, or abort embedded in the data. The zero-bit-insertion logic monitors the data stream between the opening and closing flags of a frame and inserts a 0 after detecting five contiguous 1s. Zero-bit insertion does not operate in Transparent mode or when generating flags, mark-idles, or aborts.
- **Transmit Enable:** When transmit is disabled, the transmitter waits for the current frame to complete transmission (if there is one) and for status on that frame to be reported, then sets the transmitter stopped bit and begins transmitting either flags or marks depending on the selected idle condition. While transmit is disabled, the transmitter continues to fill up its internal pipe and FIFO. If the transmit FIFO contains data when transmit is enabled, the transmitter begins transmission within one bit time of when external $\overline{\text{CTS}}$ is asserted. If the idle condition is flag-idle, the transmitter finishes the current flag before starting transmission of data. If the idle condition is mark-idle and at least 16 1s have been transmitted, the transmitter may not finish the current mark idle sequence before starting data transmission. To disable transmit, clear the HTEN bit of the HxTCON0 register.
- **Transmit-FIFO Enable:** Normal operation requires both the Transmit Enable (HTEN) and the Transmit FIFO Enable (TFIFOEN) bits of the HxTCON0 register to be set. Clearing the TFIFOEN bit causes the transmit FIFO data to be flushed. To avoid possible data loss, disable SmartDMA control before flushing the FIFO.
- **Output States:** The serial data output pin on the DCE interface (DCE_TXD_x) supports three-state (reset default), open drain, or totem pole operation under program control. The output must be set to open drain for proper operation in multidrop mode. Specify the output state in the ODRV field of the HxTCON1 register.
- **Transmitter Status:** After transmitting a frame, the transmitter generates a maskable interrupt. If an error occurs during transmission, the transmitter stops. Read the

transmitter status in the FABRST, CTSLST, TUFLO, TGOODF, and TSTOP bits of the HxISTAT0 register.

- **Automatic $\overline{\text{CTS}}$:** When automatic $\overline{\text{CTS}}$ is enabled, the transmitter does not start transmission until $\overline{\text{CTS}}$ is asserted. If the transmitter is transmitting (in-frame) and $\overline{\text{CTS}}$ is deasserted, a lost $\overline{\text{CTS}}$ has occurred. A lost $\overline{\text{CTS}}$ halts transmission and generates an abort and a maskable interrupt. If $\overline{\text{CTS}}$ is deasserted while the transmitter is in idle, the transmitter does not respond. In multiplexed mode, the transmitter ignores $\overline{\text{CTS}}$. If $\overline{\text{CTS}}$ is deasserted in Transparent mode while transmitting, the transmitter begins transmitting idles. When auto-enable $\overline{\text{CTS}}$ is disabled, the transmitter ignores the $\overline{\text{CTS}}$ input. Auto-enable $\overline{\text{CTS}}$ must be disabled for Multidrop mode. To enable automatic $\overline{\text{CTS}}$, set the AUTOCTS bit of the HxTCON1 register to 1.
- **Multidrop Mode with Collision Detection:** This mode requires the transmit data pin to be physically tied, externally, to the $\overline{\text{CTS}}$ input pin. In addition, it requires the mark-idle flag, disabled auto-enable $\overline{\text{CTS}}$, and an open drain output. The HDLC channel delays transmission until it sees a programmable number of consecutive 1s on the $\overline{\text{CTS}}$ input pin. Specify the number of 1s to delay in the TDELAY field of the HxTCON1 register. This feature provides some collision avoidance and a transmit priority based on the number of 1s waited for before transmission. When transmission begins, the transmitter samples the transmit data stream on the $\overline{\text{CTS}}$ input and internally compares it to what is transmitted by the HDLC. Upon detecting a difference, the transmitter generates a maskable interrupt (lost $\overline{\text{CTS}}$), stops the data transmission, starts transmitting idle flags, disables transmit, and flushes the transmit FIFO. To enable multidrop mode with collision detection, set the MLTDRP bit of the HxTCON1 register to 1.

CC

- **GCI D Channel Contention Resolution Request:** The transmitter asserts a signal when it wants to send data. In the Am186CC microcontroller, the GCI controller asserts a signal back indicating when access to the D channel is available. When the GCIDEN bit of the HxTCON1 register is set to 1, the transmitter does not begin transmitting until the GCI gives this indication. If the access signal is deasserted in the middle of transmission, the transmitter immediately transmits an abort, starts transmitting idles, generates a maskable interrupt, and indicates a lost $\overline{\text{CTS}}$ status. At the end of transmission (after the closing flag), the transmitter briefly stops requesting access even if additional frames are to be transmitted. See Chapter 17, “General Circuit Interface (GCI),” for additional information.
- **Transmit Bit Order:** The transmitter supports the option of transmitting data MSB first instead of LSB. To specify MSB-first transmission, set the TMSBF bit in the HxTCON1 register to 1. This ability is typically used only in Transparent mode.
- **Transparent Mode:** The transmitter supports a Transparent mode (set the TRANSM bit of the HxCON register to 1) that transmits the data exactly as it appears in the FIFO. Transparent mode does no bit stuffing, no framing with flags, and does not support CRC. Transparent mode is useful for transmitting raw data streams such as audio data (for use with a codec or DSP). To achieve byte alignment, synchronize the transmitter by resetting the HDLC after configuring the TSA and HDLC. Raw DCE mode does not support byte alignment. Additionally, byte alignment is not possible when the entire time-multiplexed bus is allocated to a single TSA/HDLC channel. To enable Transparent mode, set the TRANSM bit in the HxCON register to 1. Transparent mode is the opposite of data transparency, where zero-bit insertion (bit stuffing) is used to ensure the receiver does not recognize a flag, mark-idle, or abort in the data stream.

Figure 15-4 and Figure 15-5 show a typical transmit with auto-enable $\overline{\text{CTS}}$ enabled. $\overline{\text{CTS}}$ goes active to start the transmission, which begins with a flag. After the flag, three bits of data are transmitted before $\overline{\text{CTS}}$ is recognized as going inactive. This forces TXD High.

Figure 15-6 shows another typical transmit with auto-enable $\overline{CTS}$ enabled. At the end of the closing flag, $\overline{CTS}$ is driven inactive. $\overline{CTS}$ is actually driven inactive at the same time as the last bit of the byte before the flag, but it is not recognized until the next bit; therefore, a lost $\overline{CTS}$ does not occur.

Figure 15-4 $\overline{CTS}$ Controlled Start of Transmit

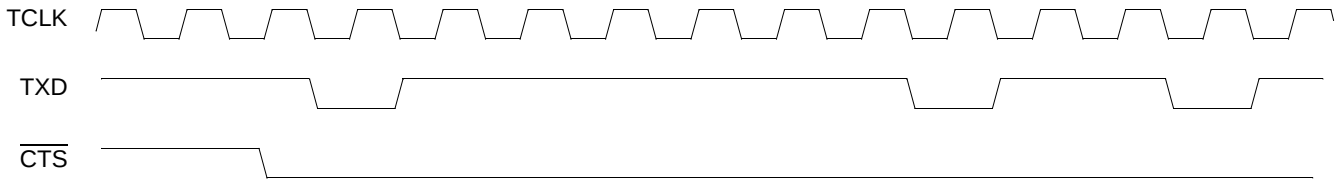


Figure 15-5 $\overline{CTS}$ Controlled End of Transmit

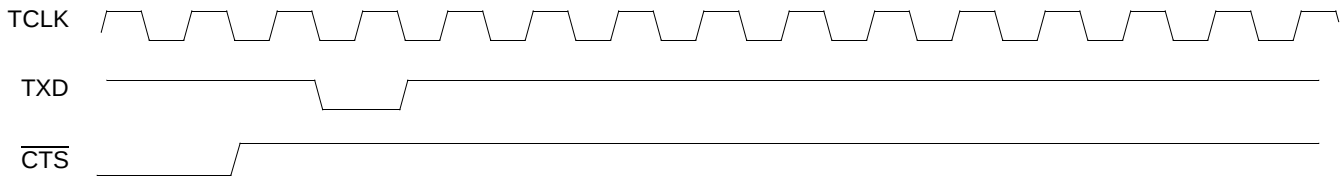
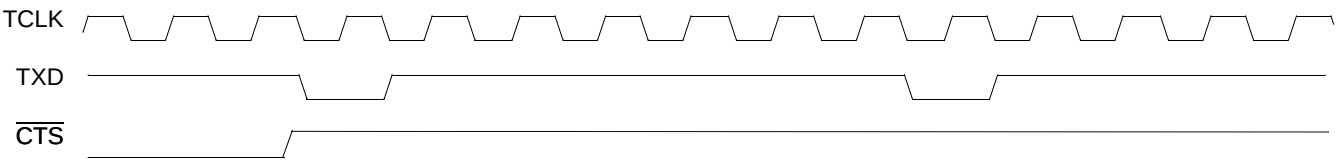


Figure 15-6 $\overline{CTS}$ Inactive at End of Frame



15.5.5 HDLC Receiver

The receiver takes serial data, determines the frame boundaries, and transfers the data to a 32-byte receive FIFO, where it is transferred to memory by the SmartDMA interface or under programmed I/O control. The SmartDMA interface automatically puts the frame status into the buffer descriptors. Programmed I/O puts the frame status into the FIFO at the end of the frame.

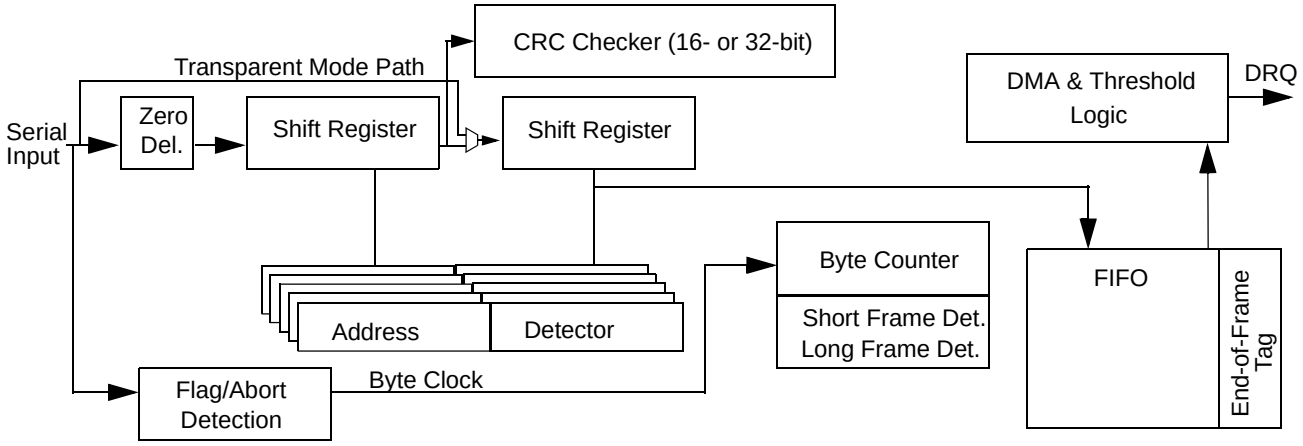
The receiver functions include:

- Mark-idle and flag-idle detection
- Flag/abort recognition
- Zero-bit deletion
- CRC checking
- Address recognition

Figure 15-7 illustrates the block diagram for the receiver.

Note: *The HDLC receiver requires frames two bytes or longer. The HDLC transmitter requires at least one byte of data surrounded by flags: the start flag, one byte of data, and the end flag. A 2-byte CRC with no data also constitutes a valid transmission.*

Figure 15-7 HDLC Receiver Block Diagram



The HDLC receivers have the following features:

- **Flag/Abort Detection:** A flag must be detected before starting to receive a frame. A frame ends and status is reported when the receiver detects a flag or abort sequence. The Am186CC and Am186CH microcontrollers support receiving back-to-back frames with only one flag between frames. In Transparent mode, flag/abort delineation is disabled, and reception begins as soon as the receiver is enabled. Reception continues until the receive FIFO overflows.
- **Zero-Bit Deletion**: Between the opening and closing flags, the receiver removes any 0 that appears after a string of five consecutive 1s (these 0s are added during transmission to prevent a data pattern from resembling an abort, or an opening or closing flag).
- **Receive-Byte Counter:** The receive-byte counter counts the number of bytes received between flags. If the number is less than a 4-bit programmable number, the frame has an error status reported, and part of the frame may be truncated. The receiver rejects very short frames (less than two bytes) and does not put them into the receive FIFO. Short frames (less than the value set in the MINRL field of the HxRCON0 register) and very short frames each generate a separate maskable interrupt. Frames that are truncated due to an abort condition do not count as short or very short frames. If the number of bytes received exceeds a 16-bit programmable number, current frame reception stops, an error status is reported, and the receiver begins to look for a flag. Transparent mode operation disables the receive-byte counter.
- **Receive-Clock Polarity:** The receive-clock polarity is programmable through the RXCINV bit in the HxTCON0 register, independent of the transmit clock polarity. An inverted clock is recommended for use only in DCE mode.
- **Frame Status:** At the end of reception, the receiver places the receive byte counter value and one byte of frame status in the receive FIFO. In programmed I/O mode, the frame status becoming available generates a maskable interrupt, indicated in the REOF bit of the HxISTAT0 register.

- **Short-Frame Counter:** The HxSFCNT and HxSFCNTP registers indicate the total number of short frames received. The HxSFCNT register clears when read; HxSFCNTP does not. This count also includes all very short frames. If the counter rolls over, it generates a maskable interrupt. This count does not include frames with mismatched addresses.
- **CRC Checker:** When the receiver detects the closing flag, it examines the 16-bit (or 32-bit) CRC. If it detects an error, it reports a status bit to that effect. The receiver supports the CRC-CCITT, CRC-16, and CRC-32 algorithms. The receiver always places the CRC in the FIFO along with the rest of the frame data (that is, all data between flags is placed in the FIFO). The CRC checker is always enabled, but software can ignore the CRC error status (byte 3 of the status read from the HxRD register). Specify the CRC type in the CRCTYPE field of the HxCON register.
- **Serial-to-Parallel Shift Register:** Output from the zero-bit-deletion unit feeds into a 16-bit shift register, which converts the serial stream into bytes. The receiver then feeds the parallel output of the shift register to the receive FIFO one byte at a time.
- **Address Detection:** The receiver uses address detection to determine whether to receive the current frame. Each HDLC channel has four 16-bit matching address registers (the HxA0–HxA3 registers) and four corresponding 16-bit matching address mask registers (the HxA0MSK–HxA3MSK registers). The mask register determines which of the first 16 data bits in the frame the receiver should compare to the corresponding address register and which to ignore. If all unmasked bits of at least one address match, the receiver accepts the frame; otherwise, it discards the frame and starts looking for the next flag. The frame status byte contains information about which address matched.
- **Mismatch-Address Counter:** The HxMACNT and HxMACNTP registers keep count of the number of frames that did not have an address match. The HxMACNT register clears when read; HxMACNTP does not. Count rollover generates a maskable interrupt. The receiver checks all frames two bytes or larger for an address match. The receiver does not check the discarded very short frames.
- **Receive FIFO:** The receive FIFO consists of a 32-byte FIFO buffer, end-of-frame logic, and DMA-request logic. Read the receive FIFO at the HxRD register.
- **Receive-FIFO Interface:** The receiver uses either programmed I/O or the DMA controller to unload the receive FIFO. In programmed I/O mode, the RDATA1 bit of the HxISTAT0 register indicates when data is ready to be read from the receive FIFO. Data ready also generates a maskable interrupt. The REOF bit of the HxISTAT0 register (and a maskable interrupt) indicate when the frame status from the last frame received is available to be read from the receive FIFO and data is no longer ready to be read. The next frame data is not available until that status bit is cleared. The SmartDMA interface automatically moves the frame status to the buffer descriptors at the end of the frame.
- **Receive-FIFO Threshold**: The receive FIFO supports thresholds of 1, 8, 16, or 32 bytes under program control. Specify the receive FIFO threshold in the RTHRSH field of the HxRCON0 register. The SmartDMA interface does not move data to memory until the receive FIFO threshold is reached, indicated by the RTHRES bit of the HxISTAT0 register. When the receive FIFO reaches the programmed threshold level, the data ready status stays set until the receive FIFO is empty. At the end of a frame, the receive FIFO outputs the remainder of the frame even if the receive FIFO threshold is not met.
- **Receive-Data Available:** For programmed I/O, the RDATA1 bit of the HxISTAT0 register indicates when there is a data byte presently available in the receive FIFO. This indication

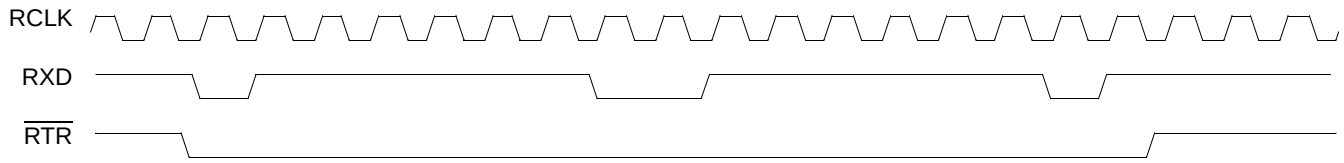
is independent of the threshold selected. The receiver can optionally generate a data-ready interrupt as well.

- **Receive End-of-Frame:** For programmed I/O, the REOF bit of the HxISTAT0 register indicates when any status bytes (that is, an end-of-frame) are present in the FIFO. This indication is independent of the threshold selected. The receiver can optionally generate a status ready interrupt as well.
- **Receive-FIFO Overflow:** If the receive FIFO overflows, it halts reception of the current frame, disables the receiver, deasserts $\overline{\text{RTR}}$, and generates a maskable interrupt. The controller puts the overflow status into the receive FIFO when space is available. The ROFLO bit of the HxISTAT1 register indicates when a receive-FIFO overflow occurs.
- **Bit Residue:** If the number of bits in a frame is not an integer multiple of eight, the receiver rejects the frame and reports the error status in the third status byte read from the HxRD register. The last byte reported of the frame may or may not contain the incomplete last byte of the frame.
- **Receiver Enable:** When the receiver is disabled, the receiver continues to receive the current frame. When the current frame ends (including the closing flag), or immediately if not in-frame, the receiver deasserts the Ready-to-Receive ($\overline{\text{RTR}}$) signal, and generates a maskable interrupt. After the $\overline{\text{RTR}}$ signal is deasserted, the receiver does not receive any data. When the receiver is re-enabled, it asserts the $\overline{\text{RTR}}$ signal. After reasserting the $\overline{\text{RTR}}$ signal, it does not receive any data until it detects a flag and goes to the in-frame state. When the receiver is disabled in Transparent mode, it immediately deasserts the $\overline{\text{RTR}}$ signal and stops reception. When the receiver is enabled in Transparent mode, it immediately asserts the $\overline{\text{RTR}}$ signal and starts reception. Disable the receiver by clearing the HREN bit of the HxRCON0 register to 0.
- **Receive Reject:** When receive reject is enabled, the receiver immediately stops reception of data and reports an error status if the event occurred while in-frame. The $\overline{\text{RTR}}$ signal is not affected. When receive reject is disabled, the receiver starts looking for a flag. To enable receive reject, set the RREJECT bit of the HxRCON0 register to 1.
- **Receiver Stop:** When the receiver is stopped, the receiver immediately stops reception of data and deasserts $\overline{\text{RTR}}$. The receiver also generates an error status if the event occurred while in-frame. To stop the receiver, set the RSTOP bit of the HxRCON0 register to 1.
- **Link Status:** The status of the receiver is reported through the link status. The possible states are: flag idle, mark idle, abort, and in-frame. For each state, the receiver can generate a maskable interrupt when it enters the state. After receiving a flag, a continuous input of 1s goes directly to the mark-idle state without transitioning to the abort state. After exiting reset and a valid state is identified, the receiver always reports the last valid state detected. Read the link status in the RTRS, ABORTS MARKIS, FLAGS, and FRAMES bits of the HxSTATE register. Read the interrupts for these states in the HxISTAT1 register.
- **Receive Bit Order:** The receiver supports the option of receiving data MSB first instead of LSB first. To specify MSB first reception, set the RMSBF bit of the HxRCON0 register to 1. This ability is typically used only in Transparent mode.
- **Transparent Mode:** The receiver supports a Transparent mode that moves the data into the FIFO exactly as it is received with no bit stuffing, flag/abort detection, or CRC support. To achieve byte alignment, synchronize the receiver through an HDLC reset after configuring the Time Slot Assigner (TSA) and the HDLC. Raw DCE mode does not support byte alignment. Additionally, alignment is not possible when the entire time-

multiplexed bus is allocated to a single TSA/HDLC channel. To enable Transparent mode, set the TRANSM bit of the HxCON register to 1.

Figure 15-8 shows the assertion and deassertion of $\overline{\text{RTR}}$ with back-to-back flags. A real frame would contain additional data between the two flags.

Figure 15-8 $\overline{\text{RTR}}$ Timing



15.5.6 HDLC and SmartDMA

All SmartDMA channels support HDLC (the general-purpose DMA channels cannot be used with HDLC). For information about using the SmartDMA interface, see Chapter 8, “DMA Controller.” This section discusses some of the issues with using an HDLC channel as the transmitter and receiver.

15.5.6.1 HDLC Transmitter

The only complication with a normal HDLC transmit using SmartDMA transfer is that, if the packet to be transmitted is composed of more than one buffer and the buffer descriptors are stored from a task that is interruptible, the HDLC transmitter could start up and underflow if the task is interrupted before the last buffer descriptor is updated. To avoid this problem, delay setting the OWN bit in the first buffer descriptor until all the other descriptors have been completely set up, and set up the other descriptors in reverse order, that is, from last to first.

If an error (such as a loss of $\overline{\text{CTS}}$ or a FIFO underflow) occurs during transmission of a packet, the transmitter stops until software clears the error condition in the HDLC controller. Software can either poll for these error conditions or, preferably, set the hardware up to generate an interrupt when they occur. When such an error occurs, the software should take the following steps:

1. Clear the SmartDMA control register TXST bit to stop the DMA. This stops the DMA without clearing the OWN bit on the current buffer descriptor or going to the next buffer descriptor.
2. Clear the HDLC error bit (CTSLST or TUFLO) in the HxISTAT0 register. This automatically flushes and restarts the FIFO and re-arms the interrupt.
3. To resend the same packet that contains more than one buffer, software must:
   a. Read the CTBD register to determine the current descriptor.
   b. Back up to the start of the packet in the descriptor ring (find the descriptor with the STP bit set), setting the OWN bit in each buffer before the current buffer back to and including the buffer with the STP bit set.
   c. Store the number of the descriptor, with the STP bit set, into the CTBD register.

   Note that this technique does not work if buffers are reclaimed by the software as soon as their OWN bits are reset by the hardware. For this technique to work properly, buffers must not be reclaimed until an entire packet has been sent.

4. Finally, set the SmartDMA TXST and POLL bits to restart the DMA and poll the current descriptor. If step 3 was executed to back the DMA to the start of the packet, or if the DMA was already at the start of the packet (e.g., if $\overline{\text{CTS}}$ was lost during transmission of the first buffer in the packet), then the packet is resent. If step 3 was not executed, and the current DMA descriptor does not have STP set, then the DMA controller clears the OWN bit on the descriptor and reads in the next descriptor. The DMA controller repeats this clearing of the OWN bit and stepping to the next descriptor until it encounters a descriptor with the OWN bit clear, or a descriptor with the STP bit set.

   In other words, if the current descriptor is not the first descriptor of a packet (STP bit is 0) and step 3 is not executed, the DMA controller automatically starts up again at the next packet boundary (next buffer with STP set), and it is up to higher-level end-to-end protocols to notice that the current packet was not transmitted successfully and to resend it.

15.5.6.2 HDLC Receiver

Under normal operation, when an HDLC packet is received, the SmartDMA interface stores it in one or more buffers, setting the STP bit in the first buffer descriptor, clearing the status bits in any middle buffer descriptors, setting the EOP (end-of-packet) and error bits, and storing the total length in the last (or only) buffer descriptor.

Software must perform two tasks, which in some systems can be performed at the same time:

- Software must fill the buffer descriptors with pointers to available buffers and information about their size, and set the OWN bits to make them available to the SmartDMA interface. If software is late in performing this task, an RBU interrupt is generated. If software is so late that data is lost, an HDLC ROFLO interrupt is generated. Software does not need to enable these interrupts or poll for this status. If software enables these interrupts, it does not need to take any action in response to the interrupts except to provide buffers to the descriptor ring (and reset the interrupt status bit in order to enable subsequent interrupts of the same kind) because the overflow status is reflected in the next packet stored to the ring.

  If software provides buffers in response to an RBU or HDLC ROFLO interrupt, the software can also set the DMA POLL bit. Setting this bit causes the DMA controller to notice that the OWN bit of the next buffer is set, sooner than the DMA controller may have noticed it on its own. There is never any reason to set the POLL bit for the receive buffer unless the DMA controller run out of empty buffers.

- Software must examine the descriptors of buffers that have been received. Software searches through the descriptor ring until it finds the first descriptor that either has the OWN bit set, or has the EOP bit set, or until it gets to the last descriptor that it has made available to the hardware. When software finds a descriptor with the OWN bit reset and the EOP bit set, it knows it has found the end of a packet. Software then moves the descriptors off the ring, and sends the buffers to a higher-level task. If the error bits are set in the descriptor with the EOP, software could simply recycle the buffers to the next free position in the ring, without sending them to the next layer.

15.5.7 Interrupts

All interrupts are individually maskable. Set the status bits in the HxISTAT0 and HxISTAT1 registers. Mask interrupts in the HxIMSK0 and HxIMSK1 registers.

15.5.7.1 Transmit Interrupts

The microcontroller provides the following transmit interrupts:

- Transmit threshold reached
- Data byte available
- Abort sent
- Lost $\overline{\text{CTS}}$
- Transmit FIFO underflow
- Good frame transmitted
- Transmitter stopped

15.5.7.2 Receive Interrupts

The microcontroller provides the following receive interrupts:

- Receive threshold reached
- Receive status available
- Data byte available
- Short frame counter rollover
- Mismatch address counter rollover
- Receive FIFO overflow
- Flag idle state entered
- Mark idle state entered
- Abort state entered
- In-frame state entered
- $\overline{\text{RTR}}$ deasserted
- Short frame detected
- Very short frame detected

15.5.8 Hardware-Related Considerations

- The Receive Threshold (RTHRSH) bits in the HDLC Channel Receive Control 0 (HxRCON0) register specify the amount of data needed in the receive FIFO before giving an interrupt. The possible values are: 1, 8, 16, or 32 bytes. The Receive FIFO Threshold Reached (RTHRES) bit in the HxISTAT0 register can generate an interrupt if this programmed threshold has been reached or exceeded in the receive FIFO and there were no status bytes present in the FIFO at that time. When RTHRES is set by hardware, the receiver reads the threshold value from the FIFO without rechecking the status. If a status byte is put into the FIFO after the RTHRES bit is set, after the first read of the FIFO the receiver clears the bit, even if there is still a threshold number of data bytes in the receive FIFO. Therefore, when the RTHRES bit is set, the receiver can read the threshold number of data bytes consecutively.

- The receiver sets the One Receive Data Byte Available (RDATA1) bit in the HDLC Channel Interrupt Status 0 (HxISTAT) register when the *current* byte available is data; the RDATA1 bit does not reflect the entire FIFO contents. If the next byte is status and the following byte is data, the receiver does *not* set RDATA1.

15.5.9 Software-Related Considerations

- After setting the HREN bit to enable the receiver, the device software must reset the HDLC FIFOs by setting the HRESET bit in the HxCON register. This clears any invalid data in the receive FIFO that might be mistaken as the start of the data stream. Invalid data is a concern when using Transparent mode (TRANSM = 1 in the HxCON register), because in Transparent mode the receiver cannot rely on flag sequences to indicate the start of valid data.
- When the HDLC channel is disabled, the FIFO status reads as full.
- CC In the Am186CC microcontroller, HDLC Channel D is multiplexed with the UART and with flow control on the High-Speed UART. The Interface 4 Select (ITF4) bits in the System Configuration (SYSCON) register must be configured for the HDLC interface.

15.5.10 Comparison to Other Devices

In addition to HDLC, the HDLC channels support the SDLC, LAP-B, LAP-D, PPP, and v.120 communications protocols. The HDLC channels can also be used in transparent mode to support the v.110 protocol.

The HDLC protocol is similar to these other bit-oriented protocols:

- The Advanced Data Communication Control Procedures (ADCCP) developed by the American National Standards Institute (ANSI X3.66) is virtually identical to the HDLC protocol.
- The Link Access Procedure Balanced (LAP-B), adopted by the International Telegraph and Telephone Consultative Committee (CCITT) as part of its X.25 packet-switched network standard, is a subset of HDLC.
- Although not a standard, IBM's Synchronous Data Link Control (SDLC) is in widespread use. SDLC is a subset of HDLC, with some differences.

15.6 INITIALIZATION

On both external and internal reset, the following occurs:

- The multiplexed HDLC signals default to the signals shown in Table 15-1 on page 15-4.
- All HDLC registers default to 00h except the HxSTATE, HxTD, HxRD/HxRDP, and HxRFSx registers.
- CC The ITF4 bit in the SYSCON register is cleared, which defaults external interface D to HDLC with flow control.
- CC The EXSYNC bit in the SYSCON register is cleared, which configures HDLC Channel C for raw DCE or PCM highway modes.

CHAPTER

16 HDLC EXTERNAL SERIAL INTERFACE CONFIGURATION (TSAS)

CC CH ***Note:*** *Only the Am186CC and Am186CH microcontrollers support the TSAs.*

16.1 OVERVIEW

Time Slot Assigners (TSAs) and muxing logic between the HDLC channels and the external communications interfaces of the chip provide flexible data path control on the Am186CC and Am186CH microcontrollers. This data path control, combined with flexible time slot allocation, allows the microcontroller's external data streams to take on a wide variety of forms.

CC The Am186CC microcontroller supports raw DCE, PCM Highway, and General Circuit Interface (GCI) external data streams.

CH The Am186CH HDLC microcontroller supports raw DCE and PCM Highway external data streams.

CC In the Am186CC microcontroller, interface A not only allows for a dedicated DCE/PCM HDLC path, but has the capability to multiplex GCI/PCM data from each of the remaining nondedicated HDLC channels.

Depending on the application, each HDLC can communicate to the external world with or without a TSA. Each TSA resides between a PCM Highway internal bus and an individual HDLC channel. A TSA's main function is to allow the transmission and reception of data to and from an individual HDLC by providing the appropriate HDLC clock and clock enable signals during its programmed time slot within an 8-KHz frame.

In *nonmultiplexed mode* (there is no time-division multiplexing), an individual external serial bus interface connects directly to an individual HDLC for both transmission and reception. Configuring the microcontroller's muxing logic for a specific raw DCE data path uses nonmultiplexed mode.

In *multiplexed mode* (there is time-division multiplexing), all HDLC data that enters or leaves the microcontroller passes through a TSA. Configuring the microcontroller's muxing logic and TSAs for the multiplexed PCM Highway uses multiplexed mode. External interface A is unique in that it allows multiple time slots, to and from each HDLC, to multiplex on and off this single interface.

CC Configuring the Am186CC microcontroller's muxing logic and TSAs for the GCI data path also uses multiplexed mode.

Time Slot selection allows up to 156 8-bit time slots within a time-division multiplexed (TDM) frame. Each TSA channel can support a burst data rate to or from the HDLC of up to 10 Mbit/s in DCE and PCM highway modes. In all modes of operation, each channel is capable of supporting full-duplex communications. With a maximum data rate of 10 Mbit/s and an 8-KHz frame, each channel provides programmability to support a maximum of 156 time slots per TDM frame. (Although the microcontroller supports up to 4096 bit positions, this requires a lower frame synchronization (frame sync) or a higher, unguaranteed clock rate.)

CC

In the Am186CC microcontroller, Time Slot selection also supports isolation of GCI B and D channels on separate HDLC channels. Each TSA channel can support a burst data rate to or from the HDLC of up to 768 Kbit/s in GCI mode.

The TSA controllers also generate control signals for programmable frame sync pulse polarity and individual channel time slot control output, which is asserted for the duration of the programmed time slot(s). The latter is routed externally for PCM Highway applications and can be used in subscriber linecard applications where it is used as an enable for three-state data buffering on the PCM Highway.

The TSA controllers support adjustable channel sizing with the ability to define time slot start and stop points. This channel adjustment and placement feature is an essential factor for the creation of a GCI frame. The adjustable sizing feature also allows the associated HDLC channel to be used for ISDN LAP-D and reduced data mode X.25 LAP-B transmissions.

For applications that do not use the entire allocated time slot but do require a defined polarity for the remaining unused bit positions, the TSA controllers provide the option of adding additional polarity bits (up to seven) to fill out the remaining bit positions.

16.2 BLOCK DIAGRAMS

Figure 16-1 and Figure 16-2 show simplified block diagrams for the TSA muxing. Figure 16-3 shows a block diagram for a single HDLC channel, including connections with the TSA and GCI.

Figure 16-1 Block Diagram For TSA Multiplexing (Am186CC Communications Controller)

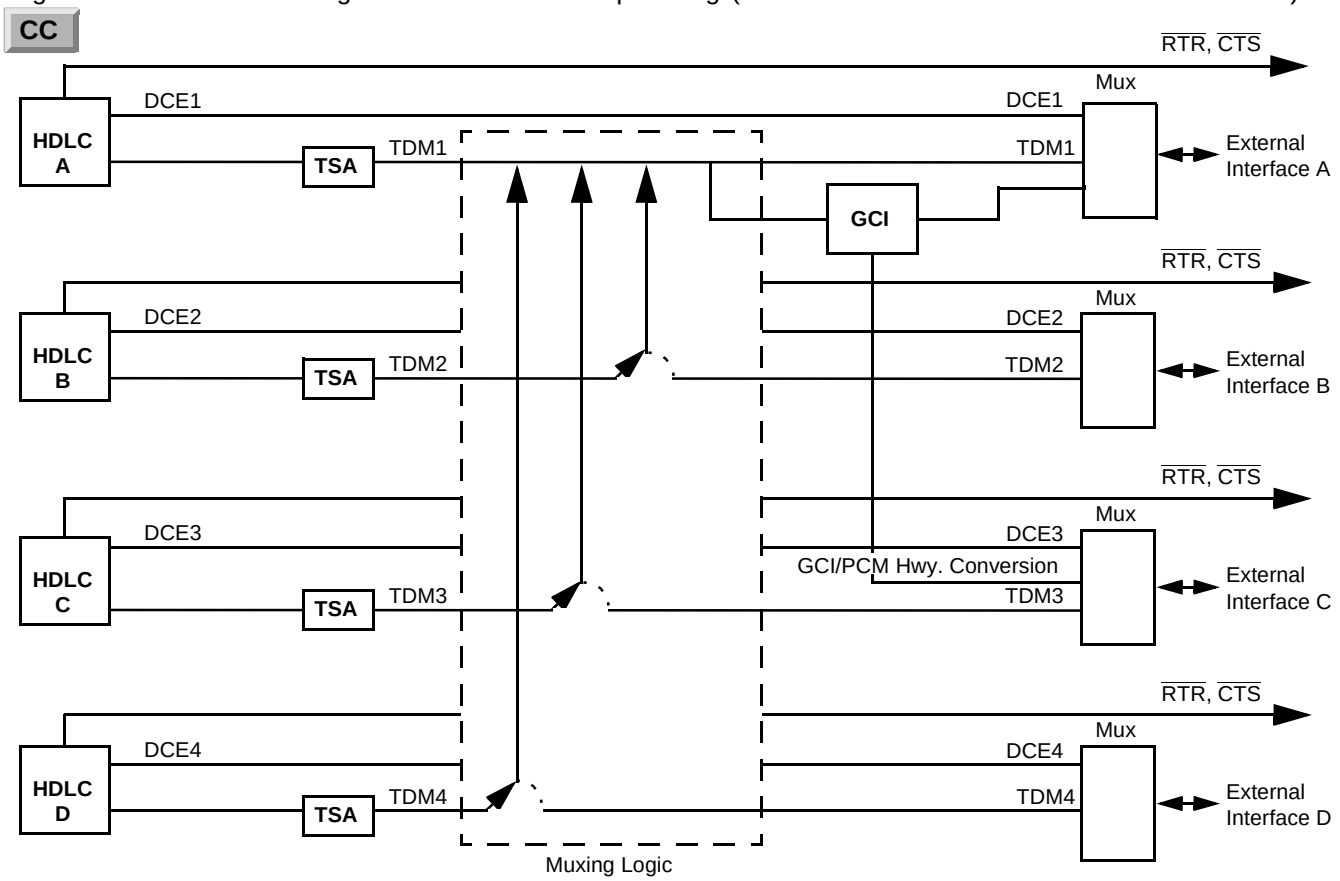


Figure 16-2 Block Diagram For TSA Multiplexing (Am186CH HDLC Microcontroller)

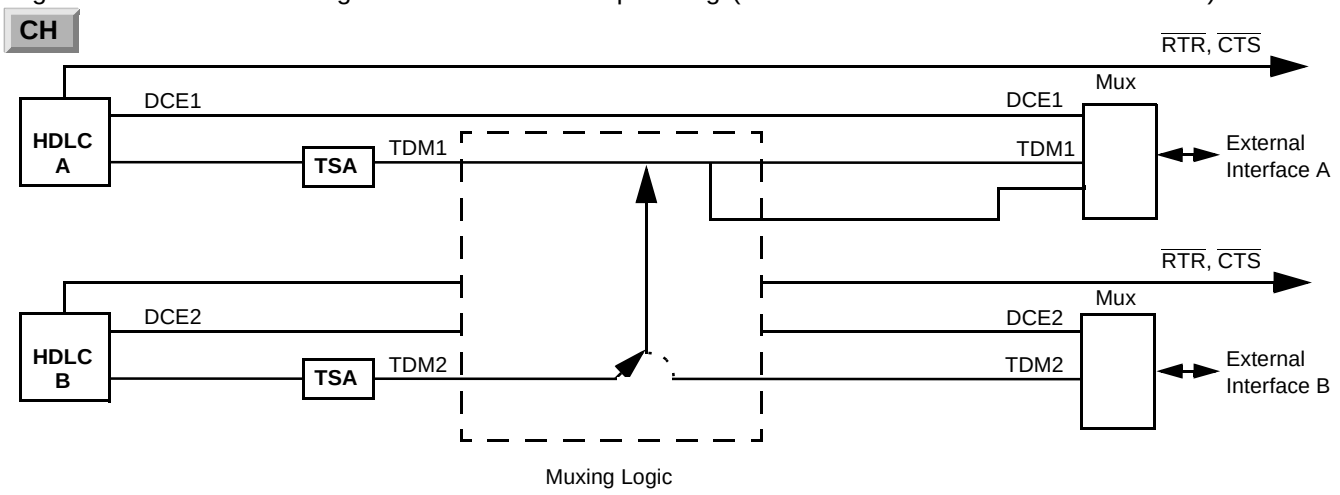
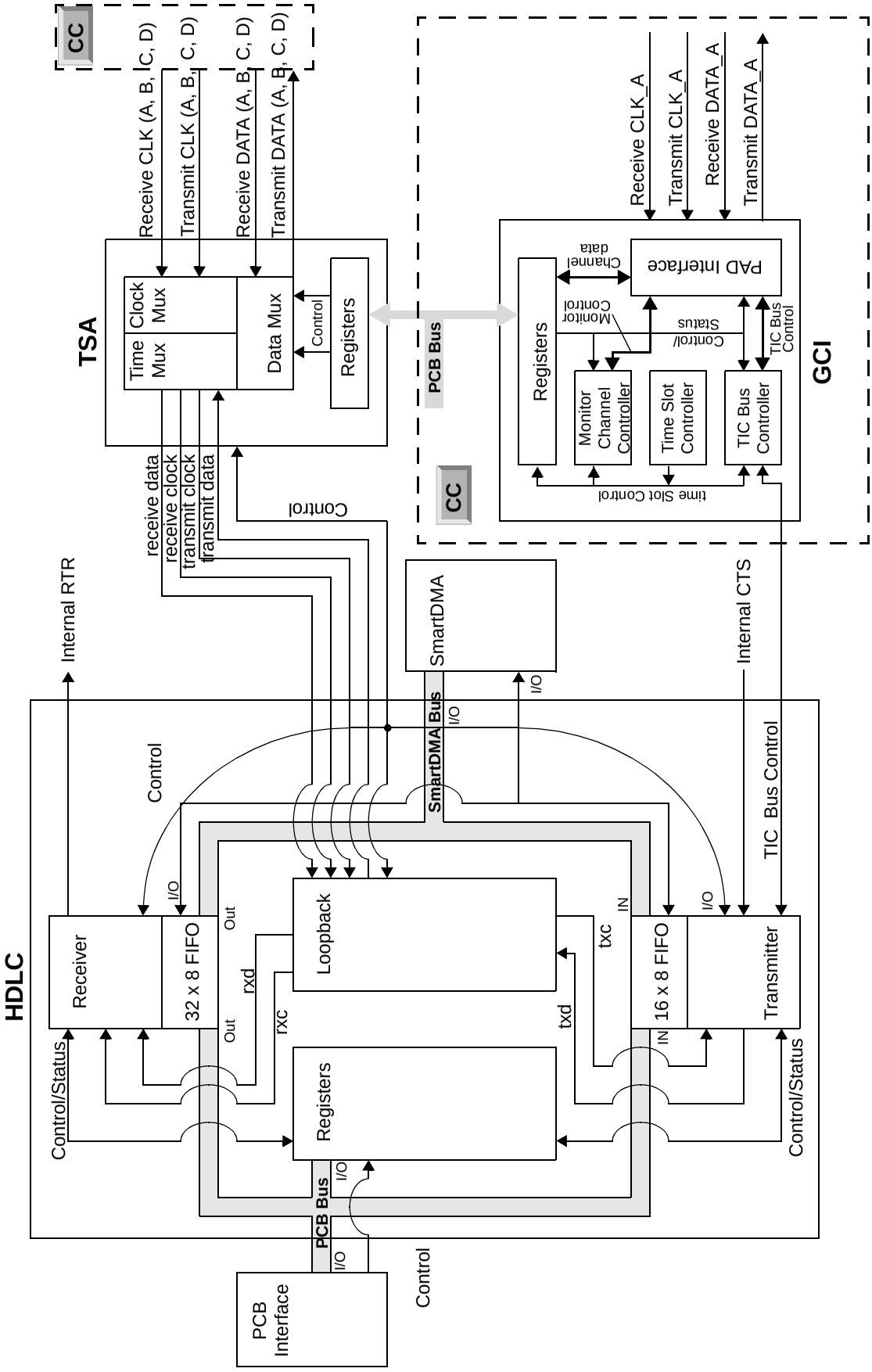


Figure 16-3 HDLC, TSA, and GCI Block Diagram (Same as Figure 15-2)



16.3 SYSTEM DESIGN

lists the signals that are multiplexed with other microcontroller functions. Pinstraps are sampled only at external reset and do not affect the pin's other functions, so they are not shown in this table. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin. Figure 16-4 illustrates an example application.

Figure 16-4 ISDN PCM System Application Example

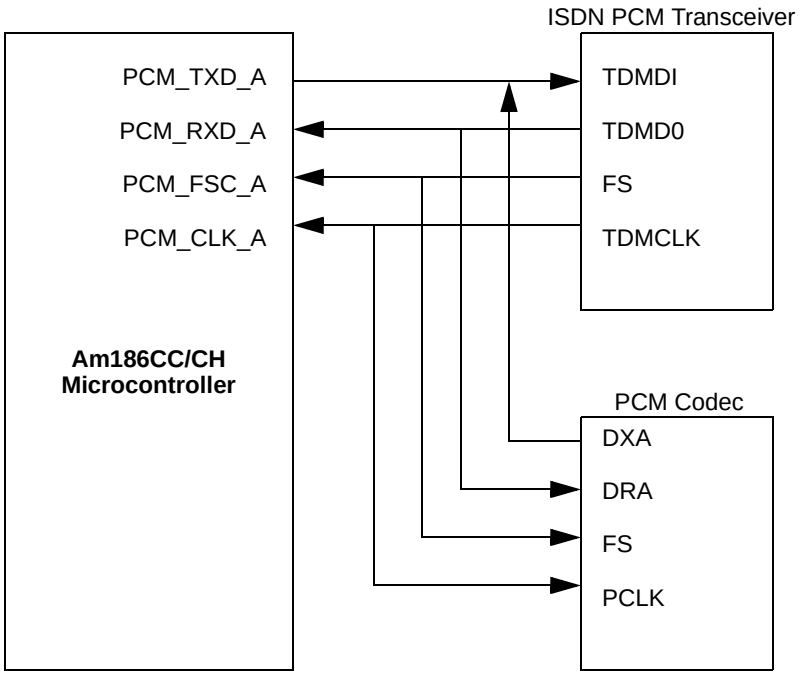


Table 16-1 HDLC/TSA/GCI Multiplexed Signals (Same as Table 15-1)

Multiplexed Signals						Function	Default Signal
Ch	External Interface				PIOs		
	DCE	PCM	GCI CC	UART			
A	DCE_RXD_A	PCM_RXD_A	GCI_DD_A	—	—	DCE and PCM data input/ GCI downstream pin	DCE_RXD_A
	DCE_TXD_A	PCM_TXD_A	GCI_DU_A	—	—	DCE and PCM data output/ GCI upstream pin	DCE_TXD_A
	DCE_RCLK_A	PCM_CLK_A	GCI_DCL_A	—	—	DCE receive clock/PCM receive and transmit clock/GCI receive and transmit clock	DCE_RCLK_A
	DCE_TCLK_A	PCM_FSC_A	GCI_FSC_A	—	—	DCE transmit clock/PCM frame sync clock/GCI frame sync clock	DCE_TCLK_A
	$\overline{\text{DCE\_CTS\_A}}$	$\overline{\text{PCM\_TSC\_A}}$	—	—	PIO17	DCE clear to send/PCM external buffer enable	PIO17
	$\overline{\text{DCE\_RTR\_A}}$	—	—	—	PIO18	DCE ready to receive	PIO18

Table 16-1 HDLC/TSA/GCI Multiplexed Signals (Same as Table 15-1) (Continued)

Multiplexed Signals						Function	Default Signal
Ch	External Interface				PIOs		
	DCE	PCM	GCI CC	UART			
B	DCE_RXD_B	PCM_RXD_B	—	—	PIO36	DCE and PCM data input pin	PIO36
	DCE_TXD_B	PCM_TXD_B	—	—	PIO37	DCE and PCM data output pin	PIO37
	DCE_RCLK_B	PCM_CLK_B	—	—	PIO40	DCE receive clock/PCM receive and transmit clock	PIO40
	DCE_TCLK_B	PCM_FSC_B	—	—	PIO41	DCE transmit clock/PCM frame sync clock	PIO41
	$\overline{\text{DCE\_CTS\_B}}$	$\overline{\text{PCM\_TSC\_B}}$	—	—	PIO38	DCE clear to send/PCM external buffer enable	PIO38
	$\overline{\text{DCE\_RTR\_B}}$	—	—	—	PIO39	DCE ready to receive	PIO39
C CC	DCE_RXD_C	PCM_RXD_C	—	—	PIO42	DCE and PCM data input pin	PIO42
	DCE_TXD_C	PCM_TXD_C	—	—	PIO43	DCE and PCM data output pin	PIO43
	DCE_RCLK_C	PCM_CLK_C	PCM_CLK_C	—	PIO22	DCE receive clock/PCM receive and transmit clock input/GCI-to-PCM conversion clock output	PIO22
	DCE_TCLK_C	PCM_FSC_C	PCM_FSC_C	—	PIO23	DCE transmit clock/PCM frame sync clock input/GCI-to-PCM conversion frame sync output	PIO23
	$\overline{\text{DCE\_CTS\_C}}$	$\overline{\text{PCM\_TSC\_C}}$	—	—	PIO44	DCE clear to send/PCM external buffer enable	PIO44
	$\overline{\text{DCE\_RTR\_C}}$	—	—	—	PIO45	DCE ready to receive	PIO45
D CC	DCE_RXD_D	PCM_RXD_D	—	RXD_U	PIO26	DCE and PCM data input/ UART data receive	PIO26
	DCE_TXD_D	PCM_TXD_D	—	TXD_U	PIO20	DCE and PCM data output/ UART data transmit	PIO20
	DCE_RCLK_D	PCM_CLK_D	—	RTR_U	PIO25	DCE receive clock/PCM receive and transmit clock input/UART ready-to-receive	PIO25
	DCE_TCLK_D	PCM_FSC_D	—	CTS_U	PIO24	DCE transmit clock/PCM frame sync clock input/UART clear-to-send	PIO24
	$\overline{\text{DCE\_CTS\_D}}$	$\overline{\text{PCM\_TSC\_D}}$	—	CTS_HU	PIO46	DCE clear to send/PCM external buffer enable/High-Speed UART clear-to-send	PIO46
	$\overline{\text{DCE\_RTR\_D}}$	—	—	RTR_HU	PIO47	DCE ready to receive/High-Speed UART ready-to-receive	PIO47

16.4 REGISTERS

Table 16-2 lists the three unique registers that program each individual TSA. The x shown in the register name is A, B, C, or D, depending on the channel selected. The offset shown is for Channel A; for Channel B, add 08h to the offset. Both the Am186CC and Am186CH microcontrollers support Channels A and B.

CC

The Am186CC microcontroller also supports Channels C and D. Add 10h to the offset for Channel C and add 18h for Channel D.

Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 16-2 TSA Register Summary

Offset[1]	Register Mnemonic	Register Name	Description
2C0h	TSxCON	TSA Channel Configuration	Configures and enables the TSA channel.
2C2h	TSxSTART	TSA Channel Bit Start Position	Defines the time slot start position for transmit and receive data frames.
2C4h	TSxSTOP	TSA Channel Bit Stop Position	Defines the bit stop position for transmit and receive data frames.

Notes:
1. The x *shown in the register name can be A, B, C, or D, depending on the channel selected. The offset shown is for Channel A; for Channel B, add 08h to the offset; Channel C, add 10h; Channel D, add 18h. Both the Am186CC and Am186CH microcontrollers support Channels A and B. The Am186CC microcontroller also supports Channels C and D.*

16.5 OPERATION

16.5.1 Usage

Note: *Before using the TSA channels, ensure multiplexed pins are configured to reflect the use of the external interface desired and not other functionality (see Table 16-1 on page 16-5).*

Configure the Time Slot Assigner (TSA) controllers using the following process:

1. Define the bit start position for the transmitted or received data frame for each specific TSA channel in the TSA Channel Bit Start Position (TSxSTART) register.
2. Define the bit stop position for the transmitted or received data frame for each specific TSA channel in the TSA Channel Bit Stop Position (TSxSTOP) register.
3. Configure the operating modes for each specific TSA channel in the TSA Channel Configuration (TSxCON) register—channel mode, channel frame sync pulse polarity, and channel adjust bit drive level—and enable each TSA channel by setting the EN bit to 1. These bits may be set simultaneously but EN cannot be set before steps 1–2.

   The TSAs are now enabled for data transfers. For information about configuring HDLC channels to begin transferring the data, see “Usage” on page 15-7.

CC

For information about using GCI, see “Usage” on page 17-5.

4. To establish byte alignment in transparent mode, additional configuration is necessary as follows:

   a. Enable transmit FIFO (TFIFOEN bit), force abort (FORABR bit), and transmit enable (HTEN bit) of the HxTCON0 register for each specific HDLC channel.

   b. Enable receiver (HREN bit) of the HxRCON0 register for each specific HDLC channel.

   c. Toggle (set, then clear) HDLC reset (HRESET bit) of the HDLC channel control register (HxCON) for each specific HDLC channel.

   d. Clear HxISTAT0 and HxISTAT1 registers for each specific HDLC channel.

The first byte received or transmitted may be corrupted while the HDLC is performing the alignment. This effect can be masked on the transmit side by configuring the transmitter to use mark idles and making the first byte transmitted all 1s (FFh).

16.5.2 Programmable Time Slots

Each TSA is unique, and time slots can start and stop on any bit boundary within a time-division multiplexed (TDM) frame, up to a maximum of 4096 bit positions. Frame boundary overlapping is allowed and occurs whenever the programmed bit start point exceeds the bit stop point. The microcontroller supports the isolation of 8-bit time slots from 0 to 155 on a standard 8-KHz TDM frame (this limitation is due to the 10-MHz limitation of HDLC).

The ability to define time slot start and stop points allows for adjustable channel sizing and placement. Adjustable channel sizing enables the TDM data channel to or from an individual HDLC to support differing data rates.

CC

In the Am186CC microcontroller, the channel adjustment and placement feature is an essential factor for the creation of a GCI frame. In GCI applications, the GCI D channel must be size-adjusted to two bits and the GCI B channels must be size-adjusted to eight bits (see Chapter 17, “General Circuit Interface (GCI),” for further information regarding B and D channel size and placement).

The adjustable sizing feature also allows the HDLC channel to be used for ISDN LAP-D and reduced data mode X.25 LAP-B transmissions such as 56 Kbit/s.

Certain applications do not use the entire allocated time slot, but do require a defined polarity for the remaining unused bit positions. For these applications, the user is given the option of adding additional polarity bits (up to seven) to fill out the remaining bit positions. In short, the user must program a start point, a stop point, the number of bits remaining to complete the allocated time slot, and a polarity for the remaining unused bit positions.

Note: *Since a maximum clock rate of 10 MHz is supported, the utilization of the full 4096 bit range is sync rate dependent. For example, the standard 8-KHz frame does not support bit ranges above 1250 bit positions or 156 8-bit time slots (this requires a clock rate higher than 10 Mhz). Applications capable of supporting lower sync rate frequencies can use the full bit range.*

16.5.3 Muxing Logic

CC

For the most part, the muxing logic controls the path data takes from an HDLC to an external communication interface (or vice versa). The exception to this can be seen in the last mux stage on interface C in the Am186CC microcontroller. Here, one of the mux options provides an adjusted GCI clock and frame sync source for external interface C. For more information, see “GCI Frame Sync and Clock Conversion” on page 16-12.

Figure 16-5 on page 16-10 demonstrates the muxing logic for an ISDN basic-rate GCI interface. The muxes at each stage level have been removed for clarity. In their place is the end data path established after proper mux initialization. This figure illustrates the following:

1. Adjustable time slot size: eight bits for each GCI B channel and two bits for the GCI D channel
2. Isolation of single time slots 0, 1, and 3
3. GCI B and D channel isolation
4. GCI support
5. Multiplexed mode for interface A (where multiple HDLC channels are multiplexed onto one line)

Depending on whether you are transmitting or receiving, Figure 16-5 on page 16-10 can be read: Stage 1, Stage 2, Stage 3; or Stage 3, Stage 2, Stage 1.

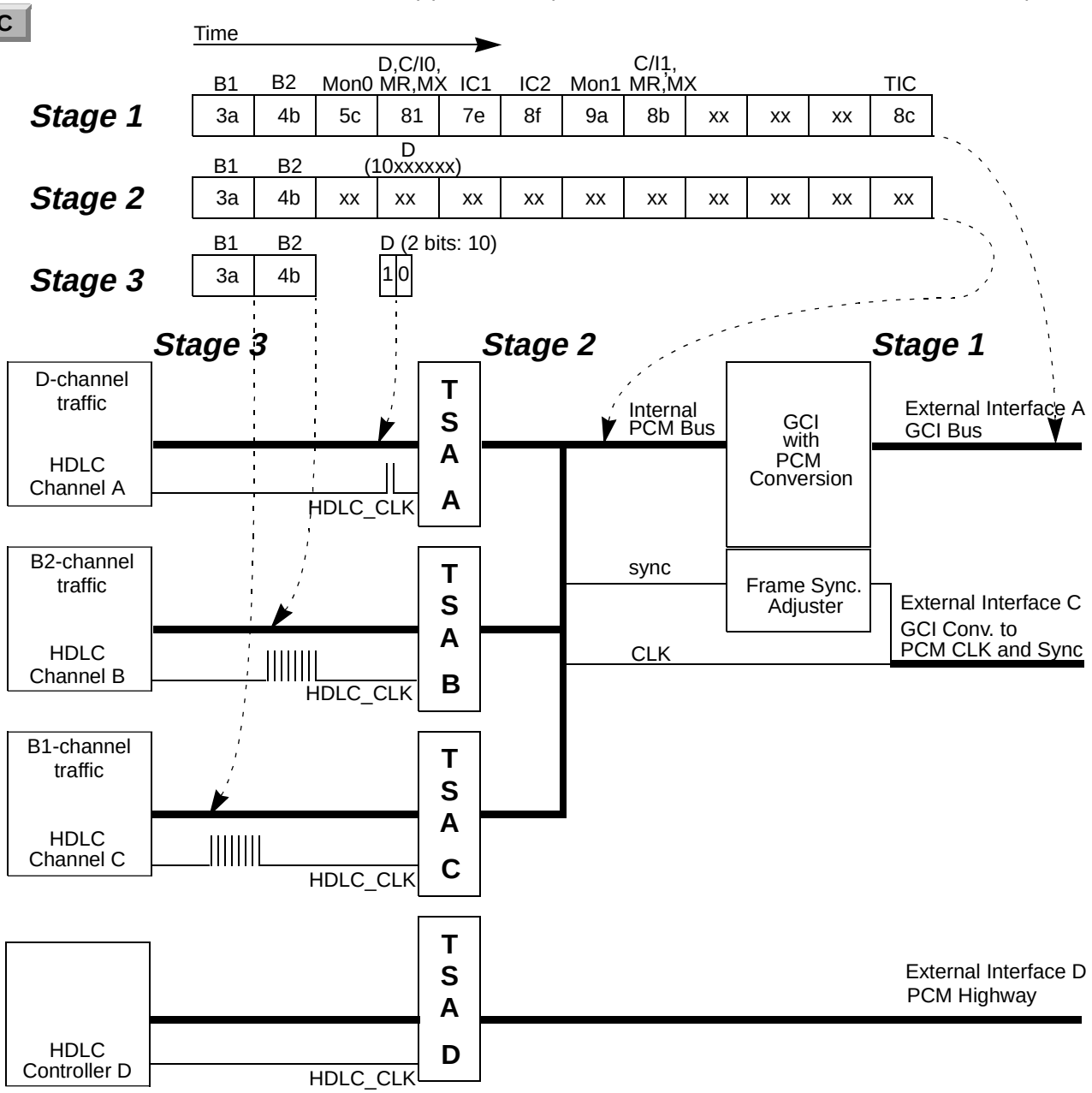
In Stage 1, the GCI controller extracts the GCI Monitor (Mon), Command/Indicate (C/I), Intercommunication (IC), and Terminal Interchip Communication (TIC) channels. At the end of stage 1, the HDLC data is multiplexed with the GCI channel data (the B1-, B2- and D-channel ISDN data is present as well as the GCI Mon, C/I, IC, and TIC data).

In Stage 2, all channels are logically muxed onto one internal bus heading to and from interface A. The GCI B and D channels are isolated.

In Stage 3, each HDLC clock is only active during the time slot for the channel it is to transmit or receive. TSA A is configured to enable HDLC clocks for GCI channel D data. TSA B is configured to enable clocks for GCI B2 channel data. TSA C is configured to enable clocks for GCI B1 channel data.

Figure 16-5 ISDN Basic-Rate GCI Application (Am186CC Communications Controller)

CC



16.5.4 External Interfaces

As mentioned previously, the Am186CC and Am186CH microcontrollers' external data streams can take the following forms: raw DCE and PCM Highway. When connecting directly to an individual HDLC, raw DCE format is available. When HDLC data passes through a TSA, the PCM Highway interface is available.

CC

In addition, the Am186CC microcontroller supports the GCI external data stream when HDLC data passes through a TSA.

16.5.4.1 Raw DCE

Raw DCE is a synchronous serial bus generally used in modem and other high-speed serial applications, and runs at up to 10 Mbit/s. The Am186CC and Am186CH microcontroller implementation requires transmit (TCLK) and receive (RCLK) clock inputs, has receive (RXD) and transmit data (TXD), and the Clear-To-Send ($\overline{CTS}$) and Ready-To-Receive ($\overline{RTR}$) flow control signals.

16.5.4.2 PCM Highway

PCM Highway is a generic serial bus used to support a wide range of data rates (including E1/T1) and runs at up to 10 Mbit/s. The Am186CC and Am186CH microcontroller implementation is composed of data transmit (TXD), data receive (RXD), data clock (CLK), frame sync clock (FSC), and time slot control ($\overline{TSC}$) signals.

Each of the individual PCM Highway interfaces are pin-multiplexed with one or more of the following serial bus interfaces: raw DCE and High-Speed UART.

CC

In the Am186CC microcontroller, the individual PCM Highway interfaces are also pin-multiplexed with GCI. A converted GCI frame sync and clock interface for the PCM codecs is also multiplexed with one of the four PCM Highway interfaces. For a listing of all the pin multiplexing, see Table 16-1 on page 16-5.

PCM channel configuration (e.g., channel size, channel length, channel placement, etc.) is provided through proper TSA initialization.

16.5.4.2.1 PCM Highway Applications

The PCM Highway implementation features the following:

- Every TSA can support a separate PCM physical interface simultaneously.
- Each PCM interface is pin-multiplexed with other serial bus interfaces.
- All of the HDLC channels can be routed to PCM Highway interface A.
- Each HDLC channel supports PCM channel time slot selection, fully configurable through the integrated TSAs.

CC

- GCI clock and frame synchronization conversion and routing directly from the GCI to the PCM Highway interface are supported for external codec applications. Data is routed externally (with respect to the Am186CC microcontroller) directly from the codec to the GCI transceiver device for this type of application.
- Support for a time slot control signal that asserts for the duration of the programmed time slots.

CC

- Targets the following external codecs (see Table 16-3 on page 16-14):
  - AMD Am79C02/03
  - AMD Am79C031
  - Motorola MC14555x
  - National TP305x family
  - National TP307x family
  - AT&T T75xx family
  - TI/Intel 291x family

CC

- Targets the following external ISDN transceivers (ISDN requires three channels):
  - AMD Am79C30A/32A S/T
  - Lucent T7237 U

CC

Note: *The Am186CC microcontroller does not provide the PCM codec master clocks for GCI applications.*

16.5.4.2.2 GCI Frame Sync and Clock Conversion

CC

To support a wide variety of external PCM codecs while in GCI mode, the microcontroller divides down the GCI clock frequency (which is twice the GCI data rate) to match the PCM data rate. In addition to a divided-down clock, a programmable one-clock-prior-to-data frame sync (the programmability determines where the frame sync appears relative to a B channel time slot) provides the needed flexibility to support the targeted PCM codecs (listed previously). These two converted signals (converted GCI frame sync and converted GCI clock) are an output on the Am186CC microcontroller's external communication interface C. All codec data movement is completely external to the microcontroller and is directly routed between the PCM codec and the GCI bus. All external PCM codecs directly connected to the GCI bus must meet all of the following conditions:

1. Be configured to output data on the rising edge of the converted clock and input data on the falling edge of the converted clock (for GCI bus compatibility).
2. Be able to accept a one-clock-prior-to-data frame sync.
3. Be able to accept an active High frame sync.
4. Be capable of accepting a data clock frequency of 768 KHz.

Note: *The Am186CC microcontroller only provides the PCM codec data/bit clock. The Am186CC microcontroller does not provide the PCM codec master clock(s).*

Figure 16-6 and Figure 16-7 illustrate both frame sync programmability and GCI conversion, respectively.

Figure 16-6 Programmable Frame Sync

CC

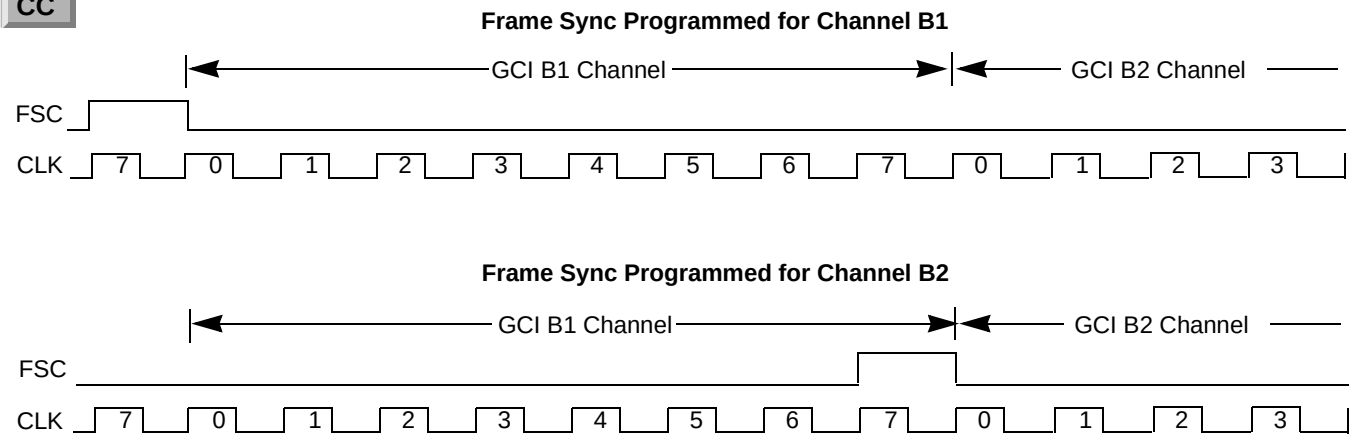


Figure 16-7 Converted GCI Clock and Frame Sync

CC

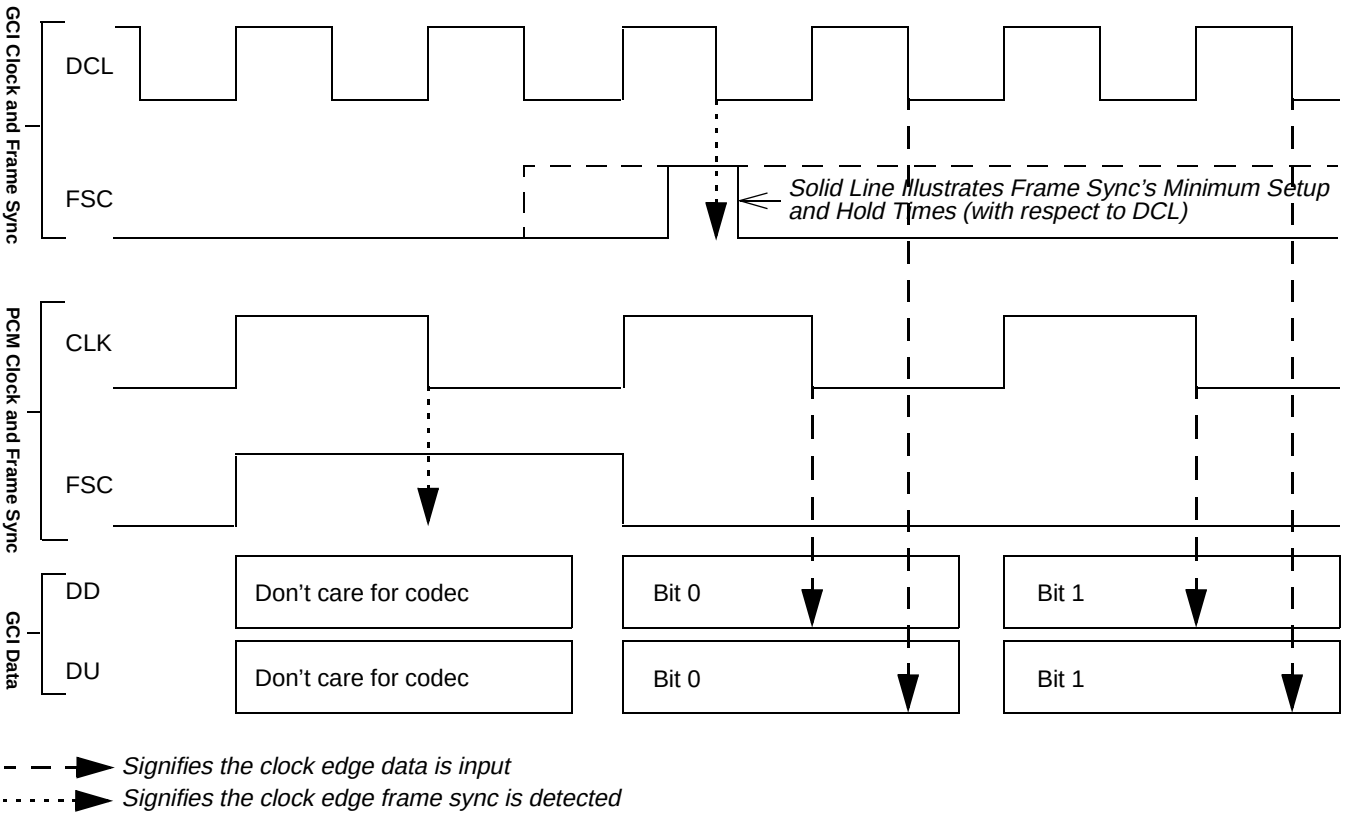


Table 16-3 Timing Parameters Per Device (Supported PCM Codecs in GCI Mode)

CC

Parameter[1]	Device Time (in ns)					
	79C30 as Master	Am186CC	Am79C02/ 03/031	MC14555 x	TP305x	TP307x
Clock Period	Min: 487 Max: 815	Min: 974 Max: 1630	Min: 122 Max: 7812			
Clock High Pulse Width	Min: 260	Min: 260	Min: 48 Max: 3890	Min: 50	Min: 160	Min: 80
Clock Low Pulse Width	Min: 260	Min: 260	Min: 48 Max: 3890	Min: 50	Min: 160	Min: 80
Frame Sync Setup Time	Min: 50	Min: 50	Min: 25 Max: clk prd-50	Min: 50	Min: 50	Min: 30
Frame Sync Hold Time	Min: 50	Min: 260 (min PW)[2]	Min: 50	Min: 50	Min: 100	Min: 30
Data Output Delay	Max: 100	Data movement occurs out-side the Am186CC	Min: 3/30ns Max: 80/150	Min: 20 Typ: 60 Max: 140	Min: 0 Max: 140	Max: 80
Data Output Hold Time	Min: 70		Min: 5/30 Max: 80/150			Max: 80
Data Input Setup Time	PW + 20		Min: 25	Min: 0	Min: 50	Min: 30
Data Input Hold Time	50		Min: 5	Min: 50	Min: 50	Min: 15

Notes:
1. All loading is 150 pF.
2. One clock prior to frame sync is a full clock period, guaranteed to hold for a minimum pulse width Low.

16.5.4.3 GCI

CC

The Am186CC microcontroller supports GCI, which is an industry-standard serial bus for interconnecting telecommunications integrated circuits. For more information, see Chapter 17, “General Circuit Interface (GCI).”

16.5.5 Software-Related Considerations

- When using the TSAs in Transparent PCM Highway mode, the first byte of data transferred must always contain a 1 as the first bit. If the first bit is a 0, the idle state (mark idle) previous to the actual data stream is corrupted, which could falsely indicate the start of actual data. This only applies when using Transparent mode.

16.5.6 Comparison to Other Devices

The Am186CC and Am186CH microcontrollers are similar to the AMD Am79C30 in clock slave mode.

16.6 INITIALIZATION

On both external and internal reset, the following occurs:

- All the TSA registers default to C0h, which disables the TSA channels (they must be configured by software before being enabled).
- The multiplexed signals default as shown in Table 16-1 on page 16-5.

CHAPTER

17 GENERAL CIRCUIT INTERFACE (GCI)

CC

Note: *Only the Am186CC microcontroller supports GCI.*

17.1 OVERVIEW

The General Circuit Interface (GCI) is an interface specification developed jointly by Alcatel, Italtel, GPT and Siemens. This specification (sometimes called IOM-2) defines an industry-standard serial bus for interconnecting telecommunications integrated circuits. The standard covers linecard, NT1, and terminal architectures for Integrated Services Digital Network (ISDN) applications. The Am186CC microcontroller supports the terminal version of GCI, which serves four main functions:

- Connection of voice/data modules to an OSI Layer 1, GCI-SCIT (Special Circuit Interface T) device (transceiver)
- Programming and control of devices that do not have a microprocessor interface (e.g., a coder-decoder (codec) or a U-Interface transceiver)
- Interchip communications between devices on the bus (e.g., a codec to a speech encryption device)
- Connection of multiple data link controllers to the D channel, including access arbitration handled through the Terminal Interchip Communication (TIC) bus

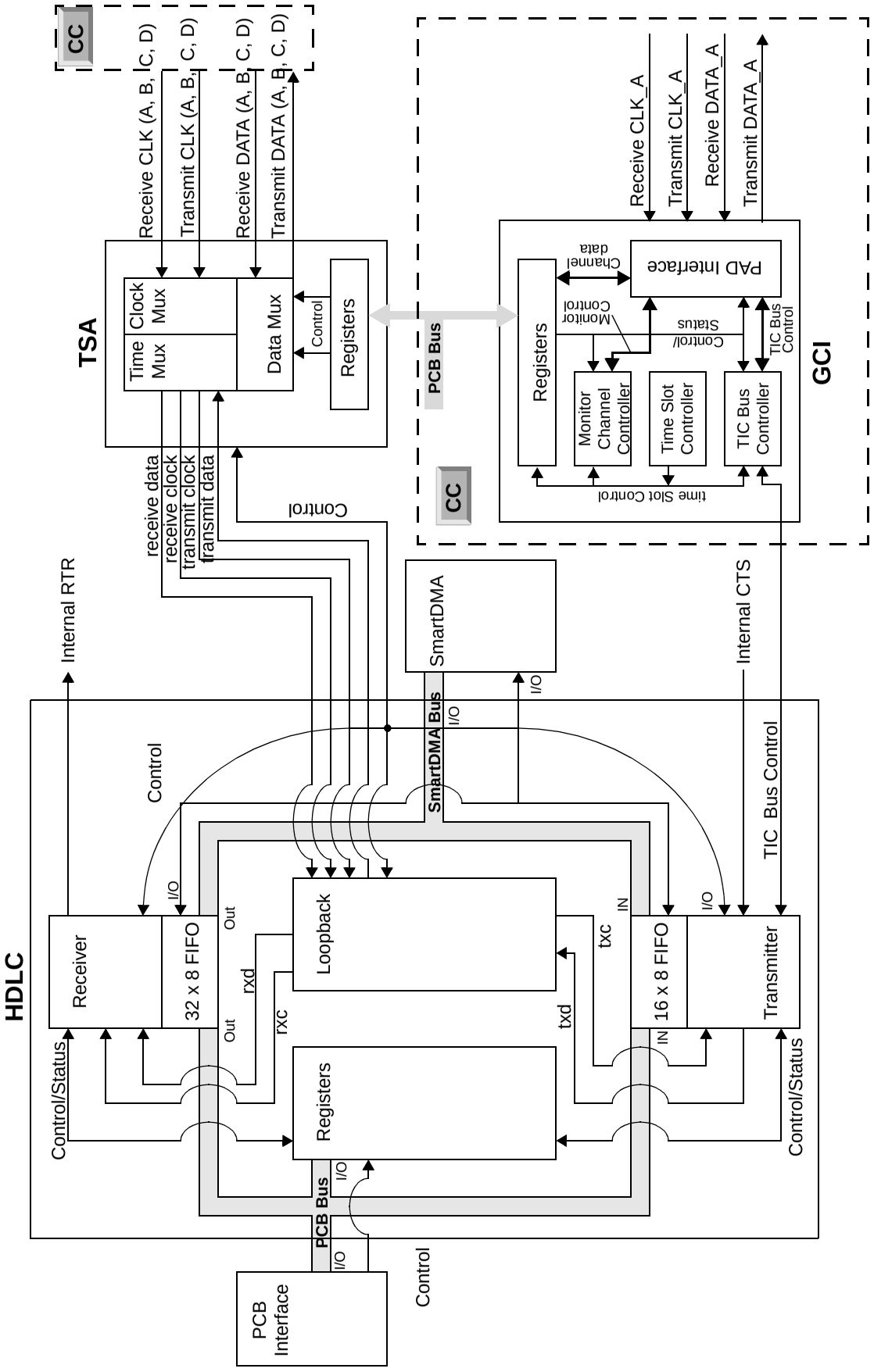
Depending on the application, each HDLC can communicate to the external world with or without a TSA. Each of the four HDLC channels can be programmed to select between raw DCE and dedicated PCM Highway external interfaces.

The Am186CC microcontroller's HDLC Channel A interfaces to the GCI controller block, and allows multiplexed PCM Highway and GCI interfaces to the other three HDLC channels. See Chapter 15, "High-Level Data Link Control (HDLC)" and Chapter 16, "HDLC External Serial Interface Configuration (TSAs)" for more information. Full documentation on GCI/IOM-2 is available in the AMD *IOM-2 Interface Reference Guide*, order #12576.

17.2 BLOCK DIAGRAM

Figure 17-1 shows the block diagram for a single HDLC channel, including connections with the TSA and GCI.

Figure 17-1 HDLC, TSA, and GCI Block Diagram (Same as Figure 15-2)



17.3 SYSTEM DESIGN

Table 17-1 lists the HDLC/TSA/GCI signals that are multiplexed with other Am186CC microcontroller functions. Pinstraps are sampled only at external reset and do not affect the pin's other functions, so they are not shown in this table. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin. Table 17-2 on page 17-3 shows an example application.

Figure 17-2 ISDN TA GCI-to-PCM Conversion System Application Example

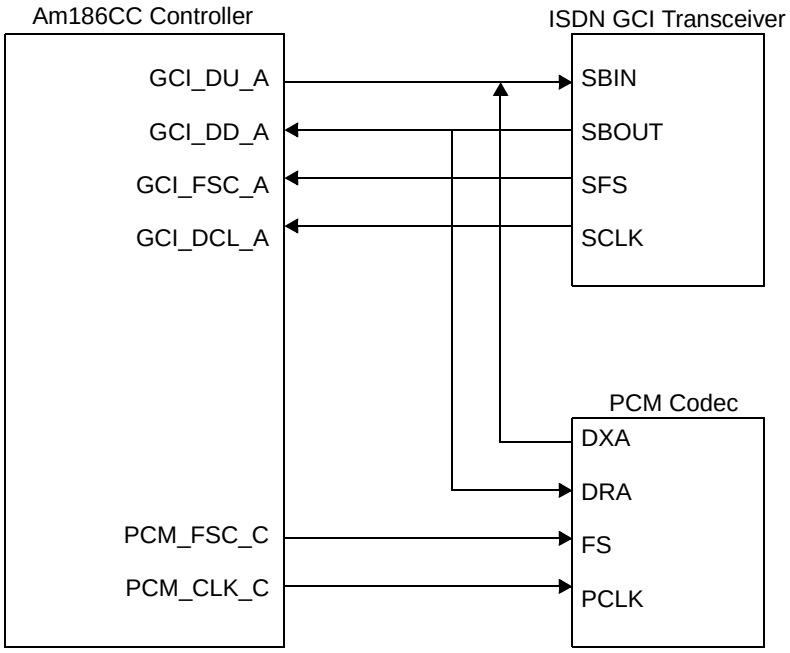


Table 17-1 HDLC/TSA/GCI Multiplexed Signals (Same as Table 15-1)

Multiplexed Signals						Function	Default Signal
Ch	External Interface				PIOs		
	DCE	PCM	GCI CC	UART			
A	DCE_RXD_A	PCM_RXD_A	GCI_DD_A	—	—	DCE and PCM data input/ GCI downstream pin	DCE_RXD_A
	DCE_TXD_A	PCM_TXD_A	GCI_DU_A	—	—	DCE and PCM data output/ GCI upstream pin	DCE_TXD_A
	DCE_RCLK_A	PCM_CLK_A	GCI_DCL_A	—	—	DCE receive clock/PCM receive and transmit clock/GCI receive and transmit clock	DCE_RCLK_A
	DCE_TCLK_A	PCM_FSC_A	GCI_FSC_A	—	—	DCE transmit clock/PCM frame sync clock/GCI frame sync clock	DCE_TCLK_A
	$\overline{\text{DCE\_CTS\_A}}$	$\overline{\text{PCM\_TSC\_A}}$	—	—	PIO17	DCE clear to send/PCM external buffer enable	PIO17
	$\overline{\text{DCE\_RTR\_A}}$	—	—	—	PIO18	DCE ready to receive	PIO18

Table 17-1 HDLC/TSA/GCI Multiplexed Signals (Same as Table 15-1) (Continued)

Multiplexed Signals						Function	Default Signal
Ch	External Interface				PIOs		
	DCE	PCM	GCI CC	UART			
B	DCE_RXD_B	PCM_RXD_B	—	—	PIO36	DCE and PCM data input pin	PIO36
	DCE_TXD_B	PCM_TXD_B	—	—	PIO37	DCE and PCM data output pin	PIO37
	DCE_RCLK_B	PCM_CLK_B	—	—	PIO40	DCE receive clock/PCM receive and transmit clock	PIO40
	DCE_TCLK_B	PCM_FSC_B	—	—	PIO41	DCE transmit clock/PCM frame sync clock	PIO41
	$\overline{\text{DCE\_CTS\_B}}$	$\overline{\text{PCM\_TSC\_B}}$	—	—	PIO38	DCE clear to send/PCM external buffer enable	PIO38
	$\overline{\text{DCE\_RTR\_B}}$	—	—	—	PIO39	DCE ready to receive	PIO39
C CC	DCE_RXD_C	PCM_RXD_C	—	—	PIO42	DCE and PCM data input pin	PIO42
	DCE_TXD_C	PCM_TXD_C	—	—	PIO43	DCE and PCM data output pin	PIO43
	DCE_RCLK_C	PCM_CLK_C	PCM_CLK_C	—	PIO22	DCE receive clock/PCM receive and transmit clock input/GCI-to-PCM conversion clock output	PIO22
	DCE_TCLK_C	PCM_FSC_C	PCM_FSC_C	—	PIO23	DCE transmit clock/PCM frame sync clock input/GCI-to-PCM conversion frame sync output	PIO23
	$\overline{\text{DCE\_CTS\_C}}$	$\overline{\text{PCM\_TSC\_C}}$	—	—	PIO44	DCE clear to send/PCM external buffer enable	PIO44
	$\overline{\text{DCE\_RTR\_C}}$	—	—	—	PIO45	DCE ready to receive	PIO45
D CC	DCE_RXD_D	PCM_RXD_D	—	RXD_U	PIO26	DCE and PCM data input/ UART data receive	PIO26
	DCE_TXD_D	PCM_TXD_D	—	TXD_U	PIO20	DCE and PCM data output/ UART data transmit	PIO20
	DCE_RCLK_D	PCM_CLK_D	—	RTR_U	PIO25	DCE receive clock/PCM receive and transmit clock input/UART ready-to-receive	PIO25
	DCE_TCLK_D	PCM_FSC_D	—	CTS_U	PIO24	DCE transmit clock/PCM frame sync clock input/UART clear-to-send	PIO24
	$\overline{\text{DCE\_CTS\_D}}$	$\overline{\text{PCM\_TSC\_D}}$	—	CTS_HU	PIO46	DCE clear to send/PCM external buffer enable/High-Speed UART clear-to-send	PIO46
	$\overline{\text{DCE\_RTR\_D}}$	—	—	RTR_HU	PIO47	DCE ready to receive/High-Speed UART ready-to-receive	PIO47

17.4 REGISTERS

The registers listed in Table 17-2 program the GCI. Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 17-2 GCI Register Summary

Offset	Register Mnemonic	Register Name	Description
2A0h	GPCON	GCI Peripheral Control	Configures GCI.
2A2h	GISTAT	GCI Interrupt Status	Contains status. All bits can generate an interrupt if not masked off in GIMSK.
2A4h	GIMSK	GCI Interrupt Mask	Mask register for GISTAT. When a mask bit is 0 (the reset value), the corresponding interrupt is masked off.
2A6h	GTIC	GCI TIC Bus Address	Enables TIC bus operation.
2A8h	GICTD	GCI Intercommunication Transmit Data	Contains user-defined transmission data for GCI IC Channel 1 or 2.
2AAh	GICRD	GCI Intercommunication Receive Data	Contains received data for either GCI IC Channel 1 or 2.
2ACh	GICRDP	GCI Intercommunication Receive Data Peek	Copy of GICRD register that does not change status when read.
2AEh	GCITD0	GCI Command/Indicate Transmit Data 0	Contains user-defined transmission data for GCI C/I0 channel.
2B0h	GCIRD0	GCI Command/Indicate Receive Data 0	Contains received data for GCI C/I0 channel.
2B2h	GCIRD0P	GCI Command/Indicate Receive Data 0 Peek	Copy of GCIRD0 register that does not change status when read.
2B4h	GCITD1	GCI Command/Indicate Transmit Data 1	Contains user-defined transmission data for GCI C/I1 channel.
2B6h	GCIRD1	GCI Command/Indicate Receive Data 1	Contains received data for GCI C/I1 channel.
2B8h	GCIRD1P	GCI Command/Indicate Receive Data 1 Peek	Copy of GCIRD1 register that does not change status when read.
2BAh	GMTD	GCI Monitor Transmit Data	Contains user-defined transmission data for GCI Mon0 or Mon1 channels.
2BCh	GMRD	GCI Monitor Receive Data	Contains received data for GCI Mon0 or Mon1 channels.
2BEh	GMRDP	GCI Monitor Receive Data Peek	Copy of GMRD register that does not change status when read.

17.5 OPERATION

17.5.1 Usage

Note: *Before using GCI, ensure multiplexed pins are configured to reflect the use of GCI and not other functionality (see Table 17-1 on page 17-3).*

1. To enable the GCI interface, software must set the MODE field of the TSACON register to 10b. This is necessary whether or not TSA Channel A is being used.
2. If transmitting using the GCI, see “Transmitting Data” on page 17-6; if receiving, see “Receiving Data” on page 17-7.

17.5.1.1 Transmitting Data

1. Configure the HDLC channels and time slot assigners to transmit the data. For details, see Chapter 15, “High-Level Data Link Control (HDLC),” and Chapter 16, “HDLC External Serial Interface Configuration (TSAs).”
2. Configure and activate the GCI channels:
   a. If using the TIC bus access procedure, set the TICEN and TICAD bits in the GTIC register. For information about the TIC bus access procedure, see “TIC Bus Support” on page 17-16.
   b. If transmitting Monitor channel data, set the applicable configuration options in the GPCON register (the MCHEN, MCHSEL, and BRDIS bits).
   c. If transmitting IC channel data, set the applicable configuration options in the GPCON register (the ICSEL and BRDIS bits).
   d. For CI/1 channel data, set the applicable configuration option in the GPCON register (the BRDIS bit).
   e. If the bus is in a deactivated state, activate the bus by setting the GCIACT bit in the GPCON register. For details, see “GCI Bus Deactivation/Activation” on page 17-9.
3. Set the interrupts to be taken with the GIMSK register. Bits in this register enable interrupts based on interrupts set in the GISTAT register. Corresponding bits must be set in both registers for the interrupt to be taken. If an interrupt is disabled in GIMSK, the status can still be read. Note that for each GCITD0 register write, if using the TIC bus access procedure, the BAR bit must be taken into consideration.
4. Wait for the DCLST bit in the GISTAT register to be set, indicating the data clock has been started by the master clock device.
5. If the bus was in a deactivated state, turn off the GCI activation request by clearing the GCIACT bit in the GPCON register.
6. For monitor channel transmission, each Transmit Buffer Available Interrupt should write new data into the transmit register until all data has been transmitted.
7. For monitor channel transmission, set the MEOMRQ bit in the GPCON register on the last byte transmitted. This bit forces the monitor channel transmitter to send an EOM when all data is written.

The outgoing MX bits and incoming MR bits held inactive for two or more frames indicates that the Monitor channel is idle in the outgoing direction.

At the start of transmission, program the GPCON register to select one of the two monitor channels. Then load data into the GCI Monitor Transmit Data (GMTD) register, which causes the GCI controller to present the first data byte to the bus and to perform an inactive-to-active transition of the outgoing MX bits. Placing data on the bus also generates the Monitor channel transmit buffer available interrupt, indicating that the next data byte may be written to the buffer. Outgoing MX bits remain active, and the data is repeated until an inactive-to-active transition of the incoming MR bit is received.

In subsequent transmissions, all the following bytes to be transmitted are presented to the bus coincident with an active-to-inactive transition of outgoing MX bits. The GCI specification defines a general case in which the transmitter waits for an inactive-to-active transition of incoming MR bits, and a maximum speed case in which the transmitter achieves a higher transmission rate by anticipating the falling edge of incoming MR bits. After transmitting the last byte of data, indicated by the GMTD register being empty and the

MEOMRQ bit being set, the GCI controller deactivates the outgoing MX bits in response to incoming MR bits going inactive, and leaves them inactive.

17.5.1.2 Receiving Data

1. Configure the HDLC channels and time slot assigners to receive the data. For details, see Chapter 15, “High-Level Data Link Control (HDLC),” and Chapter 16, “HDLC External Serial Interface Configuration (TSAs).”
2. Configure the GCI channels:
   a. If using the TIC bus access procedure, set the TICEN bits in the GTIC register. For information about the TIC bus access procedure, see “TIC Bus Support” on page 17-16.
   b. If receiving Monitor channel data, set the applicable configuration options in the GPCON register (the MCHEN, MCHSEL, and BRDIS bits).
   c. If receiving IC channel data, set the applicable configuration options in the GPCON register (the ICSEL and BRDIS bits).
   d. For CI/1 channel data, set the applicable configuration option in the GPCON register (the BRDIS bit).
   e. If the bus is in a deactivated state, activate the bus by setting the GCIACT bit in the GPCON register. For details, see “GCI Bus Deactivation/Activation” on page 17-9.
3. Set the interrupts to be taken with the GIMSK register. Bits in this register enable interrupts based on interrupts set in the GISTAT register. If software disables an interrupt in GIMSK, it can still read the interrupt status in the GISTAT register.
4. Wait for the DCLST bit in the GISTAT register to be set, indicating the data clock has been started by the master clock device.
5. If the bus was in a deactivated state, turn off the GCI activation request by clearing the GCIACT bit in the GPCON register.
6. For monitor channel transmission, on the first data available interrupt, software must set the MCARV configuration bit to continue transmission. This bit holds off the remote transmitter until software has determined the first byte is valid (the first byte is usually a known address byte). If software fails to determine that the first byte is valid, then software should abort reception (i.e., this message is for some other downstream device).
7. For monitor channel transmission, the MEOMRD interrupt is set to indicate that an end-of-message (EOM) has been received by the monitor channel.

At the time the receiver sees the first byte, indicated by the inactive-to-active transition of incoming MX bits, outgoing MR bits are by definition inactive. The GCI controller activates outgoing MR bits in response to the activation of incoming MX bits, loads the data byte on the bus into the Monitor Receive Data register, and generates a Monitor channel receive data available interrupt. Outgoing MR bits remain active until the next byte is received or an EOM is detected (incoming MX bits held inactive for two or more frames).

In subsequent receives, the GCI controller receives data into the buffer on each falling edge of incoming MX bits, and generates a Monitor channel receive data available interrupt. Note that the data was actually valid at the time the incoming MX bits became inactive, one frame before becoming active (the Am186CC microcontroller performs a data integrity check to confirm stable data for two frames). Outgoing MR bits are deactivated at the time data is read and reactivated one frame later. The receipt of an EOM, which is incoming MX bits remaining inactive for two or more frames, terminates the reception of data.

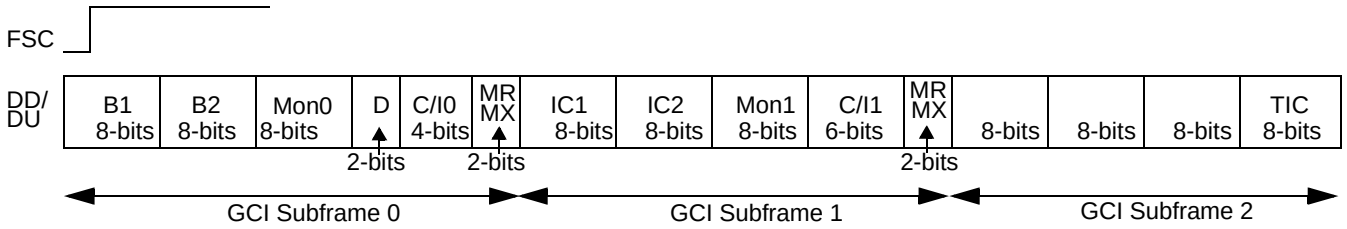
An abort is a signal from the receiver to the remote transmitter indicating that data has been missed. The receiver sends an abort (indicated with the MTARD bit in the GISTAT register) by holding MR bits inactive for two or more frames in response to MX bits going active. Receiving an abort, indicated with the MRAD bit of the GISTAT register, generates a transmitter interrupt.

The remote transmitter is held off until the Monitor Receive Data register is read, because MR bits are held active until the receive byte is read. The transmitter does not start the next transmission cycle until MR bits go inactive.

17.5.2 GCI Structure: Channels and Frames

Figure 17-3 illustrates the GCI terminal mode frame structure. The Am186CC microcontroller also provides a second interface used with the GCI interface (discussed in "GCI-to-PCM Converted Pin Interface" on page 17-14). This second interface allows an external PCM codec to multiplex directly onto a GCI terminal frame B channel. For more information, see the AMD *IOM-2 Interface Reference Guide*, order #12576.

Figure 17-3 GCI Terminal Mode Frame Structure



17.5.3 GCI Applications

The Am186CC microcontroller GCI implementation:

- Targets the following external ISDN transceivers in GCI mode:
  - AMD Am79C30/Am79C32 S
  - Siemens PEB2091 U
  - Siemens PEB2081 S/T
  - Siemens PEB2086 S
  - Motorola MC145574 S/T
  - Motorola MC145572 U
  - National TP3420 S/T
- Targets the following external codec:
  - AMD Am79C04
- With GCI-to-PCM conversion, targets the following PCM external codecs (see Table 16-3 on page 16-14):
  - AMD Am79C02/03
  - AMD Am79C031
  - Motorola MC14555

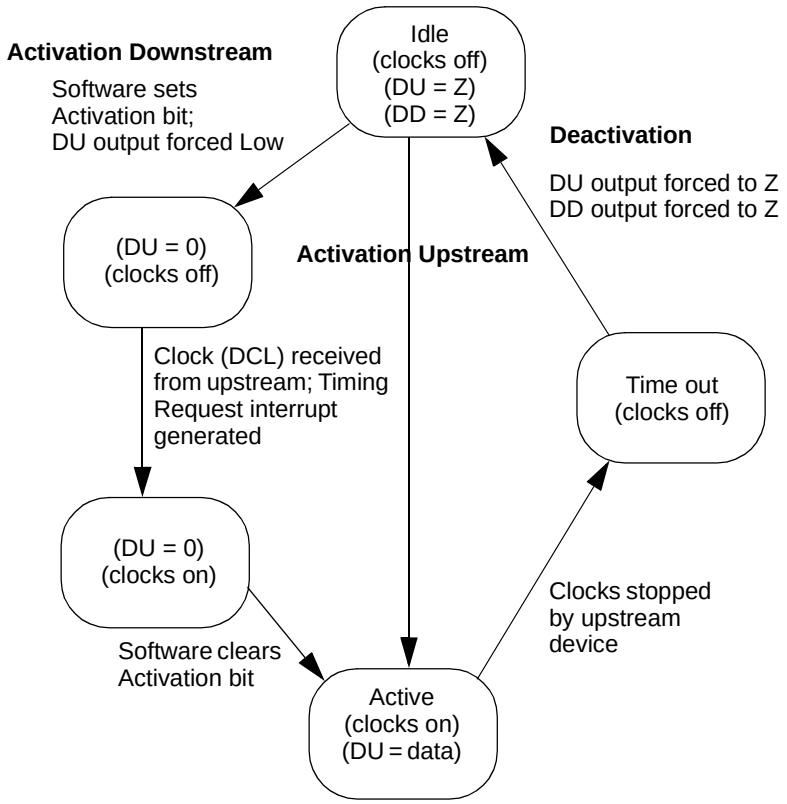
  - National TP305x family
  - National TP307x family
  - AT&T T75xx family
  - TI/Intel 291x family
- Supports GCI Terminal Mode
- Supports GCI Slave Mode (i.e., a timing slave where the GCI_FSC_A and GCI_DCL_A signals are inputs)
- Supports interdevice communication via Monitor and Command/Indicate channels
- Supports the TIC bus, providing the capability of connecting more than one device to the D and C/I channels in the first subframe (via C/I0 and D-channel arbitration)
- Supports D-channel Collision Detection via the Echo bits from the S-interface
- Is multiplexed with *one* fixed Am186CC external raw DCE interface
- Does not support:
  - GCI Linecard Mode
  - GCI Master Mode
  - TIC bus A/B bit: an optional supplementary bit used for D-channel control: 1 indicates the D channel is available, 0 indicates the D channel is blocked
  - The following terminal mode signals (used for connecting non-GCI components):
    - BCL: 1X-bit rate clock
    - SDS1 and SDS2: Data strobes which identify the location of the B channels

17.5.4 GCI Bus

17.5.4.1 GCI Bus Deactivation/Activation

The GCI bus includes an activation/deactivation capability. Either upstream components or downstream components on the bus can initiate activation and deactivation. Figure 17-4 illustrates the activation/deactivation process. When deactivated, the upstream device holds all clock outputs Low. The downstream device holds all the clock outputs Low, and forces open drain data outputs to a high-impedance state (seen as a High on the system bus due to the external pullup resistor). The activation/deactivation procedure is a combination of software handshakes through the C/I channel, and hardware indications through the clock and data lines. The AMD *IOM-2 Interface Reference Guide*, order #12576, describes both the hardware and the software protocols in detail.

Figure 17-4 Bus Activation/Deactivation



17.5.4.1.1 Deactivation

The upstream device typically initiates deactivation. When the Am186CC microcontroller receives the deactivation request over the C/I channel, it must respond by sending the deactivation indication over the C/I channel. The upstream device then sends the deactivation confirmation command over the C/I channel. The Am186CC microcontroller detects that the clock has stopped (defined as no clock pulse received for 650 ns) and forces itself to the deactivated state.

In the deactivated state, the microcontroller forces both the DU and DD signals to a high-impedance state, and monitors the DCL input (by use of the DCLST bit in the GISTAT register) for any rising edge that would indicate an activation request from the upstream device.

17.5.4.1.2 Activation

Either the upstream or the downstream device can initiate activation. For the Am186CC microcontroller to activate the interface, software must set the activation bit (GCIACT) of the GPCON register. This forces the microcontroller to pull its data output pin (DU) Low, causing the upstream device to start the GCI clocks. When the clocks are running, as indicated by the DCLST status bit being set, the microcontroller must respond to the interrupt by loading the proper C/I command response into the C/I0 transmit register, then clearing the GCIACT bit. This releases the data output pin (DU) from being held Low and allows the microcontroller to complete the activation procedure by sending the proper commands over the C/I channel. The DCL clock remains active until the upstream device stops the clock.

When activation originates from the upstream device, the DCLST bit is set when the clocks become active (DCL going High). The microcontroller begins normal GCI transmission/

reception as soon as DCL appears; no intervention from the controller is required. However, the microcontroller must respond to the interrupt and perform the normal C/I channel software handshakes before activation completes.

17.5.4.2 GCI Bus Reversal

In Terminal mode, a device may be required to transmit both upstream and downstream, based on which GCI channel is being transmitted at any one time. As a result, the actual data pins of the GCI interface need to be both inputs and outputs, changing direction based on which channel is being transmitted at the time.

17.5.4.2.1 Downstream Versus Upstream

The following terms are used in GCI:

Downstream Direction: Data is output on GCI_DD_A by an upstream device and this data is a GCI_DU_A input to the downstream device.

Upstream Direction: Data is output on GCI_DU_A by a downstream device and this data is a GCI_DD_A input to the upstream device.

Downstream Device: Generates GCI_DU_A and terminates GCI_DD_A.

Upstream Device: Generates GCI_DD_A and terminates GCI_DU_A.

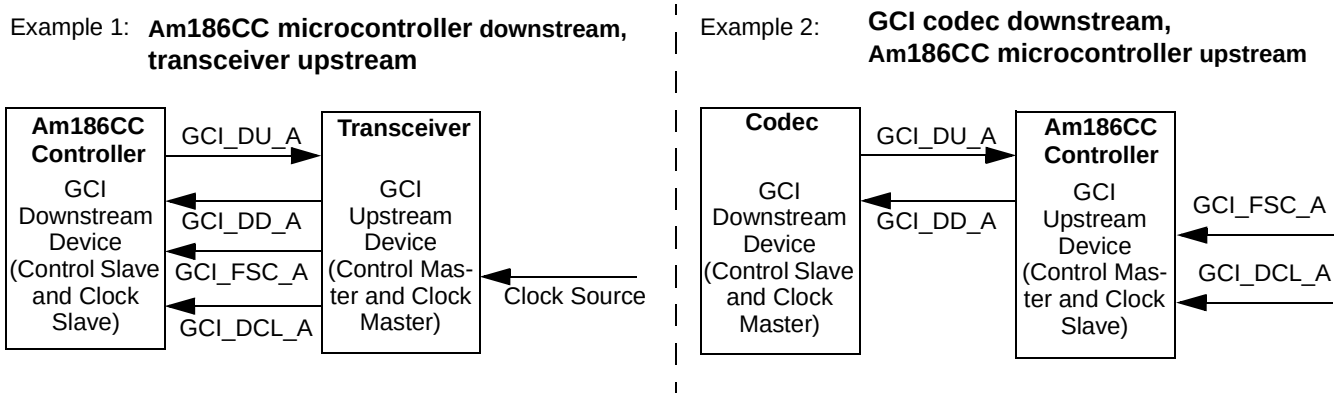
Because pin reversal is supported, a device on the GCI bus can be considered a downstream device, an upstream device, or both. Figure 17-5 demonstrates the Am186CC microcontroller as an GCI Subframe 0 downstream device (the transceiver, an upstream device, outputs data on GCI_DD_A and the Am186CC microcontroller, the downstream device, inputs data from GCI_DD_A).

Figure 17-5 also demonstrates the Am186CC microcontroller as a GCI Subframe 1 upstream device (the Am186CC microcontroller, an upstream device, outputs data on GCI_DD_A and a downstream device, such as an GCI codec, inputs data from GCI_DD_A). Devices which do not support pin reversal are fixed to transmit and receive in one direction only.

For example, a line transceiver is always an upstream device communicating solely with downstream devices (it transmits information on GCI_DD_A to downstream devices, and receives information on GCI_DU_A from devices sending information upstream to this upstream transceiver). Therefore, in this case, anything on the GCI bus is always considered downstream from the upstream transceiver.

Note: *In most documentation, where a reference point is not given, but upstream or downstream are mentioned, the default reference point is almost always the transceiver: that is, downstream (from the transceiver), upstream (to the transceiver), the upstream (transceiver) device, and so on.*

Figure 17-5 Downstream Versus Upstream

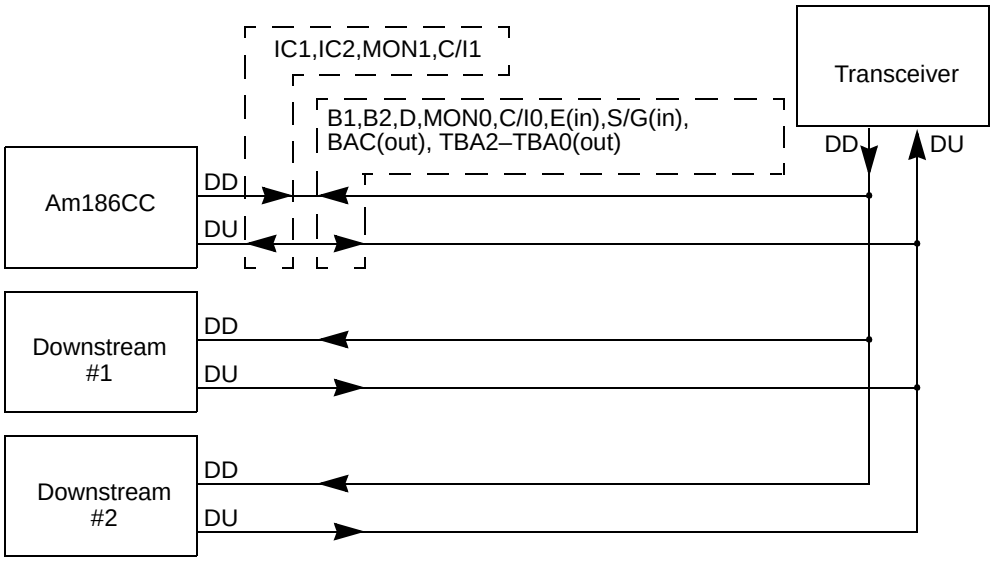


17.5.4.2.2 Bus Reversal Enabled Versus Disabled

When Bus Reversal is *enabled* (see Figure 17-6), the Am186CC microcontroller is the clock slave (GCI_FSC_A and GCI_DCL_A are inputs) and control *master* (can communicate with other downstream devices through the MON1 and C/I1 channels). When Bus Reversal is *disabled* (see Figure 17-7), the Am186CC microcontroller is the clock slave (GCI_FSC_A and GCI_DCL_A are inputs) and control *slave* (*cannot* communicate with other downstream devices).

D and C/I0 channel arbitration are provided by the TIC bus. (The TIC bus has been split up into its individual bits for illustration.)

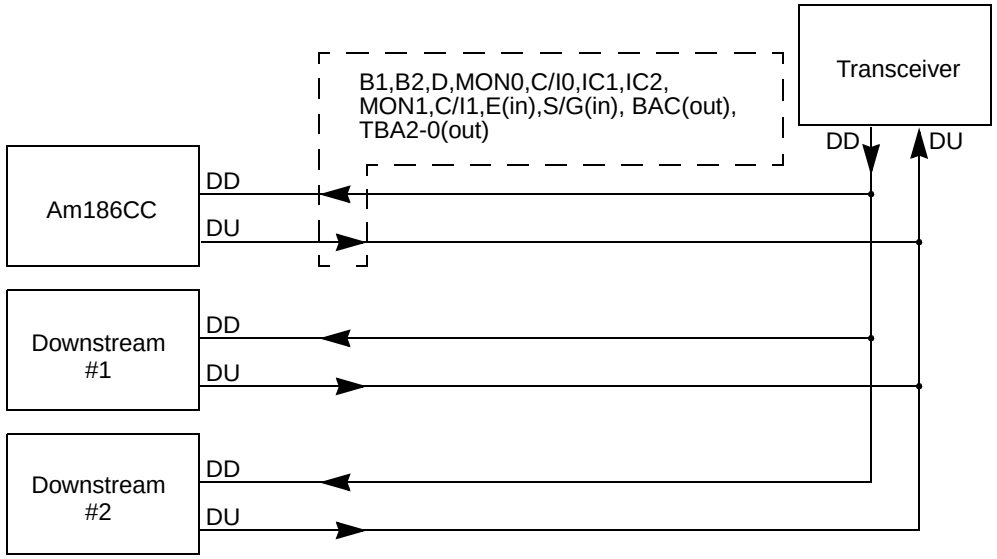
Figure 17-6 GCI With Bus Reversal Enabled



Notes:

E, S/G, BAC, and TBA2–TBA0 are bits on the TIC bus.

Figure 17-7 GCI With Bus Reversal Disabled



Notes:

E, S/G, BAC, and TBA2–TBA0 are bits on the TIC bus.

17.5.5 GCI Interface Signals

17.5.5.1 Four-Pin Interface

The GCI terminal mode interface consists of a four-pin subset of the seven-pin GCI industry standard serial bus. The GCI interface for the Am186CC microcontroller uses the frame synchronization clock (GCI_FSC_A), data clock (GCI_DCL_A), data downstream (GCI_DD_A), and data upstream (GCI_DU_A) signals. The definition of the GCI external signals is dependent on the current bus state (activated/deactivated) and the mode of operation (bus reversal enabled/disabled), as described in Table 17-3.

Table 17-3 GCI Signals

Signal	Signal Function	Mode: Reversal State: Activated	Mode: Reversal State: Deactivated	Mode: No Reversal State: Activated	Mode: No Reversal State: Deactivated
GCI_FSC_A	Frame sync clock	Input	Input	Input	Input
GCI_DCL_A	Data clock	Input	Input	Input	Input
GCI_DD_A	Data down-stream	Input/ Open Drain Output	High Impedance	Input	High Impedance
GCI_DU_A	Data up-stream	Open Drain Output/Input	Open Drain Output	Open Drain Output	Open Drain Output

17.5.5.2 GCI-to-PCM Converted Pin Interface

The converted GCI to PCM interface consists of two pins. This interface (external interface C on the Am186CC microcontroller) uses the signals listed in Table 17-4. For more information about frame sync and clock conversion, see Chapter 16, “HDLC External Serial Interface Configuration (TSAs).”

Table 17-4 Converted GCI Signals

External Signal	Function
PCM_CLK_A	Converted GCI to PCM data clock
PCM_FSC_A	Converted GCI to PCM frame synchronization clock

17.5.6 Operating Frequencies

GCI_DCL_A is used to clock data on and off of the bus, and it operates at twice the data rate.

The clock rate is 1.536 MHz (the data rate is 768 KHz). GCI_DCL_A is always generated by the upstream component. When the bus is deactivated, GCI_DCL_A is held in a Low state by the upstream device.

GCI_FSC_A is an 8-KHz clock that indicates the start of a frame. GCI_FSC_A is always generated by the upstream device, which is the Layer 1 device in terminal applications.

17.5.7 GCI Channels

The GCI channels consist of three voice/data channels (B1, B2, and D), two monitor channels (Mon0 and Mon1), two command/indication channels (C/I0 and C/I1), two Interchip Communication channels (IC1 and IC2), and the TIC bus.

17.5.7.1 GCI HDLC Channel Steering

All HDLC channel data steering is provided through proper TSA (Time Slot Assigner) and MUX initialization. While there is nothing to prevent an HDLC channel from accessing any set of contiguous bits within the GCI frame, only accesses to the B, D, and IC channels are guaranteed.

For more information about TSA configuration, see Chapter 16, “HDLC External Serial Interface Configuration (TSAs).”

17.5.7.2 Monitor Channel Operation

The monitor channel is full duplex and operates on a pseudo-asynchronous basis, that is, while data transfers on the bus take place synchronized to frame sync, the flow of data is controlled by a handshake procedure using the MX (monitor transmit) and MR (monitor receive) bits. For example, data is placed onto the monitor channel and the MX bit is activated. This data is transmitted repeatedly (once per 8-KHz frame) until the transfer is acknowledged through the MR bit. Thus, the data rate is not 8 Kbyte per second. Am186CC microcontroller monitor channel support is provided on a one-at-a-time basis (the MCHSEL bit in the GPCON register designates which monitor channel is selected). For a detailed description of the monitor channel handshake procedure, see the AMD *IOM-2 Interface Reference Guide*, order #12576.

17.5.7.3 Monitor Channel Collision Detection

For multidrop configurations, a collision resolution mechanism is implemented in the Monitor Channel transmitter that looks for the idle phase of the transmitted MX/MR bits and makes a per bit check on the *transmitted* monitor data.

To access the monitor channel through Upstream Monitor Channel Collision Detection on the first byte and Downstream Device Recognition on the first byte (these procedures are used in Monitor Channel multidrop configurations), use the following procedures.

17.5.7.3.1 Upstream Monitor Channel Data Transmission

The address of the monitor message contained in the first monitor byte transmitted determines the monitor channel access priority. The following hardware/software procedure is followed:

1. Software configures the monitor channel for data transmission.
2. Hardware waits for the idle phase before transmitting the first byte of monitor data.
3. During the first byte transmitted, a per bit check occurs on each transmitted monitor bit. If any bit mismatches, the transmitter immediately withdraws from the monitor channel by setting all remaining monitor bits to 1 (thus allowing another device with higher priority to gain control of the monitor channel), sets the monitor channel collision detection interrupt, and reverts back to waiting for the idle condition.

Note: *The collision detection interrupt is set on any monitor data transmit bit mismatch (i.e., from the first byte transmitted to the last byte transmitted). Therefore, if software wishes to differentiate how it services other byte collisions from first-byte collisions, it must maintain this knowledge itself.*

17.5.7.3.2 Downstream Monitor Channel Data Reception

Device recognition allows a downstream device to determine whether or not it is the intended target for an initiated Monitor Channel message sent by an upstream device (the address to be recognized is contained in the first byte of the monitor message). The following hardware/software procedure is followed:

1. Hardware waits for the idle phase.
2. After detecting the idle phase, hardware waits for a valid first byte to be sent by an upstream device.
3. After receiving the first byte, hardware indicates to software, through a data-available interrupt, that the first byte has arrived.
4. Software determines whether or not the microcontroller was the intended target.
5. If a valid address is recognized (from the first byte), software indicates to the receiver to continue with data reception by setting a valid address-compare bit. Otherwise, software indicates to the receiver that it should not continue receiving data (through a software-abort bit).

17.5.7.4 C/I Channel Operation

The C/I channel communicates real-time status information and maintenance commands. Unlike the monitor channel, the Am186CC microcontroller supports both C/I channels contained in GCI Subframe 0 and GCI Subframe 1 concurrently. Software reads the received data from one of the C/I Receive Data (GCIRD0 or GCIRD1) registers. Software writes C/I transmit data to one of the C/I Transmit Data (GCITD0 or GCITD1) registers. The GCI controller monitors these two channels, and generates an interrupt any time the receive data changes and is stable for two frames (GCI's standard data integrity check). Data on the C/I channel is continuously transmitted in each frame until new data is to be sent. In this way, the C/I channel can be thought of as a set of static status lines that only change when the status changes. For a list of C/I codes (for GCI Subframe 0 only), and further C/I channel operation, refer to the AMD *IOM-2 Interface Reference Guide*, order #12576.

17.5.7.5 TIC Bus Support

The meaning of each bit within the TIC bus is dependent on whether the Am186CC microcontroller is transmitting or receiving on the TIC bus. Table 17-5 lists and describes the TIC bus bits.

Table 17-5 TIC Bus Bits

Bit Name	Bit Function
BAC (Bus Accessed)	Indicates to the other devices that the TIC bus is being accessed. When 0, the bus is being accessed; when 1, it is free. This bit is driven to zero by the device that gets an address match on TBA2–TBA0.
TBA2-0 (TIC Bus Address)	Address bit used for arbitration of TIC bus control. Assumes open-drain bus such that the device with the lowest address has the highest priority. The lowest priority address, which is also the default, is 111.
E-bits (Echo Bits)	D-channel Echo bits from the S-interface.
S/G bit (Stop/Go)	Indicates availability of the S-interface D-channel. When 0, the D-channel is clear for transmission. When 1, D-channel transmission should be halted.

In the downstream direction (from the transceiver), the TIC bus on GCI Subframe 2 is used for D and C/I0 channel access control in S/T interface terminals.

The TIC bus downstream has the format shown in Figure 17-8.

Figure 17-8 TIC Bus Downstream Format

Bit Number	7	6	5	4	3	2	1	0
Bit Name	E	E	S/G	A/B	1	1	1	1

The availability of the S/T interface D-channel is indicated in bit 5 (Stop/Go bit) of the downstream TIC bus. The Am186CC microcontroller GCI TIC bus controller checks the Stop/Go bit to determine if it has access to the D-channel. If it does, it can start transmission of an HDLC frame. If the TIC bus controller does not have access, it must halt the transmission. Bits 7 and 6 are the D-channel Echo bits from the S-interface (reflecting back the two D-channel bits of the current frame). The Am186CC microcontroller GCI TIC bus controller compares the Echo bits with the sent D-channel bits to determine if a collision has occurred. A D-channel collision is reported to an HDLC through an internal signal, originating from the GCI TIC bus controller, whose function is similar to an external $\overline{\text{CTS}}$ deassertion (a mechanism that stops HDLC transmission). The Am186CC microcontroller does not use the A/B bit.

In the upstream direction (to the transceiver), the TIC bus on GCI Channel 2 is used for the TIC bus access procedure, enabling the connection of several Layer 2 D-channel protocol controllers to the GCI interface.

The TIC bus upstream has the format shown in Figure 17-9.

Figure 17-9 TIC Bus Upstream Format

Bit Number	7	6	5	4	3	2	1	0
Bit Name	1	1	BAC	TBA2	TBA1	TBA0	1	1

An Am186CC microcontroller access request can either be generated by software (microprocessor access to C/I Channel 0) or by the HDLC controller itself (transmission of an HDLC frame—signified internally by a signal, originating from the GCI TIC bus controller, whose function is similar to an external $\overline{\text{RTS}}$ assertion). In the case of an access request, the GCI TIC bus controller checks the BAC bit for the status “bus free” (BAC = 1). If the bus is free, the GCI TIC bus controller starts transmitting its individual TIC bus address (the source address indicated in Figure 17-9 by the TBA2–TBA0 bits). If an erroneous address is detected, the procedure is terminated immediately. If the complete TIC bus address can be transmitted without error, the D-channel and C/I Channel 0 are immediately occupied; during the subsequent frames the bus is identified as occupied (BAC = 0) until the access request is withdrawn. After a successful bus access, the HDLC controller is set into a lower priority class, that is, a new bus access cannot be performed until the status “bus free” (BAC = 1) is indicated in two successive frames.

If none of the D-channel protocol controllers connected to the GCI interface request access to the D and C/I channels, the TIC Bus Address 7 is present. The device with this address therefore has access, by default, to the D and C/I channels.

The following procedures gain access to the D-channel and C/I0 channel when TIC bus support is enabled.

17.5.7.5.1 D-Channel Arbitration and Collision Detection (Hardware Control)

Hardware flow control for the GCI Bus Accessed (BAC) bit is added through $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ handshaking, and follows a procedure very similar to the C/I0 arbitration scheme discussed in “C/I0 Arbitration (Software Control)” on page 17-18.

1. The HDLC controller makes a D-channel send request to the GCI TIC bus controller by asserting an internal $\overline{\text{RTS}}$ signal (this signal remains asserted until the entire HDLC frame has been transmitted).
2. The GCI TIC bus controller checks if the BAC bit is set to 1. If not, access is not currently allowed—transmission is postponed. Only when BAC = 1 does the GCI TIC bus controller continue with this access procedure. Otherwise, it remains in this state.
3. When BAC = 1, the GCI TIC bus controller, in the same frame, transmits the TIC bus address (TBA2–TBA0) on the open drain output. On the TIC bus, binary 0s overwrite binary 1s. Therefore, low TIC bus addresses have higher priority. During TIC bus reception, the S/G bit is monitored.

Note: *S/G bit generation in the GCI TIC bus is sent downstream from an upstream transceiver.*

4. After transmitting a TIC bus address bit, the GCI TIC bus controller reads back the value to check whether its own address bit has been overwritten by a controller with higher priority. This procedure continues until all three address bits are sent and confirmed—thus granting access to the GCI TIC bus. In the event a bit is overwritten by an external controller with higher priority, the GCI TIC bus controller withdraws immediately from the bus by setting all remaining TIC bus address bits to 1. (This assures that the lowest address has priority. If the remaining bits are not immediately set to 1, addresses such as 101 and 011 would have equal priority.) If a bit is overwritten and an address mismatch occurs, the TIC bus controller returns to step 2.
5. If access is granted (i.e., no address mismatch occurred) and the S/G bit is 0 (i.e., the S-interface is free for transmission), the GCI TIC bus controller asserts an internal $\overline{\text{CTS}}$ signaling to the HDLC controller that it is now allowed to clock out data on its programmed time slot starting in the following GCI frame. The BAC bit, during this HDLC transmission, is set to 0 by the GCI TIC bus controller to block all remaining controllers.

In the case where the S/G bit is 1, only the D-channel data is prevented from being switched through the GCI bus (i.e., the C/I0 channel could request access to this already established TIC bus and transmit its information). The TIC bus request remains unaffected (for example, if the microcontroller has earned the right to the GCI TIC bus it does not give up this bus and keeps BAC and the TIC address active while waiting for GO). As soon as the S-interface D-channel is clear, signified by the S/G bit cleared (GO), the controller commences with D-channel data transmission.

Note: *When GCI TIC access is granted, BAC = 0, regardless of S/G. At this point, both C/I0 and the HDLC controller have access to the GCI TIC bus (i.e., if C/I0 data needs to be transmitted it does not have to arbitrate for the GCI TIC bus—TIC bus access has already been established). To relinquish the GCI TIC bus after a C/I0 or D-channel transmission, both the C/I0 request (a software request) and the HDLC controller request (a hardware request) must be deasserted. The HDLC controller cannot transmit back-to-back frames. Therefore, if C/I0 keeps the TIC bus open (the TIC bus established by the HDLC controller), another HDLC transmission does not occur until after the C/I0 gives up the TIC bus and BAC = 1 in two successive frames (i.e., the TIC bus cannot be accessed again for at least one GCI frame—regardless of whether the HDLC controller request or the C/I0 request established the TIC bus).*

6. After the completed transmission of an HDLC frame, signified by the HDLC controller deasserting the TIC bus controller's $\overline{\text{RTS}}$, the HDLC controller is withdrawn from the TIC bus (BAC is set back to 1 in the following frame if a software TIC bus request has not been made for C/I0 communication), and the HDLC controller is prevented from accessing the TIC bus again for one GCI frame (i.e., the controller was moved into a lower priority as mentioned earlier). This also applies even if a new HDLC frame is to be transmitted in immediate succession. This gives all connected devices an equal chance to access the TIC bus.
7. If a collision occurs at any time during the transmission of a D-channel HDLC frame, the Am186CC microcontroller immediately ceases transmission (collision is signified to the HDLC controller by deasserting $\overline{\text{CTS}}$ while in frame), returns to the D-channel monitoring state (i.e., waits for another request to send and start over), and sends 1s over the D-channel.

17.5.7.5.2 C/I0 Arbitration (Software Control)

Software controls the GCI Bus Accessed (BAC) bit through the Bus Access Request (BAR) bit of the GCITDx register following a procedure very similar to the D-channel arbitration scheme described above. This bit provides access to the C/I0 channel when TIC bus support is enabled. Software should set the BAR bit whenever the microcontroller has C/I0 data available to transmit.

1. When BAR = 1, the TIC bus controller arbitrates access to the C/I0 channel.
2. The GCI TIC bus controller checks if the BAC bit is set to 1. If not, access is not currently allowed—transmission is postponed. Only when BAC = 1 does the GCI TIC bus controller continue with this access procedure. Otherwise, it remains in this state.
3. When BAC = 1, the GCI TIC bus controller, in the same frame, transmits the TIC bus address (TBA2–TBA0) on the open drain output. On the TIC bus, binary 0s overwrite binary 1s. Thus, low TIC bus addresses have higher priority.
4. After transmitting a TIC bus address bit, the GCI TIC bus controller reads back the value to check whether its own address bit has been overwritten by a controller with higher priority. This procedure continues until all three address bits are sent and confirmed—thus granting access to the GCI TIC bus. In the event a bit is overwritten by an external controller with higher priority, the GCI TIC bus controller withdraws immediately from the

bus by setting all remaining TIC bus address bits to 1. (This assures that the lowest address has priority. If the remaining bits are not immediately set to 1, addresses such as 101 and 011 would have equal priority.) If a bit is overwritten and an address mismatch occurs, the TIC bus controller returns to step 2.

5. If access was granted, the C/I0 channel is in possession of the GCI TIC bus, and C/I0 communication can begin in the following GCI frame.

Note: *When GCI TIC access is granted, BAC = 0—regardless of S/G. At this point, both C/I0 and the HDLC controller have access to the GCI TIC bus (i.e., if the HDLC controller needs to transmit D-channel data, it does not have to arbitrate for the GCI TIC bus—TIC bus access has already been established). The HDLC controller does not have to arbitrate for the GCI TIC bus, but it must wait for an asserted S/G from the transceiver before it receives its internal CTS and can transmit, as stated in the previous section. To relinquish the GCI TIC bus after a C/I0 or D-channel transmission, both the C/I0 request (a software request) and the HDLC controller request (a hardware request) must be deasserted. When the software request bit has been cleared (ending C/I0 transmission), the C/I0 channel is not allowed back onto the same established TIC bus should it remain open for a HDLC transmission. When the TIC bus is given up by the HDLC controller, neither the D-channel nor the C/I0 channel is allowed access to the TIC bus again for at least one GCI frame.*

6. After the completion of C/I0 data, software should remove its request by clearing its request bit. When done, the C/I0 channel control is withdrawn from the TIC bus (BAC is set back to 1 in the following frame as long as the HDLC controller has no D-channel communication in progress) and the C/I0 channel is prevented from accessing the TIC bus again for one GCI frame (i.e., the channel is moved into a lower priority as mentioned earlier in this chapter). This gives all connected devices an equal chance to access the TIC bus.

17.5.7.6 IC Channel Operation

The two IC channels have access to a single interrupt-driven microprocessor transmit/receive buffer. A register bit determines which channel gets access to this buffer. Because the data output is open-drain, the unused IC channel and all High bits of the chosen IC channel are placed in a high-impedance state (unless driven by an HDLC channel through a Time Slot Assigner).

17.5.8 Interrupts

The GCI controller can generate the following maskable interrupts (sharing one direct processor interrupt line) using the GISTAT and GIMSK registers.

- **IC Buffer Available or Buffer Empty:** Indicates that a byte of data has been received on the IC channel, and that a new IC byte can be loaded for transmission.
- **GCI Timing Request:** Response to GCI_DCL_A starting (going High) from the deactivated state.
- **Change in C/I1 Channel Status:** Indicates that the contents on the receive side of C/I channel 1 have changed since the C/I Receive Data register was last read.
- **Change in C/I0 Channel Status:** Indicates that the contents on the receive side of C/I channel 0 have changed since the C/I Receive Data register was last read.
- **Monitor Channel Receive Abort Detected:** Indicates an implied transmitter abort due to out-of-sequence transmit handshake bits or handshake bit transmission errors.
- **Monitor Channel Collision Detected:** Indicates that a collision has occurred on the monitor channel during the transmission of a monitor byte.

- **Monitor Channel Transmit Abort Request Received:** Indicates that an abort request has been received on the monitor channel. This indicates that the receiver on the other end of the Monitor channel has failed to receive the transmitted data correctly and is requesting that the current transmission be discontinued and the data transmission be repeated through software.
- **Monitor Channel End-of-Message Received:** Indicates that an EOM has been received on the monitor channel. This indicates that the message currently being received has concluded.
- **Monitor Channel Transmit Buffer Available:** Indicates that a new byte of data can be loaded into the Monitor Transmit Data register.
- **Monitor Channel Receive Data Available:** Indicates that a byte of data has been received on the monitor channel and is available in the Monitor Receive Data register.

17.5.9 Software-Related Considerations

To enable the GCI interface, software must set the MODE bit field to 10b in the TSA Channel A Configuration (TSACON) register. This is necessary regardless of whether TSA Channel A is being used.

17.5.10 Comparison to Other Devices

The Am186CC microcontroller's GCI interface is similar to the AMD Am79C30 in clock slave mode.

17.6 INITIALIZATION

On external and internal reset, the following occurs:

- The TSAs default to non-GCI mode.
- The GCI signals default to alternate functionality as shown in Table 17-1 on page 17-3.
- The EXSYNC bit of the SYSCON register is cleared, making the HDLC Channel C interface available for raw DCE or PCM highway operation.
- The MODE field of the TSxCON register is cleared, specifying raw DCE operation.
- The GCIDEN bit of the HxTCON1 register is cleared, disabling GCI D-Channel control of the HDLC channel.
- The MCHEN bit of the GPCON register is cleared, disabling both monitor channels.
- The MCHSEL bit of the GPCON register is cleared, selecting monitor channel 1.
- The ICSEL bit of the GPCON register is cleared, selecting IC channel 1.
- The BRDIS bit of the GPCON register is cleared, enabling bus reversal.
- The MXBA bit of the GISTAT register is set, indicating that a new byte of data can be loaded into the GMTD register.
- All GCI interrupts enables are cleared to 0 in the GIMSK register, masking the interrupts.
- The TICEN and ECHOEN bits are cleared to 0 in the GTIC register, disabling TIC bus access and D-channel echo compares, respectively.

CHAPTER

18 UNIVERSAL SERIAL BUS (USB)

CC CU

Note: *Only the Am186CC and Am186CU microcontrollers support USB.*

18.1 OVERVIEW

The Universal Serial Bus (USB) is an industry-standard bus architecture for computer peripheral attachment. The USB provides a single interface for easy, plug-and-play, hot-plug attachment of peripherals such as a keyboard, mouse, speakers, printers, scanners, and communication devices. The USB allows simultaneous use of many different peripherals with a combined transfer rate of up to 12 Mbit/s.

Both the Am186CC and Am186CU microcontrollers include a highly flexible integrated USB peripheral controller that designers can use to implement a variety of microcontroller-based USB peripheral devices for telephony, audio, or other high-end applications. These microcontrollers can be used in self-powered USB peripherals that use the full-speed signaling rate of 12 Mbit/s. They do not support the USB low-speed rate (1.5 Mbit/s). An integrated USB transceiver is provided to minimize system device count and cost, but an external transceiver can be used instead, if required.

The USB peripheral controller's features meet or exceed all of the USB device class resource requirements defined by the *USB Specification, Version 1.0*. This chapter refers to this version of the USB specification throughout. Consult the USB specification for details about overall USB system design. (At the time of this writing, the current USB specification and related information can be obtained on the Web at **www.usb.org**.)

The USB controller does not support USB host or hub functions. However, the Am186CC and Am186CU microcontrollers can be used to implement USB peripheral functions in a device that also contains separate USB hub circuitry.

The integrated USB peripheral controller provides a very efficient and easy-to-use interface, so that device software (or software) does not incur the overhead of managing low-level USB protocol requirements. Each of the controller's data endpoints is highly programmable and flexible, allowing the device to adapt to any USB host request that is made during the device configuration process. Because of the flexibility of the USB peripheral controller's endpoints, a design can allow its descriptors to be updated on-the-fly by the host's device driver, if necessary.

The USB peripheral controller hardware implements a number of USB standard commands directly; the rest can be implemented in device software. In addition, the USB peripheral controller provides a high degree of flexibility to help designers accommodate vendor- or device-class-specific commands, as well as any new features that might be added in future USB specifications.

The USB peripheral controller includes specialized hardware to support isochronous data transfers. Using the microcontroller's DMA features, isochronous transfers from an off-chip peripheral can be automatically synchronized to the USB data rate with little or no CPU overhead.

CC

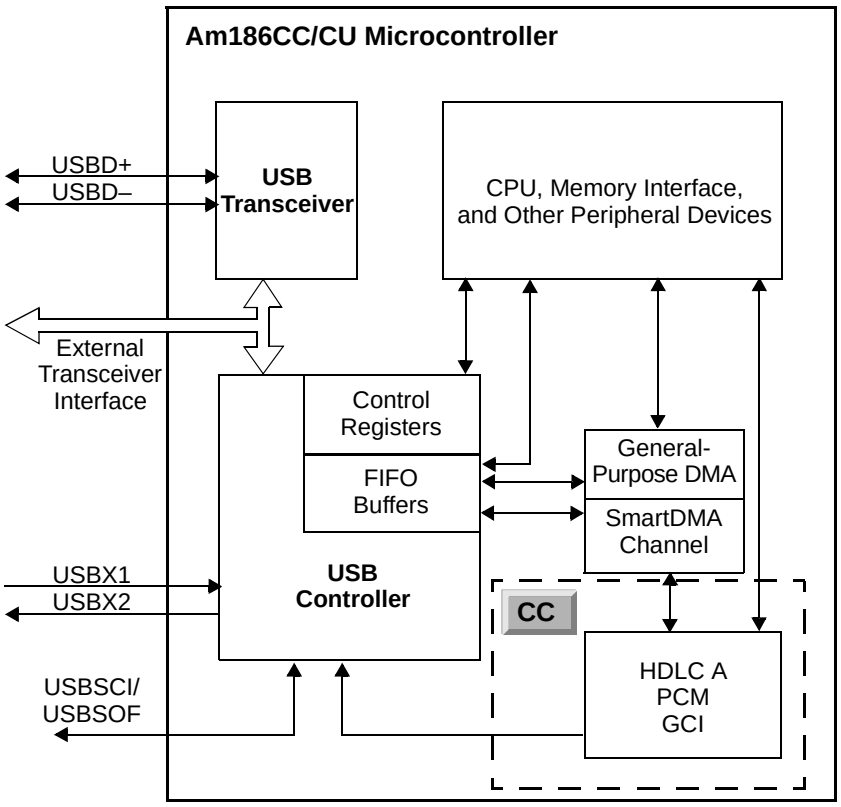
The Am186CC microcontroller also supports isochronous transfers from one of the integrated HDLC channels.

The USB peripheral controller also includes robust error detection and management features so the device software can manage transfers in any number of ways as required by the application. The USB suspend/resume, reset, and remote wake-up features are also supported.

18.2 BLOCK DIAGRAM

Figure 18-1 shows the block diagram for the USB peripheral controller.

Figure 18-1 USB Interface Block Diagram



18.3 SYSTEM DESIGN

The following sections describe pin multiplexing and feature trade-offs to consider when designing peripherals that use the USB peripheral controller.

18.3.1 Signal Trade-Offs

Table 18-1 lists the USB interface signals (including signals internal to the microcontroller) that are multiplexed with other microcontroller functions. Pinstraps are sampled only at reset and do not affect the pin's other functions, so they are not shown in this table. Other multiplexed signals, when enabled, either disable or alter any other functions that use the same pin.

Table 18-1 USB Multiplexed Signals

Signal	Function	Multiplexed Signal(s)	Default Signal
Internal USB Transceiver I/O Pins			
USBD+	Internal USB transceiver differential input/output	UDPLS	USBD+
USBD–	Internal USB transceiver differential input/output	UDMNS	USBD–
External USB Transceiver I/O Pins			
UDMNS	Status input from external transceiver	USBD–	USBD–
UDPLS	Status input from external transceiver	USBD+	USBD+
UTXDMNS	Output to the external transceiver differential driver	RSVRD_102	RSVRD_102
UTXDPLS	Output to the external transceiver differential driver	RSVRD_101	RSVRD_101
$\overline{\text{UXVOE}}$	External transceiver transmit output enable	RSVRD_103	RSVRD_103
UXVRCV	Receive input from external transceiver	RSVRD_104	RSVRD_104
USB Clock Inputs			
USBSCI	USB sample clock input	UCLK USBSOF PIO21	PIO21
USBSOF	USB start-of-frame synchronization output	UCLK USBSCI PIO21	PIO21
USBX1	USB peripheral controller crystal input	—	USBX1
USBX2	USB peripheral controller crystal output	—	USBX2

18.3.1.1 USB Transceiver Interface

By default, the USB peripheral controller utilizes an integrated USB transceiver to directly drive and receive data on the USBD+ and USBD– differential physical interface signals. This transceiver allows a USB peripheral device to be designed with the Am186CC or Am186CU microcontroller without requiring the additional board space needed by a discrete transceiver device. The USB device terminator requirement is not integrated, so the designer should add a single 1.5 K-Ω pullup resistor on the USBD+ signal to indicate this is a full-speed device.

The USB specification requires a driver impedance between 29 Ω and 44 Ω on the USBD+ and USBD– signals. The CMOS drivers used have a much lower impedance, so matching resistors must be placed in series on the USBD+ and USBD– signals as shown in Figure 18-2 on page 18-4 and Figure 18-3 on page 18-5.

18.3.1.2 Programmable Connect and Disconnect

Because the microcontroller is meant to be in a self-powered application, there are a few issues to resolve to meet USB specifications. For a full-speed device, the USB specification requires a 1.5 K-Ω device terminator to a 3.0-V to 3.6-V voltage source on USBD+, derived from or controlled by the power supplied by the USB cable (V_{USB}), that does not supply current when V_{USB} is unpowered or removed. In a self-powered USB application, the USB host/hub also cannot supply current to the USB device when the device is unpowered.

To help meet these criteria, the microcontroller is programmable to disable and three-state the internal transceiver differential outputs. The other requirement is to disable the external

1.5 K-Ω pull up on USBD+ when V_{USB} is removed. The following system design issues should be resolved to provide a robust self-powered USB device application:

On Connect:

- Monitor V_{USB} to identify a powered USB host/hub.
- Enable the 1.5 K-Ω pullup on USBD+ to signal a connect condition to the host/hub.

On Disconnect:

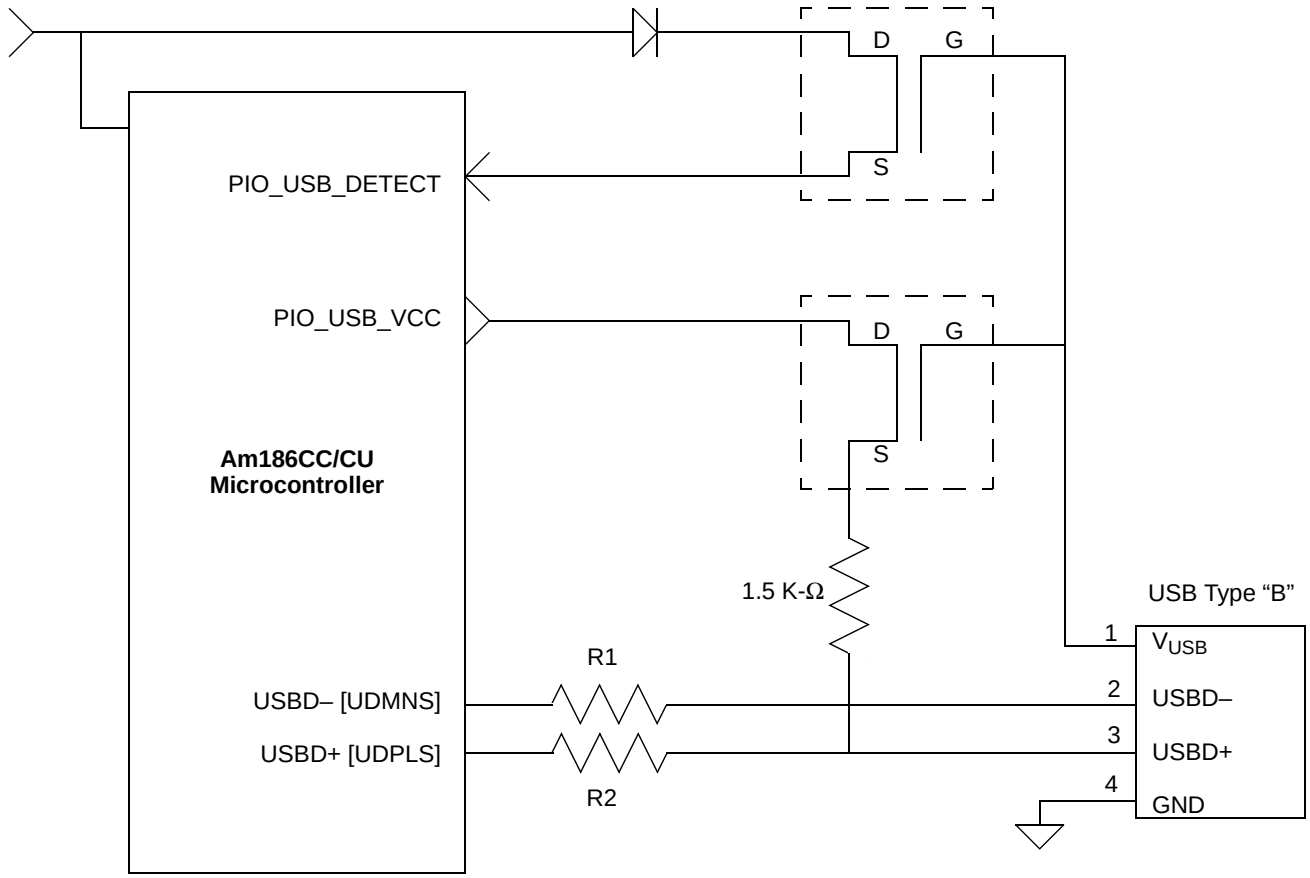
- Monitor V_{USB} to identify power being removed from the USB host/hub.
- Three-state USBD+/USBD– outputs.
- Remove power from 1.5 K-Ω pull up.

On both Connect and Disconnect:

- Isolate V_{USB} from the USB device when the device is unpowered.

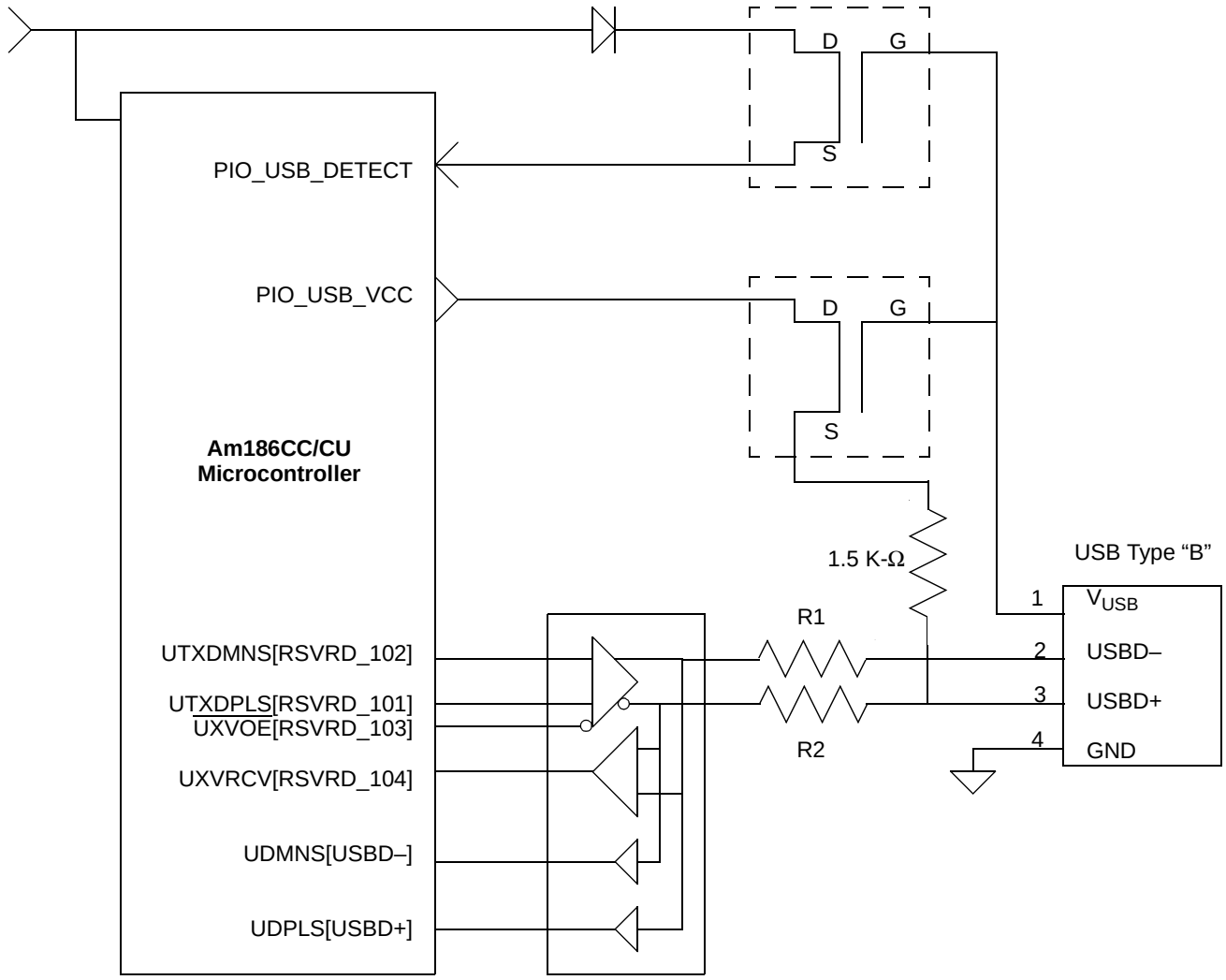
Figure 18-2 illustrates a circuit diagram of an example application using the internal transceiver. Figure 18-3 illustrates a circuit diagram of an example application using the external transceiver.

Figure 18-2 USB With Internal Transceiver



Note: *The USB specification requires a driver impedance between 29 Ω and 44 Ω on the USBD+ and USBD– signals. For information about driver characteristics and selecting a series resistor value, see the data sheets for the Am186CC and Am186CU microcontrollers.*

Figure 18-3 USB With External Transceiver



Note: *The USB specification requires a driver impedance between 29 Ω and 44 Ω on the USBD+ and USBD– signals. For information about driver characteristics and selecting a series resistor value, see the documentation for the external transceiver.*

If necessary, to enable interface signals for an external transceiver, disable the integrated USB transceiver by asserting the $\overline{\text{USBXCVR}}$ pinstrap at reset (power-on or assertion of $\overline{\text{RES}}$). Table 18-1 on page 18-3 lists all USB signals, plus information about multiplexed functions.

Note: *Before using either the internal USB transceiver or the external USB transceiver interface, software must set the PUP_XCVER bit in the USB Device Miscellaneous Functions (USBMFR) register to power up the USB transceiver and enable the transceiver interface.*

18.3.1.3 USB Clock Source

The USB peripheral controller hardware requires a 48-MHz clock input for proper operation. The USB peripheral controller can be driven directly from the primary system clock if the primary system clock is operating at 48 MHz. Otherwise, use a dedicated USB clock source so that the primary microcontroller system clock and the USB clock are independent of each other. When the dedicated USB clock source is used, the only requirement is that *the*

primary system clock must be a minimum of 24 MHz when using the USB peripheral controller.

To select the dedicated USB clock source, assert either the $\overline{\text{USBSEL2}}$ or $\overline{\text{USBSEL1}}$ pinstrap during reset (power-on or assertion of $\overline{\text{RES}}$). These pinstraps select either 4x or 2x PLL operation, allowing the use of a 12-MHz or 24-MHz crystal, respectively, as the USB clock input on pins USBX1 and USBX2. Table 18-2 lists the permutations of the USB PLL mode pinstraps.

Table 18-2 USB PLL Mode Pinstraps

{$\overline{\text{USBSEL1}}$}	**{$\overline{\text{USBSEL2}}$}**	**USB PLL Mode**
1	1	Use CPU clock, USB PLL disabled (default)
0	1	4x, USB PLL enabled
1	0	2x, USB PLL enabled
0	0	Reserved

18.3.1.4 Isochronous Synchronization Signals

The USBSCI signal input and USBSOF signal output provide for isochronous transfer synchronization, which is described on page 18-23. These signals are multiplexed on the same pin with the UART external clock input signal (UCLK).

Enabling the USBSOF signal output (by setting the ESOF_EN bit in the Isochronous Synchronization Control (ISCTL) register) overrides the USBSCI signal input if that signal is also selected (through the SAM_CLK_SEL field in the ISCTL register.)

Do not enable the USBSOF signal output at the same time as the UART external clock input (UCLK). The UCLK signal is enabled by the XTRN bit in the High-Speed Serial Port Control 1 (HSPCON1) or Serial Port Control 1 (SPCON1) registers.

The USBSCI and UCLK signal inputs can be enabled at the same time, but it is unlikely that the same signal source can be used as an input for both of these functions.

18.3.2 DMA Trade-Offs

The microcontroller contains two different kinds of DMA channels, general-purpose DMA and SmartDMA channels. The USB data endpoints can use either kind. Choosing which type of DMA channel to use, if any, involves the following system trade-offs:

CC

- The integrated HDLC controllers in the Am186CC microcontroller can use only the SmartDMA channel. Consequently, if all four HDLC controllers are to be used with DMA (for high-bandwidth HDLC connections), then the USB can use only general-purpose DMA or no DMA.
- Other integrated peripherals such as the UARTs and the external DMA request lines can use only the general-purpose DMA channels.
- For USB bulk endpoints, SmartDMA channels have advantages over general-purpose DMA channels that can result in higher performance and lower software overhead, especially when each transaction is relatively small. When most transactions are relatively large, general-purpose DMAs may have a small performance advantage over SmartDMA channels.
- For USB isochronous endpoints with true streaming data, general-purpose DMAs are slightly easier to use than SmartDMA channels.

- Each USB data endpoint can only be connected to a single specific SmartDMA channel, but can be connected to any general-purpose DMA channel. Because SmartDMA channels are directional (either transmit or receive), a general-purpose DMA channel must be used if more than 2 IN data endpoints or more than 2 OUT data endpoints are desired.

For more about DMA and other I/O options, see “Handling USB Data” on page 18-18.

18.4 REGISTERS

The registers listed in Table 18-3 program the USB peripheral controller. There are four general configuration registers, six miscellaneous control and status registers, ten registers for the dedicated control and interrupt endpoints, and eight registers each for the four data endpoints.

Appendix A summarizes the bits in all the registers. For a complete description of all the peripheral registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

Table 18-3 USB Register Summary

Offset	Register Mnemonic	Register Name	Description
USB General Configuration Registers			
1E0h	UISTAT1	USB Interrupt Status 1	Common status register for interrupt-capable status bits of each USB endpoint.
1E2h	UIMASK1	USB Interrupt Mask 1	Enables or disables interrupts generated by UISTAT1 bits.
1E4h	UISTAT2	USB Interrupt Status 2	Shows status of USB and controller features.
1E6h	UIMASK2	USB Interrupt Mask 2	Enables or disables interrupts generated by UISTAT2 status bits.
USB Miscellaneous Registers			
1E8h	USBMFR	USB Device Miscellaneous Functions	Provides internal USB transceiver power and disable control; USB suspend status; USB soft reset control; USB self-powered device attribute; and remote wake-up control/status.
1EAh	RTFMCNT	Real Time Frame Monitor Count	Used to estimate progress of the current host frame.
1ECh	TSTMP	Time Stamp	Contains the current frame number.
1EEh	TSTMPM	Time Stamp Match	Match register for time stamp bit in UISTAT2.
1F0h	ISCTL	Isochronous Synchronization Control	Used to control external frame synchronization (USBSOF signal enable), auto-rate bytes per sample, USB sample source clock; and FPMCNT bit latch count rate.
1F2h	FPMCNT	Frame Position Monitor Count	Used during isochronous IN transfers to compare the source data rate (USBSCI signal) to the USB host’s data rate. The host’s SOF rate or data rate can then be adjusted accordingly. CC CH In the Am186CC microcontroller, the HDLC A/PCM/GCI source data rate can also be compared to the USB host’s data rate.

Table 18-3 USB Register Summary (Continued)

Offset	Register Mnemonic	Register Name	Description
USB Control Endpoint Registers			
200h	CNTCTL	Control Endpoint Control/Status	Contains control and status bits for the Control endpoint (endpoint 0).
202h	CNTSIZ	Control Endpoint Receive Packet Size	Shows the size of the packet present in the Control endpoint's FIFO.
206h	CNTDAT	Control Endpoint Data Port	Used to read from or write to the Control endpoint's FIFO. The FIFO address pointer is advanced on each access.
208h	CNTRPK	Control Endpoint Receive Data Port Peek	Used to read the current value in the Control endpoint's FIFO without advancing the FIFO pointer.
20Ah	CNTDEF1	Control Endpoint Definition 1	Shows the control endpoint's definition: endpoint number, configuration, interface, alternate setting, direction, and type. These parameters are fixed for the control endpoint.
20Ch	CNTDEF2	Control Endpoint Definition 2	Shows the control endpoint's FIFO size and maximum packet size. These parameters are fixed for the control endpoint.
USB Interrupt Endpoint Registers			
210h	IEPCTL	Interrupt Endpoint Control/Status	Contains control and status bits for the Interrupt endpoint.
216h	IEPDAT	Interrupt Endpoint Data Port	Used to write to the interrupt endpoint's FIFO. The FIFO address pointer is advanced on each access.
21Ah	IEPDEF1	Interrupt Endpoint Definition 1	Used to set the interrupt endpoint's definition: endpoint number, configuration, interface, alternate setting. The interrupt endpoint's direction (IN) and type (interrupt) are fixed.
21Ch	IEPDEF2	Interrupt Endpoint Definition 2	Used to set the interrupt endpoint's FIFO size (fixed for the interrupt endpoint) and maximum packet size.
USB Data Endpoint A Registers			
220h	AEPCTL	A Endpoint Control/Status	Contains control, status, and error bits for the endpoint.
222h	AEPSIZ	A Endpoint Received Packet Size	Shows the size of the packet present in the endpoint's FIFO.
224h	AEPBUFS	A Endpoint Buffer Status	Shows the number of bytes present in the endpoint's FIFO.
226h	AEPDAT	A Endpoint Data Port	Used to read from or write to the endpoint's FIFO. The FIFO address pointer is advanced on each access.
228h	ARCVPK	A Endpoint Receive Data Port Peek	Used to read the current value in the endpoint's FIFO without advancing the FIFO pointer.
22Ah	AEPDEF1	A Endpoint Definition 1	Used to set the endpoint's definition: endpoint number, configuration, interface, alternate setting, direction, and type.

Table 18-3 USB Register Summary (Continued)

Offset	Register Mnemonic	Register Name	Description
22Ch	AEPDEF2	A Endpoint Definition 2	Used to set the endpoint's FIFO size and maximum packet size.
22Eh	AEPDEF3	A Endpoint Definition 3	Used to set auto-rate enable, status interrupt mask, transfer mode, and stop mask.
USB Data Endpoint B Registers			
230h	BEPCTL	B Endpoint Control/Status	Behaves the same as the Endpoint A registers, but for Endpoint B.
232h	BEPSIZ	B Endpoint Received Packet Size	
234h	BEPBUFS	B Endpoint Buffer Status	
236h	BEPDAT	B Endpoint Data Port	
238h	BRCVPK	B Endpoint Receive Data Port Peek	
23Ah	BEPDEF1	B Endpoint Definition 1	
23Ch	BEPDEF2	B Endpoint Definition 2	
23Eh	BEPDEF3	B Endpoint Definition 3	
USB Data Endpoint C Registers			
240h	CEPCTL	C Endpoint Control/Status	Behaves the same as the Endpoint A registers, but for Endpoint C, except that Endpoint C and D have two additional FIFO size options: 32 and 64 bytes.
242h	CEPSIZ	C Endpoint Received Packet Size	
244h	CEPBUFS	C Endpoint Buffer Status	
246h	CEPDAT	C Endpoint Data Port	
248h	CRCVPK	C Endpoint Receive Data Port Peek	
24Ah	CEPDEF1	C Endpoint Definition 1	
24Ch	CEPDEF2	C Endpoint Definition 2	
24Eh	CEPDEF3	C Endpoint Definition 3	
USB Data Endpoint D Registers			
250h	DEPCTL	D Endpoint Control/Status	Behaves the same as the Endpoint A registers, but for Endpoint D, except that Endpoint C and D have two additional FIFO size options: 32 and 64 bytes.
252h	DEPSIZ	D Endpoint Received Packet Size	
254h	DEPBUFS	D Endpoint Buffer Status	
256h	DEPDAT	D Endpoint Data Port	
258h	DRCVPK	D Endpoint Receive Data Port Peek	
25Ah	DEPDEF1	D Endpoint Definition 1	
25Ch	DEPDEF2	D Endpoint Definition 2	
25Eh	DEPDEF3	D Endpoint Definition 3	

18.5 OPERATION

The Am186CC and Am186CU microcontrollers act as USB peripheral devices. The USB is a half-duplex, master/slave, polled bus. In other words, the microcontroller only transmits on the USB in response to a request from the USB host, usually a personal computer. There can be only one transmitter on the USB at a time.

When the USB host addresses a peripheral, it also addresses a particular endpoint on that device. Each endpoint is configured with a logical number that the USB host uses to address that endpoint. No two endpoints can be configured with the same logical number.

The endpoint responds to the host's requests, sending or receiving device data. In USB nomenclature, data flowing *from* the host travels in the OUT direction, and data flowing *to* the host travels in the IN direction. Because the Am186CC or Am186CU microcontroller resides in a USB peripheral, its OUT endpoints receive data, and its IN endpoints transmit data.

Each endpoint is supported by a first-in-first-out buffer (FIFO). The FIFO is a temporary storage location for the data that is passed between the microcontroller's CPU or memory bus and the integrated USB peripheral controller.

The microcontroller supports six endpoints:

- One dedicated control endpoint (Endpoint 0)
- One dedicated interrupt endpoint
- Four fully programmable data endpoints (named A–D)

The following sections describe the USB endpoints and explain how to use them.

18.5.1 Usage

This section briefly lists the tasks that software must perform to program the USB peripheral controller for various applications. The following programming tasks do not cover all possibilities. They are intended to provide a basic understanding of USB register usage. The user should program the registers appropriately for each specific application.

Many of the subjects mentioned in the following lists are discussed more thoroughly elsewhere in this chapter.

18.5.1.1 General USB Peripheral Controller Programming Issues

- Always power up the transceiver (internal or external) by setting the PUP_XCVR bit in the USB Device Miscellaneous Functions (USBMFR) register.
- Always configure an endpoint's definition registers before enabling the endpoint. Changing the endpoint register values while the endpoint is enabled could result in unpredictable behavior.
- When using USB status bits as interrupt sources, be sure to program the interrupt Channel 2 Control (CH2CON) register to enable the channel and select its internal source (USB).
- Refer to the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916 for register default values and details about using each register field.

18.5.1.2 Programming the Control Endpoint

The host uses the USB peripheral controller's dedicated control endpoint for detection and control of the device. The endpoint contains an 8-byte FIFO for storage of commands, command data (for host command writes), and responses (for host command reads). The maximum packet size of the control endpoint is always eight bytes, the physical size of the FIFO.

The control endpoint's number is fixed at 0. All USB devices must have a control endpoint with endpoint number 0. The USB host uses this endpoint to initialize and control the device. Endpoint 0 gives the host access to the device's configuration information (device descriptors) and overall status. All of the USB standard and vendor- or device-class-specific commands are directed to this endpoint. See "Command Handling" on page 18-26.

The following registers configure the control endpoint:

- Control Endpoint Definition 1 (CNTDEF1):

  All fields in this register are read only. They can be read to determine endpoint attributes.

- Control Endpoint Definition 2 (CNTDEF2):

  All fields in this register are read only. They can be read to determine endpoint attributes.

- Control Endpoint Control/Status (CNTCTL):
  - The endpoint enable bit (EP_EN) enables or disables the endpoint.

    Device software can enable the endpoint in the stalled state by clearing the EP_NOT_STALLED bit while the EP_EN bit is being set. The hardware, however, sets the EP_NOT_STALLED bit upon reception of a SETUP packet from the host.

  - Device software can be interrupted by two sources: the ACT_REQ bit or the NEW_COMMAND bit. To enable these bits as interrupt sources, set the CNT_EP_ACT and CNT_EP_NEW bits in the USB Interrupt Mask 1 (UIMASK1) register.

    On reset, hardware owns the FIFO (indicated by the cleared ACT_REQ bit). After hardware fills the FIFO, it sets the ACT_REQ bit to transfer ownership of the FIFO to software. The first time software owns the FIFO, it clears the NEW_COMMAND bit and proceeds with writing data to or reading data from the FIFO. Software can use the Control Endpoint Receive Packet Size (CNTSIZ) register to determine the number of valid data bytes present in the endpoint FIFO. After software has completed its tasks, it hands the FIFO back to the hardware by clearing the ACT_REQ bit. At the end of data (the last time software puts data in the FIFO), software also clears the COMMAND_BUSY bit. Hardware then sets the ACT_REQ bit once for the last packet, and then again for the end of command. For more information, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

18.5.1.3 Programming the Interrupt Endpoint

Because USB is polled, this is not an interrupt in the traditional sense. When the device wishes to interrupt the host, it returns data when the interrupt endpoint is polled. The USB peripheral controller's dedicated interrupt endpoint contains a 16-byte FIFO, which software loads with the data to return on the next poll from the host. The maximum packet size can be set to eight or 16 bytes. For more information, see "Interrupt Endpoint Programming" on page 18-29.

The host polls the interrupt endpoint once every 1 to 255 ms. Device software requests a poll rate when it sets up the endpoint's descriptor data structure, which the host obtains by issuing a GET_DESCRIPTOR command during device configuration.

Note that the interrupt endpoint can only be used in non-DMA mode.

The following registers are used to configure the interrupt endpoint in response to commands received from the USB host:

- Interrupt Endpoint Definition 1 (IEPDEF1):
  - Based on the SET_CONFIGURATION command, the device software should write the EP_CFG field in the IEPDEF1 register.
  - Based on the SET_INTERFACE command, the device software should write the EP_INT and EP_ASET fields in the IEPDEF1 register.
  - Based on the endpoint descriptor associated with the alternate setting, the device software should write the EP_NUM field in the IEPDEF1 register.
- Interrupt Endpoint Definition 2 (IEPDEF2):

  Endpoint maximum packet value can be programmed to a value of 8 or 16.
- Interrupt Endpoint Control/Status (IEPCTL):
  - The endpoint enable bit (EP_EN) enables or disables the endpoint.
  - Initial control of the data FIFO is assigned to software. Device software can therefore write to the endpoint FIFO (IEPDAT). After writing to the FIFO, the software should clear the ACT_REQ bit, thereby giving control back to the USB endpoint hardware and allowing it to transmit the written data.
  - Hardware sets the ACT_REQ bit after the endpoint has successfully sent a data packet to the host and the packet has been acknowledged. To enable the ACT_REQ bit as an interrupt source, set the INT_EP_ACT bit in the UIMASK1 register.
  - There is a feature that allows the device software to update stale data if it has not been transmitted. This is done by clearing the NOT_FLUSH bit, which causes the hardware to revert control to the device software by setting the ACT_REQ bit. Note that the ACT_REQ bit is set only if there is no active data transfer from this endpoint to the host. Device software can verify if the ACT_REQ bit is set and if it is, can update stale data by writing to the FIFO (IEPDAT).

18.5.1.4 Programming Data Endpoints

The USB peripheral controller provides four data endpoints.Two have 16-byte FIFOs, and the other two have 64-byte FIFOs. Each data endpoint is individually programmable as to direction (IN or OUT relative to the host), transfer type (bulk, isochronous, or interrupt), and maximum packet size. The maximum packet size set for these endpoints can be greater than the FIFO's physical size if using a general-purpose DMA or SmartDMA channel. (Note that the endpoints have differences in how they interface to the SmartDMA channels.) Legal maximum packet sizes are any power of 2 between 8 and 64 for data endpoints configured for bulk transfers, and any integer up to 1023 for data endpoints configured for isochronous transfers.

The four endpoints are named A, B, C, and D. Where the following description applies to any of them, an "x" is used in the register name in place of the endpoint name.

The following registers configure a data endpoint in response to commands received from the USB host. For details on any of these registers, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916.

- Endpoint Definition 1 (xEPDEF1):
  - Based on the SET_CONFIGURATION command, the device software should write the EP_CFG field in the xEPDEF1 register.
  - Based on the SET_INTERFACE command, the device software should write the EP_INT and EP_ASET fields in the xEPDEF1 register.
  - The Endpoint number should be configured through the EP_NUM field in the xEPDEF1 register.
  - Endpoint direction and Endpoint type should be configured through the EP_DIR and EP_TYPE fields, respectively.
- Endpoint Definition 2 (xEPDEF2):

  FIFO size and endpoint maximum packet value fields can be programmed. The values depend on the endpoint type selection. (Endpoint A and B FIFOs can be 8 or 16 bytes. Endpoint C and D FIFOs can be 8, 16, 32, or 64 bytes.)
- Endpoint Definition 3 (xEPDEF3):
  - Based on application requirements, the appropriate interrupt mask and stop mask fields are programmable.
  - The MODE field can configure the endpoint. This determines how the endpoint interfaces with system memory or another peripheral's data port.
  - To enable the auto rate feature, use the AUTO_RATE_EN field. Note that this feature only applies to an endpoint that is configured as an isochronous IN endpoint and interfaces with a DMA mode. This feature requires additional programming in the Isochronous Synchronization Control (ISCTL) register. For more information, see “Isochronous Transfer Features” on page 18-24.
- Endpoint Received Packet Size (xEPSIZ):

  This is a status register that provides information on the size of the received packet (in bytes) when the endpoint is configured for the OUT direction.
- Endpoint Buffer Status (xEPBUFS):

  This is a status register that provides information on the number of bytes, if any, is in the endpoint FIFO.
- Endpoint Data Port (xEPDAT):

  Device software or the DMA controller uses this register to read/write to the endpoint FIFO. A valid access to this register increments the address pointer.
- Endpoint Receive Data Port Peek (xRCVPK):

  Debug or emulator software uses this register to read the endpoint data FIFO *without* advancing the address pointer. It is only applicable for the OUT direction.

■ Endpoint Control/Status (xEPCTL):

This register controls various aspects of the data endpoint. Because the data endpoint is flexible in terms of the endpoint type, direction, and mode, besides other programmable features, use of this register is discussed in the following specific application scenarios.

18.5.1.4.1 Endpoint A Configured as Bulk OUT, Non-DMA Mode

1. Program the Endpoint A registers:
   – Set EP_TYPE = 10b (Bulk) and EP_DIR = 1 (OUT) in the AEPDEF1 register.
   – Set MODE = 000b (Non-DMA) in the AEPDEF3 register.
   – If interrupts are to be used, set the appropriate bits in the UIMASK1 or UIMASK2 register. If endpoint status bits are to generate interrupts, also set the appropriate mask bits in the AEPDEF3 register.
   – Perform any additional programming of the definition registers that is required for the specific application.
2. Enable the data endpoint by setting the EP_EN bit in the AEPCTL register.
3. Assign initial control of the data FIFO to the USB endpoint hardware by clearing the ACT_REQ bit in the IEPCTL register.

   Data sent by the host is written to the data FIFO. At the end of a successful transfer, hardware sets the ACT_REQ bit to transfer control to software. To enable this bit as an interrupt source, set the A_EP_ACT bit in the UIMASK1 register.
4. When the ACT_REQ bit is set, read the endpoint FIFO (AEPDAT). Note that the number of bytes written by the host can be obtained from the AEPSIZ register.
5. After reading the appropriate number of bytes from the FIFO, clear the ACT_REQ bit to give control back to the USB endpoint hardware and allow it to reuse the FIFO for subsequent data.

18.5.1.4.2 Endpoint A Configured as Bulk IN, Non-DMA Mode

1. Program the Endpoint A registers:
   – Set EP_TYPE = 10b (Bulk) and EP_DIR = 0 (IN) in the AEPDEF1 register.
   – Set MODE = 000b (Non-DMA) in the AEPDEF3 register.
   – If interrupts are to be used, set the appropriate bits in the UIMASK1 or UIMASK2 register. If endpoint status bits are to generate interrupts, also set the appropriate mask bits in the AEPDEF3 register.
   – Perform any additional programming of the definition registers that is required for the specific application.
2. Enable the data endpoint by setting the EP_EN bit in the AEPCTL register.
3. For an IN endpoint, initial control of the data FIFO is assigned to the device. Device software can therefore write to the endpoint FIFO (AEPDAT) when it is ready.
4. After it is finished writing to AEPDAT, software should clear the ACT_REQ bit in the IEPCTL register to give control back to the USB endpoint hardware and allow it to transmit the written data.

   Hardware sets the ACT_REQ bit after the endpoint has successfully sent a data packet to the host and the packet has been acknowledged. To enable the ACT_REQ bit as an interrupt source, set the A_EP_ACT bit in the UIMASK1 register.

5. Device software can clear the NOT_FLUSH bit in the AEPCTL register if it needs to update stale data in the FIFO before the data is transmitted. This causes the USB hardware to return control to software by setting the ACT_REQ bit. However, the ACT_REQ bit is set only if there is no active data transfer from this endpoint to the host. Device software can verify if the ACT_REQ bit is set and if it is, can update stale data by writing to the FIFO (AEPDAT).

18.5.1.4.3 *Endpoint C Configured as Bulk OUT, General-Purpose DMA Mode*

1. Program the endpoint C registers:
   - Set EP_TYPE = 10b (Bulk) and EP_DIR = 1 (OUT) in the CEPDEF1 register.
   - Set MODE = 010b or 011b (general-purpose DMA) in the CEPDEF3 register. These modes behave the same when used for an OUT endpoint.
   - Set the appropriate interrupt mask bits in the UIMASK1 or UIMASK2 register. For general-purpose DMA operation, enable the appropriate interrupt mask bits in the CEPDEF3 register to allow the device software to be notified of the FIFO status. For example, the FULL_PKT, SHORT_PKT, OTHER_ERROR, or BUFFER_ERROR status bits could stop the hardware, requiring device software to take appropriate action and then clear the ACT_REQ bit to let the hardware continue.
   - Perform any additional programming of the definition registers that is required for the specific application.
2. In the General-Purpose DMA Control 0 (GDxCON0) register for an available general-purpose DMA channel, set DSEL = 11100b (USB Endpoint C, source-synchronized). Make any other DMA channel configuration settings that are required, then set ST = 1 in the GDxCON0 register to enable the DMA channel. Enable the DMA channel before enabling the DMA request source to avoid data loss or initial error conditions.

   It is important to note that in DMA mode, the ACT_REQ bit no longer serves as a semaphore lock for the data FIFO. The data FIFO now behaves as a circular FIFO with simultaneous read/write capability. The ACT_REQ bit acts as a Stop/Go bit for the hardware. For details, see the xEPCTL register description in the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916. If software sets the endpoint's ACT_REQ bit, the DMA transfer stops until software clears the bit again.

18.5.1.4.4 *Endpoint C Configured as Bulk IN, General-Purpose DMA Mode with Terminal Count Not Ignored*

1. Program the Endpoint C registers:
   - Set EP_TYPE = 10b (Bulk) and EP_DIR = 0 (IN) in the CEPDEF1 register.
   - Set MODE = 011b (general-purpose DMA, terminal count not ignored) in the CEPDEF3 register.

     In this mode, when the terminal count for the general-purpose DMA channel reaches zero, the byte of data written is marked as the last byte in the USB endpoint FIFO. If the transfer size is an integer multiple of the maximum packet size, device software can write a zero byte to the endpoint FIFO by clearing the NOT_ZERO bit in the CEPCTL register and following that with a dummy write to the CEPDAT register. The NOT_ZERO bit is set automatically when the data port is written.
   - Set the appropriate interrupt mask bits in UIMASK1 or UIMASK2. For general-purpose DMA operation, enable the appropriate interrupt mask bits in the CEPDEF3 register to allow the device software to be notified of the FIFO status. For example, the FULL_PKT, SHORT_PKT, OTHER_ERROR, or BUFFER_ERROR status bits could

stop the hardware, requiring device software to take appropriate action and then clear the ACT_REQ bit to let the hardware continue.

– Perform any additional programming of the definition registers that is required for the specific application.

2. In the General-Purpose DMA Control 0 (GDxCON0) register for an available general-purpose DMA channel, set DSEL = 11101b (USB Endpoint C, destination synchronized). Make any other DMA channel configuration settings that are required, then set ST = 1 in the GDxCON0 register to enable the DMA channel. Enable the DMA channel before enabling the DMA request source to avoid data loss or initial error conditions.

   It is important to note that in DMA mode, the ACT_REQ bit no longer serves as a semaphore lock for the data FIFO. The data FIFO now behaves as a circular FIFO with simultaneous read/write capability. The ACT_REQ bit acts as a Stop/Go bit for the hardware. For details, see the xEPCTL register description in the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916. If software sets the endpoint's ACT_REQ bit, the DMA transfer stops until software clears the bit again.

18.5.2 Data Transmission and Data Types

For the Am186CC and Am186CU microcontrollers, all communication across the USB takes place in Full-speed mode. USB bus transactions involve transmissions in up to three types of packets: token, data, and handshake. The token packet contains information about the type and direction of the transaction as well as the device address and which endpoint to use. The data packet, if any, contains actual commands or data. There can be one data packet, none, or more than one in a transaction. The format of a data packet varies according to what type of endpoint is being used. The handshake packet contains information regarding whether or not the transaction was completed successfully.

When beginning a transfer, the host issues a start-of-frame (SOF) packet. When the USB peripheral controller decodes this packet, it indicates the start-of-frame in the USB Interrupt Status 2 (UISTAT2) register. Also decoded in the start-of-frame packet is a time stamp, which the USB peripheral controller places in the Time Stamp (TSTMP) register.

18.5.2.1 USB Suspend, Resume, and Remote Wakeup

A USB Suspend is indicated if traffic across the USB cable ceases for 3 ms or more. This causes the USB peripheral controller to go into Suspend mode, which hardware indicates by setting both the SUSP bit in the USB Device Miscellaneous Functions (USBMFR) register and the USB_SUS bit in the USB Interrupt Status 2 (UISTAT2) register. The USB_SUS can be enabled as an interrupt source by setting the corresponding bit in the UIMASK2 register.

When a USB Suspend is detected, software should take any necessary action and wait for a USB Resume, which hardware indicates by clearing the SUSP bit in the USBMFR register and by setting the USB_RES bit in the UISTAT2 register. USB_RES can also be enabled as an interrupt source by setting the corresponding bit in the UIMASK2 register.

The Remote Wakeup feature is provided for peripheral devices that might need to wake up the USB remotely. The device's Remote Wakeup feature must be enabled by the host, which does so by issuing an appropriate SET_FEATURE command to the device. This automatically sets the RWAKE_EN bit in the USBMFR register.

If the RWAKE_EN bit is set and the controller is in USB Suspend mode, device software can initiate a USB Resume by setting the RWAKE bit in the USBMFR register.

18.5.2.2 USB Reset

Hardware sets the USB_RST bit in the UISTAT2 register when a USB reset signal is detected on the USB bus. The USB_RST bit can be enabled as an interrupt source by setting the corresponding bit in the UIMASK2 register.

18.5.2.3 USB Protocol Handling, IN Direction

For endpoints that are configured for the IN direction (transmit), data to be sent to the USB host is placed in the endpoint FIFO by the device software or the DMA controller.

When the USB host issues an IN token packet to the endpoint, the controller hardware converts the data stored in the endpoint's FIFO into a serial data stream, computes the CRC, performs the bit stuffing operation, and generates the NRZI converted data stream. At the same time, it assembles the data packet in the correct format, including the SYNC, PID, DATA, CRC, and EOP fields as required by the USB specification.

Finally, the USB peripheral controller drives the data stream out to the USB host through the integrated transceiver drivers on the USBD+ and USBD– signal lines (or to an external transceiver if one is used).

The device software is responsible for filling the endpoint's FIFO before starting the transaction, and for keeping the FIFO full during the transaction, if necessary. If a packet error occurs, the device software is responsible for responding appropriately. For bulk transfers, this entails refilling the endpoint FIFO with the data that was sent in the last frame. For isochronous transfers, the software must proceed to fill the FIFO with data for the next packet. For more information, see "Error Recovery on Bulk and Interrupt Endpoints" on page 18-22 and "Error Recovery on Isochronous Endpoints" on page 18-23.

The controller hardware automatically generates the appropriate USB handshake packets for the various transfer types. The device software can cause the endpoint to enter its stalled condition when appropriate.

18.5.2.4 USB Protocol Handling, OUT Direction

For endpoints that are configured in the OUT direction (receive), the USB peripheral controller receives the serial data stream from the USB host. The USB peripheral controller hardware identifies the incoming SYNC field, performs the NRZI-to-NRZ conversion, performs the bit-stripping operation, decodes the PID, and tests the device's ADDR and ENDP fields. For packets that are addressed to the device and an enabled endpoint number, the USB peripheral controller performs the serial-to-parallel conversion and places the data into the endpoint's FIFO.

During this process, the endpoint hardware checks the packet's CRC, Data toggle sense, and all handshake packets. In addition, the controller hardware monitors the number of data bytes sent by the host. If the number of bytes sent exceeds the endpoint's maximum packet size, the USB peripheral controller automatically flags an error and sends a negative acknowledge packet to the host if the endpoint type is bulk, control, or interrupt.

The primary responsibility of the device software or DMA controller is to move data written into the endpoint's FIFO to system memory or some other microcontroller peripheral. The device software must also monitor the USB peripheral controller to service packet errors that are detected during reception.

The appropriate USB handshake packets are generated automatically by the controller hardware for the various transfer types and error conditions. The device software can also cause the endpoint to enter its stalled condition when appropriate.

For the USB control endpoint, the system software is responsible for decoding and servicing several of the USB standard commands and all device class or vendor specific commands. Hardware is provided that allows the system software to detect incoming commands, and respond appropriately. The hardware also allows the software to detect all command abort scenarios.

18.5.3 Handling USB Data

The USB peripheral controller handles all of the low-level USB protocol requirements in hardware. Data movement between device memory or other microcontroller peripherals and the USB peripheral controller's endpoints is managed by device software executing on the microcontroller CPU.

The device software can use status polling or interrupts to handle FIFO data for any endpoint (control, interrupt, and A–D). In addition, the data endpoints (A–D) support either general-purpose DMA or SmartDMA channel transfers.

Device software sets up the method of operation for the endpoints by programming control and definition registers. There are register bits to enable or disable interrupts that can be generated as data transfers proceed, or the software can poll status bits to determine the status of each endpoint. Registers for each data endpoint determine the DMA channels used (if any), the endpoint's direction (IN or OUT, relative to the host), and its type (isochronous, bulk, or interrupt). These registers are also used to set up other information used in the USB configuration process.

For control, interrupt, and bulk data transfers, USB guarantees correct data delivery with automatic retry. Microcontroller hardware performs this task transparently to the software except for data endpoints that have been configured to use DMA. When DMA is being used, the device software is involved in error detection and recovery.

For isochronous data transfers, the USB specification calls for only a good-faith attempt at delivery. Isochronous transfers call for real-time delivery of each packet, so damaged packets cannot be retransmitted.

Special status and interrupt bits are provided for the control endpoint to indicate whether the packet currently in that endpoint's FIFO is a command that must be handled by device software.

18.5.4 Polled I/O

In Polled I/O mode, no DMA channel is specified, and interrupts are disabled. The device software must actively poll the USB status register to determine when it owns the endpoint's FIFO, and then it must write or read the endpoint's Data Port register to fill or empty the FIFO.

An endpoint operates in this mode only when the maximum packet size has been programmed to be less than or equal to the size of the FIFO. In this mode, the FIFO cannot operate in a circular fashion as it does for DMA transfers (see page 18-19).

For a receive endpoint (OUT direction relative to the host), the USB peripheral controller sets the endpoint's ACT_REQ bit in the status register whenever the FIFO is full of valid data, or when an end-of-packet event has occurred. If this bit is set, the software can empty the FIFO the next time it polls this endpoint. The amount of valid data in the FIFO is indicated by the endpoint's Received Packet Size and Buffer Status registers. When software has finished reading the FIFO, it must clear the ACT_REQ bit to release FIFO ownership to the USB peripheral controller.

For a transmit endpoint (IN direction), the ACT_REQ bit is set if the FIFO is ready to be filled with data. If this bit is set, software can fill the FIFO when it has data for that endpoint to transmit, then it must clear the ACT_REQ bit to release the FIFO.

If an error occurs on a packet received by a bulk, control, or interrupt endpoint, the ACT_REQ bit is not set. Instead, the FIFO is flushed, and the host retransmits the packet. If an error occurs when the endpoint being addressed is isochronous, no retransmission can occur; the data that was sent or received must be used as is.

18.5.5 Interrupt-Driven I/O

A single interrupt channel can be configured to alert software that the USB peripheral controller requires attention. Interrupt mask fields allow the device software to enable the events it is interested in, and the status registers show which events have occurred.

The interrupt mode of operation is very similar to the polled mode. It is an extension of the polled mode in which the ACT_REQ bit is enabled to cause an interrupt. The device software's interrupt handler then polls the status bits to see which endpoint needs service. Errors that occur in this mode are handled the same as in polled mode.

18.5.6 Using USB with DMA

Compared with polled or interrupt I/O, using DMA with USB gives the following benefits:

- **Improved Throughput:** This is an important consideration, not only from the microcontroller's perspective, but also from the USB host's perspective. If the microcontroller is ready to receive or transmit data whenever the host wishes, it reduces USB bus overhead due to retries.
- **Larger Packets:** When the USB peripheral controller is used with DMA, there is no restriction on packet size, other than that mandated by the USB specification (1023 bytes/packet for isochronous, 64 bytes/packet for bulk). When DMA is not used, packets are restricted to the size of the endpoints' FIFO.
- **Automatic Rate Control:** The microcontroller's Automatic Rate Control feature is only available when using DMA. This feature allows the amount of data sent in an isochronous IN packet to be controlled by the number of PCM highway frames or other external events that occur in each USB frame.

However, using DMA with USB is more complicated than using polled or interrupt I/O. In Polled or Interrupt mode, the USB hardware performs all error handling itself. The host is notified only when a packet has been received or transmitted without errors. With DMA, software is responsible for recovering from errors. This includes backing up DMA pointers, taking into account the amount of data that has not yet been transferred to or from the endpoint's FIFO, and so on. In addition, using DMA requires extra programming effort even before exception handling is considered.

18.5.6.1 DMA Availability

DMA mode is only available for Endpoints A–D. In DMA mode, endpoints are programmed to use the microcontroller's general-purpose DMA or SmartDMA channels.

When used with a USB data endpoint, the general-purpose DMA channels allow the device software to set up a single USB packet or an entire I/O request packet (IRP) to transfer data automatically between memory and the endpoint's FIFO. During the transfer, software interaction is required only to handle FIFO and USB packet errors.

SmartDMA channel pairs 2 and 3 can be used with specific endpoints if they are configured in the correct direction, as shown in Table 18-4. SmartDMA channels allow device software to set up single or multiple USB packets, or single or multiple IRPs, to be moved

automatically between the endpoint's FIFO and memory (or I/O), possibly using even less overhead than general-purpose DMA. Software interaction is still required to handle FIFO and USB packet errors.

Table 18-4 USB Endpoints Used with DMA

USB Endpoint	DMA Channel
USB data Endpoint (A–D) configured in either direction	Any General-Purpose DMA Channel, 0–3
USB data Endpoint A if configured as a USB OUT (receive) endpoint[1]	SmartDMA Pair 2 Receive Channel
USB data Endpoint B if configured as a USB IN (transmit) endpoint.	SmartDMA Pair 2 Transmit Channel
USB data Endpoint C if configured as a USB OUT endpoint[1]	SmartDMA Pair 3 Receive Channel
USB data Endpoint D if configured as a USB IN endpoint[1]	SmartDMA Pair 3 Transmit Channel

Notes:
1. SmartDMA Channels 2 and 3 Transmit and Receive cannot be assigned to different peripherals. For example, if SmartDMA Channel 2 Receive is assigned to USB data endpoint A, then SmartDMA Channel 2 Transmit can be used for USB data endpoint B, but cannot be used with the HDLC controller on the Am186CC microcontroller.

Selection of what type (if any) of DMA to use for a particular type of USB data pipe should take into account several issues:

- SmartDMA channels may be better for some tasks. For example, SmartDMA channels can transition from one FIFO to the next without incurring any interrupt overhead or latency.
- General-purpose DMA is simpler to program and understand for many tasks.
- Each SmartDMA channel is only capable of operation in a single direction. Because each endpoint is associated with a particular SmartDMA channel, a given system can have a maximum of two endpoints for any direction (IN or OUT).

When selecting DMA channels to use, be sure to consider other microcontroller functions (such as HDLC or UARTS) that might be using DMA. See "DMA Trade-Offs" on page 18-6.

18.5.6.2 DMA/FIFO Interaction

Unlike the polled I/O or interrupt methods, in DMA mode the maximum packet size can be programmed to a value greater than the physical size of the FIFO. Because of this, the protocol for filling or emptying a FIFO is different than when using polled I/O or interrupts. The FIFO in the endpoint operates in a circular fashion while in DMA mode.

For a receive (OUT) endpoint, the USB peripheral controller issues a DMA request whenever the FIFO is not empty. It continues to assert the DMA request until the FIFO is empty. The USB peripheral controller detects that a receive transaction has completed; either successfully or unsuccessfully. If a SmartDMA channel is configured to store packet status to the FIFO descriptor, it handles packet errors automatically and places the error status in the last three bytes sent. If general-purpose DMA is used, or if a SmartDMA channel is configured to not store packet status in the FIFO descriptor, bits in the endpoint's status register can cause an interrupt when an error occurs.

For a transmit (IN) endpoint, a DMA request is asserted whenever the FIFO is not full. The request assertion continues until the FIFO is full. Data bytes can be marked either as the last byte of the transfer or as a null byte. If the endpoint is configured to use a general-purpose DMA channel, it can indicate the last byte upon reaching the terminal count. If the

endpoint is configured to use a Smart DMA channel, information in the FIFO descriptor indicates if a byte is the last byte or a null byte.

18.5.6.3 Setting Up DMA for USB

The USB peripheral controller gives the programmer a large degree of freedom in using DMA with USB endpoints. In general, most methods of using DMA with USB fall into one of three categories:

- **Undelimited Transfers** are generally used for isochronous data that has no natural boundaries, such as audio data. For these types of transfers, either the SmartDMA channel or the general-purpose DMA serve equally well to transfer data into a circular FIFO. In addition, SmartDMA control can transfer data to or from another peripheral, such as the HDLC controller on the Am186CC microcontroller.

  For undelimited IN (to the host) transfers, the amount of data transferred in each packet is the endpoint's maximum packet size unless Auto Rate control is enabled on the endpoint. If Auto Rate control is enabled, the packet size can equal the number of samples received during the previous frame multiplied by a programmable byte/sample factor, if this value is less than the programmed maximum packet size.

- **Buffer-Per-Packet** transfers can be used for either bulk transactions or nonstreaming isochronous transfers. The amount of data transferred for each packet is determined by the FIFO size. Buffer per packet transfers are required if SmartDMA is used with packet status stored in the FIFO descriptor (MODE = 101b in the xEPDEF3 register).

- **Buffer-Per-I/O Request Packet (IRP)** transfers are similar to buffer-per-packet transfers, except that a DMA FIFO contains multiple packets. In general, Buffer-per-IRP is simpler to program for the normal case, but error handling is more complicated because DMA must be restarted in the middle of the FIFO. Buffer per IRP transfers are highly recommended for IN endpoints using a SmartDMA channel.

18.5.6.4 Short Packets

Short packets typically delineate the end of a USB I/O request packet (IRP). For example, if the maximum packet size is 64 bytes, and a FIFO that is 260 bytes is to be transferred, four full-length packets are transferred followed by a packet that contains only four bytes.

This delineation is very useful because it provides an "out-of-band" indication of where one information FIFO ends and the next one starts. In fact, it is so useful that USB specifically allows for zero-length packets, to ensure that this delineation can be performed even when the FIFO size is a multiple of the maximum packet size.

The SmartDMA channel is fully capable of sending and receiving zero-length packets. (On receive, it simply stores the byte count provided by the USB peripheral controller, which is zero, and on transmit, a special signal from the SmartDMA controller indicates that the packet has no data.)

With general-purpose DMA, receiving a zero-length packet is exactly the same as receiving any other short packet—the USB should be set to stop on receipt of a short packet, and the software examines the received length. Sending a zero-length packet is performed differently because no data is transferred through the DMA controller. To send a short packet of one byte or greater, simply program the DMA controller to send the desired count, after programming the USB peripheral controller's DMA mode to 011. In this mode, when the DMA controller sends the last byte, the packet is sent, even if the USB FIFO is not full.

To send a zero-byte packet in general-purpose DMA mode, you must clear the NOT_ZERO bit in the xEPCTL register, and then write one byte (of any value) to the endpoint's FIFO.

18.5.6.5 Error Recovery on Bulk and Interrupt Endpoints

When an endpoint is configured as a bulk or interrupt endpoint, data delivered over the endpoint is guaranteed to be correct, but is not guaranteed to be delivered within any certain time interval. When DMA is used, device software must intervene when exceptions occur to guarantee correct data. The following error recovery situations must be considered:

- SmartDMA channel receive does not require any low-level software intervention if the USB DMA mode is set to 110 (store status in SmartDMA channel FIFO). Instead, the higher level software that pulls packets out of the SmartDMA channel FIFO descriptor ring must examine each FIFO's status to determine if it was received correctly or not, and discard FIFOs with errors.
- If SmartDMA channel receive is used when the USB DMA mode is set to 100 (don't store status), then all status processing must be performed by low level software. This mode is typically used for bulk data if it is desired to receive an entire USB IRP as a single FIFO. Because the USB hardware is not storing any status in the SmartDMA channel descriptors, it is up to software to program the USB endpoint to interrupt whenever an error or a short packet is received. The interrupt handler should then disable the SmartDMA channel, update the SmartDMA channel descriptor (because some good data may have been stored before the bad data—the descriptor is updated to point to where the bad data was received so that it can be overwritten) and the SmartDMA channel descriptor pointer, re-enable the SmartDMA channel, and then restart the USB endpoint by clearing the ACT_REQ and interrupt bits. This is a significant amount of overhead, but the interrupt routine is only executed when an error or an end of packet occurs, and the higher level software never needs to worry about retrieving bad data from the FIFO, because the interrupt routine can make sure that all stored data is good before status is stored.
- Like SmartDMA channel receive, general-purpose DMA receive can either be performed per packet or per IRP, depending on whether the USB is programmed to stop on all packets or just on short packets. If an error occurs, it is up to the software to back the DMA pointer up to the start of the current packet (using information about where it is and how much data is still left in the FIFO), and clear the error and the ACT_REQ bit.
- SmartDMA channel transmit requires the USB to be set to stop on errors. If an error occurs during transmit, the interrupt handler must disable the SmartDMA channel, program its FIFO pointer to point back to the failing location, restart the DMA, and then flush the FIFO and restart the endpoint by clearing the ACT_REQ, NOT_FLUSH, and error bits. If the SmartDMA channel is being used in buffer-per-packet mode, then only the SmartDMA channel's FIFO descriptor pointer needs to be updated, but if the SmartDMA channel is being used in a buffer-per-IRP mode, then the SmartDMA channel's memory pointer must be read, and the actual FIFO descriptor in memory must be reprogrammed (starting address and length) so that the DMA can be restarted in the middle of the FIFO. When reprogramming the starting address, the number of bytes that were transferred from memory to the FIFO before the error occurred must be taken into account.
- General-purpose DMA transmit requires that the DMA mode be set to 011, to stop and interrupt on DMA terminal count. As with the SmartDMA channel, just because a DMA FIFO has been sent to the host doesn't mean that it has been successfully delivered. Interrupts should be enabled for FIFO errors and other errors, and if an error occurs, the current packet must be resent. If multiple packets are in the FIFO (buffer per IRP), then the start of the current packet in the FIFO must be calculated by taking into account the current DMA pointer and the number of bytes that are currently stored in the endpoint's FIFO. The endpoint's FIFO should be flushed before DMA is restarted.

18.5.6.6 Error Recovery on Isochronous Endpoints

Isochronous data, by definition, is very time-sensitive. Neither PCM highway nor USB have any mechanism or concept of retransmission of isochronous data. Nevertheless, there are error-recovery issues with isochronous data. For the intended audio applications, these primarily revolve around making sure that FIFO pointers do not overlap or drift too far apart.

CC

For example, assume the Am186CC microcontroller's HDLC controller is storing audio data in a circular FIFO, and the USB peripheral controller is pulling audio data out of the FIFO. Because USB operation happens in (nominally) 8 sample bursts, and PCM highway operation happens one sample at a time, if the FIFO pointers ever overlapped, old data could be transmitted intermixed with the new data, and the audio would be garbled. Likewise, if the pointers get too far apart, excessive delay is introduced in the audio.

During normal operation, the pointers should stay a relatively constant distance apart. However, it is possible to miss a frame's worth of data on the USB, because isochronous transfers are not guaranteed. When this occurs, the best that software can do is to adjust the DMA pointers to keep the error localized as a single glitch in the audio, rather than let it accumulate and cause excessive delay, or cause garbled audio (by the pointers repeatedly crossing each other). It is probably also a good idea for a missed OUT transaction for USB (PCM highway pipes) to inject silence into the FIFO for the duration of the missed transaction, to minimize the annoyance of the audio glitch.

Adjusting the pointers is very straightforward on a general-purpose DMA circular FIFO (e.g., stop the DMA, add a constant to the pointer, and restart the DMA), but is more complicated on the SmartDMA channel. If a SmartDMA channel is being used for isochronous data, the simplest thing to do is to set it up so that there are two descriptors in the ring. Each descriptor points to a portion of the circular FIFO. When a pointer needs to be adjusted, the DMA is stopped, the current location (low order 16 bits) of the memory pointer is read from the DMA hardware, a new value is calculated by adding or subtracting the adjustment from the memory pointer, and the FIFO descriptors are updated so that the next one executed covers the portion of the FIFO from the new memory pointer to the end of the FIFO, and the other descriptor covers the portion of the FIFO from the start of the FIFO to the new pointer. Then the DMA is restarted.

Note that, because such an adjustment could make one FIFO very small (e.g. one byte), it is important to use the feature that allows DMA OWN bits to be reset. Otherwise, DMA effectively stops and requires software intervention each time through the FIFOs, and there is a latency requirement to service both the descriptors within a very short time period.

18.5.7 Isochronous Transfer Synchronization

The isochronous transfer type is required by audio, telephony, or other applications that need real-time streaming delivery to avoid distortion. The USB configuration process ensures that the data pipe from the host to an isochronous endpoint has enough bandwidth to transfer the endpoint's maximum packet size in every frame, but the design must also synchronize the data so it is delivered at the correct rate.

Isochronous synchronization involves converting the data stream from its sample rate (for example, the 44.1-KHz rate of an audio CD player) into packets delivered at the fixed USB start-of-frame (SOF) rate of 1 KHz (1000 frames per second). The USB specification defines three types of isochronous synchronization:

- **Asynchronous**: The data sample clock and the USB frame rate are independent of each other. It is up to the host's device driver and device software to convert the data rate as needed. For example, a receiving endpoint's software (host or device) can provide feedback so the transmitting endpoint's software can adjust the amount of isochronous data sent in each frame.
- **Synchronous**: The data sample rate is synchronized with the USB SOF rate so the same amount of isochronous data can be transmitted in every frame. There are two ways to achieve this:
  - Lock the data source sample clock to the USB SOF rate. For example, a design can route the microcontroller's USBSOF output through a PLL to drive the sample clock of an external codec. For more about the USBSOF signal, see "Isochronous Synchronization Signals" on page 18-6.
  - Request USB master client capability (through the USB driver basic host interface) and then adjust the USB SOF rate to keep it synchronized with the sample clock. Only one device can be the master client at a time, so devices that use this method must be able to operate asynchronously if master client capability is denied.
- **Adaptive**: The data sample clock can be freely adjusted to receive or transmit data at any rate within a given range. The microcontroller's Auto Rate feature (described in the following section) allows isochronous IN endpoints to implement adaptive synchronization with a variety of input sources.

The type of synchronization to use for an isochronous endpoint depends on the design requirements and capabilities of the peripheral device. All of these synchronization types make use of USB peripheral controller features described in the following section.

18.5.8 Isochronous Transfer Features

The USB peripheral controller provides full support for the Isochronous transfer type while minimizing system resource overhead. A USB peripheral device using the Am186CC or Am186CU microcontroller can easily support the isochronous data transfer in the IN direction as an asynchronous, synchronous, or adaptive synchronous data source. These features combined with the other integrated communications devices and DMA controller allow many different communications and audio devices to be built with this device. The following microcontroller features are provided to support isochronous transfers:

- **Missing-SOF Detection**: The USB peripheral controller implements an adaptive missing-SOF detection mechanism. A missing SOF packet is detected when the current USB frame length is six USB bit times greater than the last frame in which a SOF packet was successfully received.

  Hardware indicates a missing SOF by setting the UISTAT2 register's MS_SOF bit, which software can enable as an interrupt source by setting the corresponding bit in the UIMASK2 register.
- **SOF Generation**: Whenever an SOF is detected, hardware sets the UISTAT2 register's SOF_GEN bit, which software can enable as an interrupt source by setting the corresponding bit in the UIMASK2 register.

The SOF is also reflected on the controller's USBSOF output signal, which is used in the first method (lock the sample clock) of synchronous isochronous synchronization, as described in "Isochronous Transfer Synchronization" on page 18-23.

If a missing SOF is detected, the USB peripheral controller automatically generates an internal SOF, which is reflected by the SOF_GEN bit and the USBSOF signal. This allows synchronous isochronous endpoints to remain locked to the USB clock even when the SOF packet is corrupted on the bus.

- **USB Frame Position Monitoring**: This allows the device software to detect any difference between the sample rate of a data source and the USB frame rate. This is required for an Isochronous IN endpoint that uses the second method (request USB master client capability) for synchronous isochronous synchronization, as described in "Isochronous Transfer Synchronization" on page 18-23.

CC

In the Am186CC microcontroller, the SAM_CLK_SEL field in the ISCTL register can select a sample rate clock source: either the USBSCI signal (on the UCLK pin) or the frame synchronization signal used for HDLC Channel A, PCM Highway, and GCI.

During each USB frame, the FPMCNT register latches the USB frame position bit counter after a specific number of source clocks are counted on the sample input. The value latched in the FPMCNT register is the number of USB bit times counted during the source clock interval specified in the BCNT_LRATE field of the ISCTL register (1–64 source clocks, programmable in powers of two). Device software can compare these two values to determine whether the USB frame rate and the source sample clock are moving relative to each other.

If the device is granted master client capability, it is able to use the USB Device basic host interface (defined in the USB specification) to gradually increase or decrease the USB SOF rate to correct any drift with respect to the data source's sample rate.

Whenever FPMCNT is updated, hardware sets the UISTAT2 register's POS_UP bit, which software can enable as an interrupt source by setting the corresponding bit in the UIMASK2 register.

- **Auto Rate:** This allows the designer to implement adaptive synchronization on an isochronous IN endpoint using general-purpose DMA or SmartDMA to handle an arbitrary data source rate. The Auto Rate feature uses the data source's sample rate clock (frame rate) as an input to automatically control the number of data bytes sent to the USB host during each transaction.

CC

In the Am186CC microcontroller, the SAM_CLK_SEL field in the ISCTL register can select a sample rate clock source: either the USBSCI signal (on the UCLK pin) or the frame synchronization clock (FSC) signal used for HDLC Channel A, PCM Highway, and GCI.

The BYTES_SAM field in ISCTL sets the number of bytes to move per source clock sample (1, 2, or 4 bytes). Also make sure that the Max Packet Size programmed for the endpoint is greater than or equal to the largest number of data bytes that the endpoint might need to move during a USB transaction.

After the sample clock source and bytes per sample are selected, set the AUTO_RATE_EN bit in the xEPDEF3 register (where x = A, B, C, or D) to enable auto rate for the endpoint.

The specified number of bytes is transferred on each sample clock as long as data is present in the endpoint's FIFO, or is sequentially written to the FIFO as needed during the transaction.

- **Start of Frame and Frame Number Monitoring**: The USB peripheral controller monitors the USB SOF packet and latches the frame number value into the Time Stamp (TSTMP) register upon successfully receiving the SOF packet from the USB host.

  Software can arm the Time Stamp Match (TSTMPM) register by writing a specific USB frame number to it. Then, when the USB peripheral controller receives an SOF packet with a number greater or equal to the written value, hardware sets the UISTAT2 register's TSTMP_M bit, which software can enable as an interrupt by setting the corresponding bit in UIMASK2. The interrupt does not occur again until TSTMP_M is cleared in UISTAT2 and TSTMPM is written again.

  This mechanism allows software to start a certain data pattern during a specific USB frame, if required. This feature can be used for asynchronous USB data sources using implicit data pattern generation.

18.5.9 Command Handling

The primary function of the device's control endpoint is to accept and respond to commands issued to it by the USB host. All of the USB standard, device class, and vendor specific commands are issued to the control endpoint known as the device endpoint 0. The USB peripheral controller hardware handles some of these commands without requiring that the device software decode and specifically "handle" the command. Other commands are received from the USB host and passed on to the device software for processing. These commands and how they are handled are outlined in the following sections.

18.5.9.1 Commands Handled by Device Software

Table 18-5 on page 18-27 describes the commands that must be handled by the device software.

When any command is received by the USB peripheral controller, hardware sets the NEW_COMMAND bit in the CNTCTL register. If the device software must take some action, the ACT_REQ bit is also set in the affected endpoint's xEPCTL register.

The NEW_COMMAND bit and all of the ACT_REQ bits have mirror bits in the UISTAT1 register. (A mirror bit is set whenever the corresponding status bit is set.) Each mirror bit can be enabled as an interrupt source by setting the corresponding bit in the UIMASK1 register.

The software is then required to decode the command data and either:

- Accept subsequent data associated with the command (for OUT commands). When it is finished handling an OUT command, software must clear the COMMAND_BUSY bit in the CNTCTL register to indicate that it is ready to process more commands.
- Return the appropriate data requested in the command (for IN commands). The USB peripheral controller hardware automatically clears the COMMAND_BUSY bit in the CNTCTL register when it finishes transmitting the requested data.

All of the low-level USB protocol processing is handled entirely in hardware (that is, all handshake packets are accepted from or returned to the USB host automatically).

Table 18-5 USB Commands Handled by Device Software

Command	Parameters and Data Passed	Data Direction	Results
GET_DESCRIPTOR	Device, Configuration, or String Descriptor	IN	The device software, upon detecting this command, should return all of the data associated with the particular descriptor that was requested. Because this controller allows the endpoint parameters to be programmed at any time, an unlimited number of descriptors can be supported.
SET_CONFIGURATION	Device Configuration	OUT	The device software, upon detecting this command, should configure all of the endpoints with the applicable USB parameters based on the descriptor information that was passed to the host during the GET_DESCRIPTOR command.
SET_INTERFACE	Interface Alternate Setting	OUT	The device software, upon detecting this command, should configure the endpoints associated with the specified interface, for the requested alternate setting, based on the descriptor information that was passed to the host during the GET_DESCRIPTOR command.
SET_DESCRIPTOR	Device, Configuration, String, Interface, or Endpoint Descriptor	OUT	The device software, upon detecting this command, should accept a new descriptor from the USB host.
SYNC_FRAME	Synchronization Frame	IN	The device software, upon detecting this command, should return the frame number in which an isochronous, IN endpoint begins its data pattern.
Device Class or Vendor Specific	Various	IN and OUT	The device software should service the commands that it has been programmed to handle. If the device software does not recognize a particular command it should clear the EP_NOT_STALLED bit in the CNTCTL register to direct the controller hardware to return the stalled handshake in the data stage. (The NEW_COMMAND bit must be cleared at the same time, or clearing the EP_NOT_STALLED bit has no effect.)

18.5.9.2 Commands Handled by the USB Peripheral Controller Hardware

Table 18-6 on page 18-28 describes the commands that do not require device software handling. When these commands are detected by the USB peripheral controller, they are handled entirely in hardware. The device software does have the ability to detect the reception of any USB setup packet sent to it by the USB host, but it cannot monitor the specific setup packet type when the command is handled solely by the controller hardware.

Table 18-6 USB Commands Handled by USB Peripheral Controller Hardware

Command	Parameters and Data Passed	Data Direction	Results
SET_ADDRESS	Device's USB address	OUT	Device stores the address assigned to it by the USB host.
SET_FEATURE	Device Remote Wake-up, or Endpoint Stall	OUT	The device's remote wake up feature is enabled (or) A particular endpoint is forced to be stalled. If the specified endpoint is not configured, the device returns the stalled handshake to the host during the status stage.
CLEAR_FEATURE	Device Remote Wake-up, or Endpoint Stall	OUT	The device's remote wake-up feature is disabled (or) A particular endpoint is forced to be un-stalled. If the specified endpoint is not configured, the device returns the stalled handshake to the host during the status stage.
GET_STATUS	Device Self Powered, Device Remote Wake-up, or Endpoint Stall Status	IN	The device's programmable Self Powered bit is returned to the USB host as a value of 1 or 0, indicating that the device is self-powered or bus-powered. The Remote Wake up bit is returned with a value that depends on whether the remote wake up feature was last set or cleared. A particular endpoint's stall status is returned to the USB host. If the specified endpoint is not configured, the device returns the stalled handshake to the host during the status stage.
GET_CONFIGURATION	Device's current configuration	IN	The current device configuration number is returned to the USB host.
GET_INTERFACE	Interface's current selected alternate setting	IN	The currently selected alternate setting for the interface number that is specified in this command is returned to the USB host. If the specified interface is not present, the device returns the stalled handshake to the host during the status stage.

18.5.10 Command Protocol

As a slave device, the Am186CC or Am186CU microcontroller must always be prepared to let the USB master send a new SETUP packet. In other words, software could be processing a command, and the host could send a new command without warning. The software should stop working on the old command and deal with the new one immediately. In practice, it is impossible to arbitrarily stop a program at any point like this. For example, the software could have been writing out a response to the previous command, and stopping it at a precise moment is very difficult.

The NEW_COMMAND interlock bit allows the hardware to ignore the software during time periods when the software is still in the middle of processing a previous command.

During processing of a command, the ACT_REQ bit is "politely" bounced back and forth between hardware and software. For setup and control write packets (data from the host), hardware sets the ACT_REQ bit to indicate that the host has filled the FIFO. software then clears the ACT_REQ bit to indicate that it has drained the FIFO. For control read packets,

software clears the ACT_REQ bit when it has filled the FIFO with information to go to the host, and hardware sets the ACT_REQ bit after the information has safely made it to the host.

The host can send a new command at any time, so the NEW_COMMAND bit provides a somewhat less “polite” method for the hardware to inform the software of who “owns” the FIFO. When a SETUP packet is detected (before it is written to the FIFO), the hardware clears the ACT_REQ bit, and sets the NEW_COMMAND bit, to show that the hardware “stole” ownership from the software.

Because the software could be busy trying to update the FIFO and/or the ACT_REQ, EP_NOT_STALLED, or COMMAND_BUSY bits, the host locks out accesses to the FIFO and these bits whenever NEW_COMMAND is set. Attempts by software to read or write the FIFO, or to alter these bits, fail silently.

When a command is received that software must handle, it is stored in the FIFO and then the ACT_REQ bit is set to indicate that the FIFO contains valid data.

18.5.10.1 Data Transfer Using the Control Endpoint

The control endpoint can transfer data, but there are several potential problems.

At the end of control read data transfers, it is impossible for the software to know whether the host accepted the most recent data sent, or whether it sent the status stage before accepting the data. For USB commands, it is not an error for the host to terminate a command early. For example, it can ask to read descriptors, and enter the status phase before it has finished reading all the descriptors. This is problematic for data transfers, and the only real way around it is for the host to transmit information in the SETUP packet that describes where in the data stream it wishes to start reading.

At the end of control write data transfers, it is impossible for the software to know whether the device successfully completed the status phase, or whether a new setup packet aborted the status phase. As with the control read problem, this problem can be alleviated by the host sending information in the command packet about where in the data stream this write should start.

18.5.10.2 Control Endpoint Interrupts

The ACT_REQ bit and the NEW_COMMAND bit are reflected in the UISTAT1 register as the CNT_EP_ACT and CNT_EP_NEW bits. Software can mask off these interrupts in the UIMASK1 register. Most applications use only the CNT_EP_ACT interrupt, but some applications may find it advantageous to use the CNT_EP_NEW interrupt. This interrupt is useful if the system spawns a new task to deal with data transactions. In this case, the software could use a CNT_EP_NEW interrupt to spawn the task dealing with the aborted command.

18.5.11 Interrupt Endpoint Programming

The microcontroller's USB interface contains one interrupt endpoint. The purpose of an interrupt endpoint is to allow small amounts of data to be transferred from the device to the host. According to section 4.7.3 of the USB Specification, “A small, spontaneous data transfer from a device is referred to as interrupt data. Such data can be presented for transfer by a device at any time, and is delivered by the USB at a rate no slower than as is specified by the device. Interrupt data typically consists of event notification, characters, or coordinates that are organized as one or more bytes. An example of interrupt data is the coordinates from a pointing device. Although an explicit timing rate is not required, interactive data may have response time bounds that the USB must support.”

18.5.11.1 USB Command Processing and the Interrupt Endpoint

When a SET_CONFIGURATION or SET_INTERFACE command is received, software must reprogram the Interrupt Endpoint Definition registers (if necessary) to reflect the new configuration and alternate interface setting. Also, the descriptor relating to the interrupt endpoint (which is returned to a host GET_DESCRIPTOR request) must contain the correct maximum packet size (8 or 16 bytes) and Interval value (1-ms to 255-ms interrupt rate).

18.5.11.2 Data Transfer with the Interrupt Endpoint

The interrupt endpoint can be used in two different ways. If the amount of data to be transferred on each interrupt is less than or equal to the maximum packet size, each packet sent to the host constitutes an entire transaction. If the amount of data to be transferred on each interrupt exceeds the maximum packet size, an interrupt can consist of multiple packets. In this case, each packet except the last one must be MaxPacketSize bytes. Transferring a number of bytes between 0 and MaxPacketSize – 1 (inclusive) denotes the end of the transaction. If a large number of bytes are to be transferred on each interrupt, it is strongly suggested that the maximum packet size be set to 16, because this makes more efficient use of the USB bandwidth than a setting of 8.

18.5.11.3 Interrupt Endpoint Interrupts

The ACT_REQ bit is reflected in the UISTAT1 register as the INT_EP_ACT bit. Software can mask off this interrupt in the UIMASK1 register.

18.5.12 Endpoint Definitions

The USB specification provides for endpoints to be grouped into interfaces. Multiple interfaces that do not share endpoints can be grouped into configurations, and a device can have multiple configurations, only one of which can be in use at any one time. In the Am186CC and Am186CU microcontrollers, software assigns each of the endpoints (other than the control endpoint, which has a number of 0 and appears in every interface) an endpoint number, interface number, configuration number, alternate setting number, Max Packet Size, and direction.

The *USB Specification, Version 1.0* defines the endpoint configuration process:

“Host software should only set configuration and interface values that match a device descriptor returned by the device in response to a GET_DESCRIPTOR command. However, the USB hardware accepts as valid any configuration or feature setting in the range of 0d to 3d, regardless of the available descriptors. To help ensure reliable operation in any USB environment, device software can define a minimal descriptor (i.e., Endpoint 0 with no bandwidth allocation) for any configuration and interface settings that it does not define otherwise.”

18.5.12.1 Control Endpoint Definition

The control endpoint features are not programmable as are the other endpoints. This endpoint is common to, and is required by, all USB device class specifications. Table 18-7 lists the control endpoint parameters.

The control endpoint (endpoint 0) is always considered to be a member of all device configurations, a member of all interfaces present in a device configuration, and a member of all alternate settings of any given interface.

Table 18-7 Control Endpoint Definition

Parameter	Value
USB Parameters	
Number	0
Configuration	All
Interface	All
Alternate Setting	All
Type	Control
Maximum Packet Size	Eight bytes
System Parameters	
Data Handling	Polled I/O or interrupt driven
FIFO Depth	Eight bytes

18.5.12.2 Interrupt Endpoint Definition

The Am186CC and Am186CU microcontrollers each provide one dedicated interrupt endpoint. Typically, all of the USB Device Class specifications require that a USB device contain at least one interrupt endpoint. The USB host uses this endpoint, in conjunction with the device's control endpoint, to process class-specific transactions and to generally service specific device requests when required.

The interrupt endpoint features are highly programmable. Device software can modify these features at any time in response to the various commands issued to the device's control endpoint. The USB host can make these requests during the device enumeration process or at any other time during the device operation. Table 18-8 lists the interrupt endpoint features.

Table 18-8 Interrupt Endpoint Definition

Parameter	Value
USB Parameters	
Number	1–15
Configuration	0–3
Interface	0–3
Direction	IN (to host only)
Alternate Setting	0–7
Type	Interrupt
Max Packet Size	8 or 16 bytes
System Parameters	
Data Handling	Polled I/O or interrupt driven
FIFO Depth	16 bytes

18.5.12.3 Data Endpoint Definition

The Am186CC and Am186CU microcontrollers each provide four general-purpose data endpoints. These endpoints transfer large amounts of data between the USB host and device using either the USB bulk, isochronous, or interrupt transfer protocols. Note that if the data endpoint is programmed for interrupt transfer, the DMA mode is not applicable. Typically, all of the USB Device Class specifications require that a USB device contain any number and type of data endpoints to transfer the various data types required by the device class.

The data endpoint features are highly programmable. Device software can modify these features at any time in response to the various commands issued to the device's control endpoint. Table 18-9 lists the data endpoint features.

Table 18-9 Data Endpoints A–D Definition

Parameter	**Values**				
USB Parameters					
Number	1–15				
Configuration	0–3				
Interface	0–3				
Alternate Setting	0–7				
Direction (Endpoints A and C)	IN or OUT			OUT	
Direction (Endpoints B and D)				IN	
Type	Interrupt, Bulk, or Isochronous	Bulk	Isochronous	Bulk	Isochronous
Max. Packet Size (Endpoints A and B)	1–16 bytes (8 or 16 for bulk)	8, 16, 32, or 64 bytes	1–1023 bytes[1]	8, 16, 32, or 64 bytes	1–1023 bytes[1]
Max. Packet Size (Endpoints C and D)	1–64 bytes (8 16, 32, or 64 for bulk)	8, 16, 32, or 64 bytes	1–1023 bytes	8, 16, 32, or 64 bytes	1–1023 bytes
System Parameters					
Data Handling	Polled I/O or Interrupt Driven	General-Purpose DMA		SmartDMA Channel	
FIFO Depth (Endpoints A and B)	8 or 16 bytes				
FIFO Depth (Endpoints C and D)	8, 16, 32, or 64 bytes				

Notes:
1. A 24-MHz processor clock is not fast enough for software to keep up with 1023-byte isochronous IN or OUT packets using only an 8, 16, or 32-byte FIFO. If a 24-MHz processor clock is used and a Max Packet Size of 1023 bytes is required for isochronous data, use endpoint C or D and set the FIFO size to 64 bytes.

18.5.13 Software-Related Considerations

- A data endpoint must be configured with the xEPDEFx register before enabling it with the EP_EN bit in the xEPCTL register.
- When the MODE bit field in the xEPDEF3 register is set to 101b (SmartDMA channel, status stored in the buffer descriptor), a bulk OUT transfer that results in a retransmission of data by the host due to handshake packet errors produces the following buffer descriptor field values: STP = 1, ENP = 1, and CRC = 1. The MCNT value in the buffer descriptor is invalid because setting the CRC bit causes the ERR bit to be set as well.

  Also, when the MODE bit field is set to 101b, a bulk or isochronous OUT transfer with a message size that is an integer multiple of the maximum packet size results in the following buffer descriptor field values: STP = 1, ENP = 1, and MCNT = 0.

18.6 INITIALIZATION

On both an external and internal reset, the following occurs:

- All USB interrupts are cleared and masked.
- The USB peripheral controller reports that it is self-powered (S_POWER bit of the USBMFR register is set).
- The interrupt endpoint number is set to 1.
- The interrupt endpoint FIFO defaults to 16 bytes deep.
- The interrupt endpoint maximum packet size is set to 16d.
- The A, B, C, and D endpoints default to OUT direction, bulk type, with a maximum packet size of 8 bytes.
- The A and B endpoint FIFOs default to 16 bytes deep.
- The C and D endpoint FIFOs default to 64 bytes deep.
- The Isochronous Missed Packet and Full Data Packet interrupts are unmasked.

APPENDIX

A REGISTER SUMMARY

Table A-1 on page A-2 provides a summary of all the Am186CC/CH/CU microcontrollers' peripheral control block (PCB) registers, listed in offset order. The table includes the following information for each register:

- Abbreviated name
- Register description page number
- Relative offset from the PCB base (set in RELOC)
- Default location in I/O space (equal to the default PCB base of FC00h plus the register's relative offset)
- Default value at reset
- Bit and field names and layout

An "x" in the default value column denotes a digit for which the default value is not defined. A "?" indicates that the digit's value depends on external inputs. If a digit contains both undefined and external input bits, a "?" is used.

If more than one default value is given for a register, it contains one or more bits with undefined defaults. In this case either value might be present.

If a group of registers is not supported on all the Am186CC/CH/CU microcontrollers, the group heading indicates the controllers that support that group of registers. An exclamation point (**!**) following a specific bit or register name indicates that additional controller-specific information can be found in the individual register or bit description.

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HDLC Channel A Registers CC CH																			
HACON	00h	FC00h	0000h	*Res*								HRESET	*Res*	NRZI	TRANSM	LOOPR	LOOPL	CRCTYPE	
HATCON0	02h	FC02h	0000h	*Res*				TTHRSH				*Res*				TFIFOEN	FORABR	HTEN	IMSTART	CRCDIS	LBREAD	LBNOW
HATCON1	04h	FC04h	0000h	*Res*				FLAGIDL	MLTDRP	AUTOCTS	TMSBF	TXCINV	GCIDEN	ODRV		TDELAY			
HARCON0	06h	FC06h	0000h	*Res*				RTHRSH		RCPST	RMSBF	RXCINV	RREJECT	RSTOP	HREN	MINRL			
HARCON1	08h	FC08h	0000h	MAXRL															
HASTATE	0Ah	FC0Ah	0010h 0030h	*Res*										CTSS	RTRS	ABORTS	MARKIS	FLAGS	FRAMES
HAISTAT0	0Ch	FC0Ch	0000h	*Res*			REOF	RTHRES	RDATA1	TTHRES	TDATA1	*Res*			FABRST	CTSLST	TUFLO	TGOODF	TSTOP
HAIMSK0	0Eh	FC0Eh	0000h	*Res*			REOF	RTHRES	RDATA1	TTHRES	TDATA1	*Res*			FABRST	CTSLST	TUFLO	TGOODF	TSTOP
HAISTAT1	10h	FC10h	0000h	*Res*						MAMC	SFMC	SHORT	VSHORT	RTRDES	ROFLO	ABORTE	MARKIE	FLAGE	FRAMEE
HAIMSK1	12h	FC12h	0000h	*Res*						MAMC	SFMC	SHORT	VSHORT	RTRDES	ROFLO	ABORTE	MARKIE	FLAGE	FRAMEE
HATD	14h	FC14h	00xxh	*Res*								TDATA							
HARD	16h	FC16h	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	RDATA							
HARFS1	(16h)	(FC16h)	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	FBCNT[7–0]							
HARFS2	(16h)	(FC16h)	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	FBCNT[15–8]							
HARFS3	(16h)	(FC16h)	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	FRAM	FOFLO	CRCE	MTCH	FABORT	FLONG	FSHORT	
HARDP	18h	FC18h	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	RDATA							
HASFCNT	1Ah	FC1Ah	0000h	HSFCNT															
HASFCNTP	1Ch	FC1Ch	0000h	HSFCNTP															
HAMACNT	1Eh	FC1Eh	0000h	HMACNT															
HAMACNTP	20h	FC20h	0000h	HMACNTP															
HAA0	22h	FC22h	0000h	HA															
HAA0MSK	24h	FC24h	0000h	HAMSK															
HAA1	26h	FC26h	0000h	HA															
HAA1MSK	28h	FC28h	0000h	HAMSK															
HAA2	2Ah	FC2Ah	0000h	HA															
HAA2MSK	2Ch	FC2Ch	0000h	HAMSK															
HAA3	2Eh	FC2Eh	0000h	HA															
HAA3MSK	30h	FC30h	0000h	HAMSK															
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HDLC Channel B Registers CC CH																			
HBCON	40h	FC40h	0000h	*Res*								HRESET	*Res*	NRZI	TRANSM	LOOPR	LOOPL	CRCTYPE	
HBTCON0	42h	FC42h	0000h	*Res*				TTHRSH			*Res*		TFIFOEN	FORABR	HTEN	IMSTART	CRCDIS	LBREAD	LBNOW
HBTCON1	44h	FC44h	0000h	*Res*				FLAGIDL	MLTDRP	AUTOCTS	TMSBF	TXCINV	GCIDEN	ODRV		TDELAY			
HBRCON0	46h	FC46h	0000h	*Res*				RTHRSH		RCPST	RMSBF	RXCINV	RREJECT	RSTOP	HREN	MINRL			
HBRCON1	48h	FC48h	0000h	MAXRL															
HBSTATE	4Ah	FC4Ah	0010h 0030h	*Res*										CTSS	RTRS	ABORTS	MARKIS	FLAGS	FRAMES
HBISTAT0	4Ch	FC4Ch	0000h	*Res*			REOF	RTHRES	RDATA1	TTHRES	TDATA1	*Res*			FABRST	CTSLST	TUFLO	TGOODF	TSTOP
HBIMSK0	4Eh	FC4Eh	0000h	*Res*			REOF	RTHRES	RDATA1	TTHRES	TDATA1	*Res*			FABRST	CTSLST	TUFLO	TGOODF	TSTOP
HBISTAT1	50h	FC50h	0000h	*Res*						MAMC	SFMC	SHORT	VSHORT	RTRDES	ROFLO	ABORTE	MARKIE	FLAGE	FRAMEE
HBIMSK1	52h	FC52h	0000h	*Res*						MAMC	SFMC	SHORT	VSHORT	RTRDES	ROFLO	ABORTE	MARKIE	FLAGE	FRAMEE
HBTD	54h	FC54h	00xxh	*Res*								TDATA							
HBRD	56h	FC56h	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	RDATA							
HBRFS1	(56h)	(FC56h)	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	FBCNT[7–0]							
HBRFS2	(56h)	(FC56h)	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	FBCNT[15–8]							
HBRFS3	(56h)	(FC56h)	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	FRAM	FOFLO	CRCE	MTCH	FABORT	FLONG	FSHORT	
HBRDP	58h	FC58h	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	RDATA							
HBSFCNT	5Ah	FC5Ah	0000h	HSFCNT															
HBSFCNTP	5Ch	FC5Ch	0000h	HSFCNTP															
HBMACNT	5Eh	FC5Eh	0000h	HMACNT															
HBMACNTP	60h	FC60h	0000h	HMACNTP															
HBA0	62h	FC62h	0000h	HA															
HBA0MSK	64h	FC64h	0000h	HAMSK															
HBA1	66h	FC66h	0000h	HA															
HBA1MSK	68h	FC68h	0000h	HAMSK															
HBA2	6Ah	FC6Ah	0000h	HA															
HBA2MSK	6Ch	FC6Ch	0000h	HAMSK															
HBA3	6Eh	FC6Eh	0000h	HA															
HBA3MSK	70h	FC70h	0000h	HAMSK															
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HDLC Channel C Registers CC																			
HCCON	80h	FC80h	0000h	*Res*								HRESET	*Res*	NRZI	TRANSM	LOOPR	LOOPL	CRCTYPE	
HCTCON0	82h	FC82h	0000h	*Res*				TTHRSH			*Res*		TFIFOEN	FORABR	HTEN	IMSTART	CRCDIS	LBREAD	LBNOW
HCTCON1	84h	FC84h	0000h	*Res*				FLAGIDL	MLTDRP	AUTOCTS	TMSBF	TXCINV	GCIDEN	ODRV		TDELAY			
HCRCON0	86h	FC86h	0000h	*Res*				RTHRSH		RCPST	RMSBF	RXCINV	RREJECT	RSTOP	HREN	MINRL			
HCRCON1	88h	FC88h	0000h	MAXRL															
HCSTATE	8Ah	FC8Ah	0010h 0030h	*Res*										CTSS	RTRS	ABORTS	MARKIS	FLAGS	FRAMES
HCISTAT0	8Ch	FC8Ch	0000h	*Res*			REOF	RTHRES	RDATA1	TTHRES	TDATA1	*Res*			FABRST	CTSLST	TUFLO	TGOODF	TSTOP
HCIMSK0	8Eh	FC8Eh	0000h	*Res*			REOF	RTHRES	RDATA1	TTHRES	TDATA1	*Res*			FABRST	CTSLST	TUFLO	TGOODF	TSTOP
HCISTAT1	90h	FC90h	0000h	*Res*						MAMC	SFMC	SHORT	VSHORT	RTRDES	ROFLO	ABORTE	MARKIE	FLAGE	FRAMEE
HCIMSK1	92h	FC92h	0000h	*Res*						MAMC	SFMC	SHORT	VSHORT	RTRDES	ROFLO	ABORTE	MARKIE	FLAGE	FRAMEE
HCTD	94h	FC94h	00xxh	*Res*								TDATA							
HCRD	96h	FC96h	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	RDATA							
HCRFS1	(96h)	(FC96h)	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	FBCNT[7–0]							
HCRFS2	(96h)	(FC96h)	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	FBCNT[15–8]							
HCRFS3	(96h)	(FC96h)	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	FRAM	FOFLO	CRCE	MTCH	FABORT	FLONG	FSHORT	
HCRDP	98h	FC98h	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	RDATA							
HCSFCNT	9Ah	FC9Ah	0000h	HSFCNT															
HCSFCNTP	9Ch	FC9Ch	0000h	HSFCNTP															
HCMACNT	9Eh	FC9Eh	0000h	HMACNT															
HCMACNTP	A0h	FCA0h	0000h	HMACNTP															
HCA0	A2h	FCA2h	0000h	HA															
HCA0MSK	A4h	FCA4h	0000h	HAMSK															
HCA1	A6h	FCA6h	0000h	HA															
HCA1MSK	A8h	FCA8h	0000h	HAMSK															
HCA2	AAh	FCAAh	0000h	HA															
HCA2MSK	ACh	FCACh	0000h	HAMSK															
HCA3	AEh	FCAEh	0000h	HA															
HCA3MSK	B0h	FCB0h	0000h	HAMSK															
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HDLC Channel D Registers CC																			
HDCON	C0h	FCC0h	0000h	*Res*								HRESET	*Res*	NRZI	TRANSM	LOOPR	LOOPL	CRCTYPE	
HDTCON0	C2h	FCC2h	0000h	*Res*				TTHRSH			*Res*		TFIFOEN	FORABR	HTEN	IMSTART	CRCDIS	LBREAD	LBNOW
HDTCON1	C4h	FCC4h	0000h	*Res*				FLAGIDL	MLTDRP	AUTOCTS	TMSBF	TXCINV	GCIDEN	ODRV		TDELAY			
HDRCON0	C6h	FCC6h	0000h	*Res*				RTHRSH		RCPST	RMSBF	RXCINV	RREJECT	RSTOP	HREN	MINRL			
HDRCON1	C8h	FCC8h	0000h	MAXRL															
HDSTATE	CAh	FCCAh	0010h 0030h	*Res*										CTSS	RTRS	ABORTS	MARKIS	FLAGS	FRAMES
HDISTAT0	CCh	FCCCh	0000h	*Res*			REOF	RTHRES	RDATA1	TTHRES	TDATA1	*Res*			FABRST	CTSLST	TUFLO	TGOODF	TSTOP
HDIMSK0	CEh	FCCEh	0000h	*Res*			REOF	RTHRES	RDATA1	TTHRES	TDATA1	*Res*			FABRST	CTSLST	TUFLO	TGOODF	TSTOP
HDISTAT1	D0h	FCD0h	0000h	*Res*						MAMC	SFMC	SHORT	VSHORT	RTRDES	ROFLO	ABORTE	MARKIE	FLAGE	FRAMEE
HDIMSK1	D2h	FCD2h	0000h	*Res*						MAMC	SFMC	SHORT	VSHORT	RTRDES	ROFLO	ABORTE	MARKIE	FLAGE	FRAMEE
HDTD	D4h	FCD4h	00xxh	*Res*								TDATA							
HDRD	D6h	FCD6h	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	RDATA							
HDRFS1	(D6h)	(FCD6h)	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	FBCNT[7–0]							
HDRFS2	(D6h)	(FCD6h)	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	FBCNT[15–8]							
HDRFS3	(D6h)	(FCD6h)	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	FRAM	FOFLO	CRCE	MTCH	FABORT	FLONG	FSHORT	
HDRDP	D8h	FCD8h	00xxh	STAT1A	STAT0A	STATNUM		RTHRES	RDATA1	TTHRES	TDATA1	RDATA							
HDSFCNT	DAh	FCDAh	0000h	HSFCNT															
HDSFCNTP	DCh	FCDCh	0000h	HSFCNTP															
HDMACNT	DEh	FCDEh	0000h	HMACNT															
HDMACNTP	E0h	FCE0h	0000h	HMACNTP															
HDA0	E2h	FCE2h	0000h	HA															
HDA0MSK	E4h	FCE4h	0000h	HAMSK															
HDA1	E6h	FCE6h	0000h	HA															
HDA1MSK	E8h	FCE8h	0000h	HAMSK															
HDA2	EAh	FCEAh	0000h	HA															
HDA2MSK	ECh	FCECh	0000h	HAMSK															
HDA3	EEh	FCEEh	0000h	HA															
HDA3MSK	F0h	FCF0h	0000h	HAMSK															
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
General-Purpose DMA Channel 0 Registers																			
GD0CON0	100h	FD00h	0000h	ST	AST	TC	INT	*Res*		P		*Res*	TS	*Res*	DSEL !				
GD0CON1	102h	FD02h	0000h	SM/$\overline{IO}$	SAW			SINC				DM/$\overline{IO}$	DAW			DINC			
GD0SRCL	104h	FD04h	0000h	DSA[15–0]															
GD0SRCH	106h	FD06h	0000h	*Res*												DSA[19–16]			
GD0DSTL	108h	FD08h	0000h	DDA[15–0]															
GD0DSTH	10Ah	FD0Ah	0000h	*Res*												DDA[19–16]			
GD0TC	10Ch	FD0Ch	0000h	TC															
General-Purpose DMA Channel 1 Registers																			
GD1CON0	110h	FD10h	0000h	ST	AST	TC	INT	*Res*		P		*Res*	TS	*Res*	DSEL !				
GD1CON1	112h	FD12h	0000h	SM/$\overline{IO}$	SAW			SINC				DM/$\overline{IO}$	DAW			DINC			
GD1SRCL	114h	FD14h	0000h	DSA[15–0]															
GD1SRCH	116h	FD16h	0000h	*Res*												DSA[19–16]			
GD1DSTL	118h	FD18h	0000h	DDA[15–0]															
GD1DSTH	11Ah	FD1Ah	0000h	*Res*												DDA[19–16]			
GD1TC	11Ch	FD1Ch	0000h	TC															
General-Purpose DMA Channel 2 Registers																			
GD2CON0	120h	FD20h	0000h	ST	AST	TC	INT	*Res*		P		*Res*	TS	*Res*	DSEL !				
GD2CON1	122h	FD22h	0000h	SM/$\overline{IO}$	SAW			SINC				DM/$\overline{IO}$	DAW			DINC			
GD2SRCL	124h	FD24h	0000h	DSA[15–0]															
GD2SRCH	126h	FD26h	0000h	*Res*												DSA[19–16]			
GD2DSTL	128h	FD28h	0000h	DDA[15–0]															
GD2DSTH	12Ah	FD2Ah	0000h	*Res*												DDA[19–16]			
GD2TC	12Ch	FD2Ch	0000h	TC															
General-Purpose DMA Channel 3 Registers																			
GD3CON0	130h	FD30h	0000h	ST	AST	TC	INT	*Res*		P		*Res*	TS	*Res*	DSEL !				
GD3CON1	132h	FD32h	0000h	SM/$\overline{IO}$	SAW			SINC				DM/$\overline{IO}$	DAW			DINC			
GD3SRCL	134h	FD34h	0000h	DSA[15–0]															
GD3SRCH	136h	FD36h	0000h	*Res*												DSA[19–16]			
GD3DSTL	138h	FD38h	0000h	DDA[15–0]															
GD3DSTH	13Ah	FD3Ah	0000h	*Res*												DDA[19–16]			
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GD3TC	13Ch	FD3Ch	0000h	TC															
SmartDMA Channel Pair 0 Registers CC CH																			
SD0CON	140h	FD40h	0000h	*Res*				TEPI	TBUI	TTCI	REPI	RBUI	RTCI	TXSO	RXSO	P		POLL	*Res*	TXST	RXST
SD0TRCAL	142h	FD42h	0000h	TRA[15–4]												*Res*	TRC		
SD0TRAH	144h	FD44h	0000h	*Res*												TRA[19–16]			
SD0RRCAL	146h	FD46h	0000h	RRA[15–4]												*Res*	RRC		
SD0RRAH	148h	FD48h	0000h	*Res*												RRA[19–16]			
SD0STAT	14Ah	FD4Ah	0000h	*Res*				TEP	TBU	TTC	REP	RBU	RTC	*Res*					
SD0CBD	14Ch	FD4Ch	0000h	*Res*	CRBD							*Res*	CTBD						
SD0CTAD	14Eh	FD4Eh	0000h	CTAD															
SD0CRAD	150h	FD50h	0000h	CRAD															
SmartDMA Channel Pair 1 Registers CC CH																			
SD1CON	158h	FD58h	0000h	*Res*				TEPI	TBUI	TTCI	REPI	RBUI	RTCI	TXSO	RXSO	P		POLL	*Res*	TXST	RXST
SD1TRCAL	15Ah	FD5Ah	0000h	TRA[15–4]												*Res*	TRC		
SD1TRAH	15Ch	FD5Ch	0000h	*Res*												TRA[19–16]			
SD1RRCAL	15Eh	FD5Eh	0000h	RRA[15–4]												*Res*	RRC		
SD1RRAH	160h	FD60h	0000h	*Res*												RRA[19–16]			
SD1STAT	162h	FD62h	0000h	*Res*				TEP	TBU	TTC	REP	RBU	RTC	*Res*					
SD1CBD	164h	FD64h	0000h	*Res*	CRBD							*Res*	CTBD						
SD1CTAD	166h	FD66h	0000h	CTAD															
SD1CRAD	168h	FD68h	0000h	CRAD															
SmartDMA Channel Pair 2 Registers CC CU																			
SD2CON	170h	FD70h	0000h	*Res*				TEPI	TBUI	TTCI	REPI	RBUI	RTCI	TXSO	RXSO	P		POLL	DSEL !	TXST	RXST
SD2TRCAL	172h	FD72h	0000h	TRA[15–4]												*Res*	TRC		
SD2TRAH	174h	FD74h	0000h	*Res*												TRA[19–16]			
SD2RRCAL	176h	FD76h	0000h	RRA[15–4]												*Res*	RRC		
SD2RRAH	178h	FD78h	0000h	*Res*												RRA[19–16]			
SD2STAT	17Ah	FD7Ah	0000h	*Res*				TEP	TBU	TTC	REP	RBU	RTC	*Res*					
SD2CBD	17Ch	FD7Ch	0000h	*Res*	CRBD							*Res*	CTBD						
SD2CTAD	17Eh	FD7Eh	0000h	CTAD															
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SD2CRAD	180h	FD80h	0000h	CRAD															
SmartDMA Channel Pair 3 Registers CC CU																			
SD3CON	188h	FD88h	0000h	*Res*				TEPI	TBUI	TTCI	REPI	RBUI	RTCI	TXSO	RXSO	P		POLL	DSEL !	TXST	RXST
SD3TRCAL	18Ah	FD8Ah	0000h	TRA[15–4]												*Res*		TRC	
SD3TRAH	18Ch	FD8Ch	0000h	*Res*												TRA[19–16]			
SD3RRCAL	18Eh	FD8Eh	0000h	RRA[15–4]												*Res*		RRC	
SD3RRAH	190h	FD90h	0000h	*Res*												RRA[19–16]			
SD3STAT	192h	FD92h	0000h	*Res*				TEP	TBU	TTC	REP	RBU	RTC	*Res*					
SD3CBD	194h	FD94h	0000h	*Res*	CRBD							*Res*	CTBD						
SD3CTAD	196h	FD96h	0000h	CTAD															
SD3CRAD	198h	FD98h	0000h	CRAD															
Universal Serial Bus (USB) General Configuration Registers CC CU																			
UISTAT1	1E0h	FDE0h	0000h	*Res*				D_EP_ STATINT	D_EP_ ACT	C_EP_ STATINT	C_EP_ ACT	B_EP_ STATINT	B_EP_ ACT	A_EP_ STATINT	A_EP_ ACT	OTHER_ INT	INT_EP_ ACT	CNT_EP_ NEW	CNT_EP_ ACT
UIMASK1	1E2h	FDE2h	0008h	*Res*				D_EP_ STATINT	D_EP_ ACT	C_EP_ STATINT	C_EP_ ACT	B_EP_ STATINT	B_EP_ ACT	A_EP_ STATINT	A_EP_ ACT	OI_UNM	INT_EP_ ACT	CNT_EP_ NEW	CNT_EP_ ACT
UISTAT2	1E4h	FDE4h	0000h	USB_RST	USB_SUS	USB_RES	*Res*									TSTMP_M	POS_UP	SOF_GEN	MS_SOF
UIMASK2	1E6h	FDE6h	0000h	USB_RST	USB_SUS	USB_RES	*Res*									TSTMP_M	POS_UP	SOF_GEN	MS_SOF
USBMFR	1E8h	FDE8h	0008h	*Res*									PUP_XCVER	SUSP	S_RES	S_POWER	DIS_XCVER	RWAKE	RWAKE_EN
RTFMCNT	1EAh	FDEAh	0000h	*Res*				RTFCNT											
TSTMP	1ECh	FDECh	0000h	*Res*								TSTMP							
TSTMPM	1EEh	FDEEh	0000h	*Res*								TSTMPM							
ISCTL	1F0h	FDF0h	0000h	ESOF_EN	*Res*						BYTES_SAM	*Res*			BCNT_LRATE			SAM_CLK_SEL !	
FPMCNT	1F2h	FDF2h	0000h	*Res*				FPM_CNT											
USB Control Endpoint Registers CC CU																			
CNTCTL	200h	FE00h	0000h	EP_EN	EP_NOT_ STALLED	NOT_ FLUSH	ACT_REQ	NEW_ COMMAND	COMMAND_ BUSY	*Res*									
CNTSIZ	202h	FE02h	0000h	*Res*												RCV_PKT_SIZE			
CNTDAT	206h	FE06h	00xxh	*Res*								D							
CNTRPK	208h	FE08h	00xxh	*Res*								D							
CNTDEF1	20Ah	FE0Ah	0000h	EP_NUM				EP_CFG		*Res*		EP_INT		*Res*		EP_ASET		EP_DIR	EP_TYPE
CNTDEF2	20Ch	FE0Ch	0008h	*Res*				FIFO_SIZE		EP_MX_PCT									
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

***!** = See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USB Interrupt Endpoint Registers CC CU																			
IEPCTL	210h	FE10h	0000h	EP_EN	EP_NOT_STALLED	NOT_FLUSH	ACT_REQ	*Res*											
IEPDAT	216h	FE16h	00xxh	*Res*								D							
IEPDEF1	21Ah	FE1Ah	1003h	EP_NUM				EP_CFG		*Res*	EP_INT		*Res*	EP_ASET			EP_DIR	EP_TYPE	
IEPDEF2	21Ch	FE1Ch	0410h	*Res*				FIFO_SIZE		EP_MX_PCT									
USB Data A Endpoint Registers CC CU																			
AEPCTL	220h	FE20h	0000h	EP_EN	EP_NOT_STALLED	NOT_FLUSH	ACT_REQ	STAT_INT	*Res*	NOT_ZERO	NOT_LAST_BYTE	*Res*	ISO_START	ISO_STOP	ISO_MS	FULL_PKT	SHORT_PKT	BUF_ERR	OTHER_ERR
AEPSIZ	222h	FE22h	0000h	*Res*						RPS									
AEPBUFS	224h	FE24h	0000h	*Res*								BUF_STAT							
AEPDAT	226h	FE26h	00xxh	*Res*								D							
ARCVPK	228h	FE28h	00xxh	*Res*								D							
AEPDEF1	22Ah	FE2Ah	2006h	EP_NUM				EP_CFG		*Res*	EP_INT		*Res*	EP_ASET			EP_DIR	EP_TYPE	
AEPDEF2	22Ch	FE2Ch	0408h	*Res*					FIFO_SIZE	EP_MX_PCT									
AEPDEF3	22Eh	FE2Eh	0018h	*Res*		AUTO_RATE_EN	ISO_MS_IMSK	FULL_PKT_IMSK	SHRT_PKT_IMSK	BUF_ERR_IMSK	OTH_ERR_IMSK	MODE			ISO_MS_SMSK	FULL_PKT_SMSK	SHRT_PKT_SMSK	BUF_ERR_SMSK	OTH_ERR_SMSK
USB Data B Endpoint Registers CC CU																			
BEPCTL	230h	FE30h	0000h	EP_EN	EP_NOT_STALLED	NOT_FLUSH	ACT_REQ	STAT_INT	*Res*	NOT_ZERO	NOT_LAST_BYTE	*Res*	ISO_START	ISO_STOP	ISO_MS	FULL_PKT	SHORT_PKT	BUF_ERR	OTHER_ERR
BEPSIZ	232h	FE32h	0000h	*Res*						RPS									
BEPBUFS	234h	FE34h	0000h	*Res*								BUF_STAT							
BEPDAT	236h	FE36h	00xxh	*Res*								D							
BRCVPK	238h	FE38h	00xxh	*Res*								D							
BEPDEF1	23Ah	FE3Ah	3006h	EP_NUM				EP_CFG		*Res*	EP_INT		*Res*	EP_ASET			EP_DIR	EP_TYPE	
BEPDEF2	23Ch	FE3Ch	0408h	*Res*					FIFO_SIZE	EP_MX_PCT									
BEPDEF3	23Eh	FE3Eh	0018h	*Res*		AUTO_RATE_EN	ISO_MS_IMSK	FULL_PKT_IMSK	SHRT_PKT_IMSK	BUF_ERR_IMSK	OTH_ERR_IMSK	MODE			ISO_MS_SMSK	FULL_PKT_SMSK	SHRT_PKT_SMSK	BUF_ERR_SMSK	OTH_ERR_SMSK
NAME	**OFFSET**	**DEFAULT LOCATION**	**DEFAULT VALUE**	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USB Data C Endpoint Registers CC CU																			
CEPCTL	240h	FE40h	0000h	EP_EN	EP_NOT_ STALLED	NOT_ FLUSH	ACT_REQ	STAT_INT	*Res*	NOT_ ZERO	NOT_ LAST_ BYTE	*Res*	ISO_ START	ISO_ STOP	ISO_MS	FULL_ PKT	SHORT_ PKT	BUF_ERR	OTHER_ ERR
CEPSIZ	242h	FE42h	0000h	*Res*						RPS									
CEPBUFS	244h	FE44h	0000h	*Res*								BUF_STAT							
CEPDAT	246h	FE46h	00xxh	*Res*								D							
CRCVPK	248h	FE48h	00xxh	*Res*								D							
CEPDEF1	24Ah	FE4Ah	4006h	EP_NUM				EP_CFG		*Res*	EP_INT		*Res*	EP_ASET			EP_DIR	EP_TYPE	
CEPDEF2	24Ch	FE4Ch	0C08h	*Res*				FIFO_SIZE		EP_MX_PCT									
CEPDEF3	24Eh	FE4Eh	0018h	*Res*		AUTO_ RATE_EN	ISO_MS_ IMSK	FULL_PKT_ IMSK	SHRT_PKT_ IMSK	BUF_ERR_ IMSK	OTH_ERR_ IMSK	MODE			ISO_MS_ SMSK	FULL_PKT_ SMSK	SHRT_PKT_ SMSK	BUF_ERR_ SMSK	OTH_ERR_ SMSK
USB Data D Endpoint Registers CC CU																			
DEPCTL	250h	FE50h	0000h	EP_EN	EP_NOT_ STALLED	NOT_ FLUSH	ACT_REQ	STAT_INT	*Res*	NOT_ ZERO	NOT_ LAST_ BYTE	*Res*	ISO_ START	ISO_ STOP	ISO_MS	FULL_ PKT	SHORT_ PKT	BUF_ERR	OTHER_ ERR
DEPSIZ	252h	FE52h	0000h	*Res*						RPS									
DEPBUFS	254h	FE54h	0000h	*Res*								BUF_STAT							
DEPDAT	256h	FE56h	00xxh	*Res*								D							
DRCVPK	258h	FE58h	00xxh	*Res*								D							
DEPDEF1	25Ah	FE5Ah	5006h	EP_NUM				EP_CFG		*Res*	EP_INT		*Res*	EP_ASET			EP_DIR	EP_TYPE	
DEPDEF2	25Ch	FE5Ch	0C08h	*Res*				FIFO_SIZE		EP_MX_PCT									
DEPDEF3	25Eh	FE5Eh	0018h	*Res*		AUTO_ RATE_EN	ISO_MS_ IMSK	FULL_PKT_ IMSK	SHRT_PKT_ IMSK	BUF_ERR_ IMSK	OTH_ERR_ IMSK	MODE			ISO_MS_ SMSK	FULL_PKT_ SMSK	SHRT_PKT_ SMSK	BUF_ERR_ SMSK	OTH_ERR_ SMSK
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
High-Speed Asynchronous Serial Port (High-Speed UART) Registers																			
HSPCON0	260h	FE60h	0000h	*Res*			RSIE	BRK	AB	FC	TXIE	RXIE	TMODE	RMODE	EVN	PEN	ABEN	D7	STP2
HSPCON1	262h	FE62h	0000h	TFEN	RFEN	TFLUSH	RFLUSH	ABAUD	*Res*			MEN	MAB2	MAB1	MAB0	BRKVAL	EXDWR	EXDRD	XTRN !
HSPSTAT	264h	FE64h	0000h	RTHRSH	TTHRSH	*Res*	OERIM	*Res*	MATCH	BRK	AB	RDR	THRE	FER	OER	PER	TEMT	IDLED	IDLE
HSPIMSK	266h	FE66h	02F8h	RTHRSH	TTHRSH	*Res*	OERIM	*Res*	MATCH	BRK	AB	RDR	THRE	FER	OER	PER	TEMT	IDLED	IDLE
HSPTXD	268h	FE68h	0000h	*Res*							AB	TDATA							
HSPRXD	26Ah	FE6Ah	0000h	RDR	THRE	FER	OER	PER	MATCH	BRK	AB	RDATA							
HSPRXDP	26Ch	FE6Ch	0000h	RDR	THRE	FER	OER	PER	MATCH	BRK	AB	RDATA							
HSPBDV	26Eh	FE6Eh	0000h	BAUDDIV															
HSPM0	270h	FE70h	0000h	MCHR1								MCHR0							
HSPM1	272h	FE72h	0000h	MCHR3								MCHR2							
HSPM2	274h	FE74h	0000h	MCHR5								MCHR4							
HSPAB0	276h	FE76h	0000h	ABDIV0								ABTHRSH0							
HSPAB1	278h	FE78h	0000h	ABDIV1								ABTHRSH1							
HSPAB2	27Ah	FE7Ah	0000h	ABDIV2								ABTHRSH2							
HSPAB3	27Ch	FE7Ch	0000h	ABDIV3								ABTHRSH3							
Asynchronous Serial Port (UART) Registers																			
SPCON0	280h	FE80h	0000h	*Res*			RSIE	BRK	AB	FC	TXIE	RXIE	TMODE	RMODE	EVN	PEN	ABEN	D7	STP2
SPCON1	282h	FE82h	0000h	*Res*												BRKVAL	EXDWR	EXDRD	XTRN !
SPSTAT	284h	FE84h	0000h	*Res*						BRK	AB	RDR	THRE	FER	OER	PER	TEMT	IDLED	IDLE
SPIMSK	286h	FE86h	02F8h	*Res*						BRK	AB	RDR	THRE	FER	OER	PER	TEMT	IDLED	IDLE
SPTXD	288h	FE88h	0000h	*Res*							AB	TDATA							
SPRXD	28Ah	FE8Ah	0000h	RDR	THRE	FER	OER	PER	*Res*	BRK	AB	RDATA							
SPRXDP	28Ch	FE8Ch	0000h	RDR	THRE	FER	OER	PER	*Res*	BRK	AB	RDATA							
SPBDV	28Eh	FE8Eh	0000h	BAUDDIV															
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
General Circuit Interface (GCI) Registers CC																			
GPCON	2A0h	FEA0h	0000h	*Res*							PCMFSC	MCARV	MARQ	MCHEN	MCHSEL	MEOMRQ	ICSEL	GCIACT	BRDIS
GISTAT	2A2h	FEA2h	0002h 0102h	*Res*						IC	DCLST	CHGCI1	CHGCI0	MRAD	MCD	MTARD	MEOMRD	MXBA	MRDA
GIMSK	2A4h	FEA4h	0000h	*Res*						IC	DCLST	CHGCI1	CHGCI0	MRAD	MCD	MTARD	MEOMRD	MXBA	MRDA
GTIC	2A6h	FEA6h	0007h	*Res*								TICEN	ECHOEN	*Res*			TICAD		
GICTD	2A8h	FEA8h	00FFh	*Res*								IC12T							
GICRD	2AAh	FEAAh	0000h	*Res*								IC12R							
GICRDP	2ACh	FEACh	0000h	*Res*								IC12P							
GCITD0	2AEh	FEAEh	000Fh	*Res*								BAR	*Res*			CI0T			
GCIRD0	2B0h	FEB0h	000Fh	*Res*												CI0R			
GCIRD0P	2B2h	FEB2h	000Fh	*Res*												CI0P			
GCITD1	2B4h	FEB4h	003Fh	*Res*										CI1T					
GCIRD1	2B6h	FEB6h	003Fh	*Res*										CI1R					
GCIRD1P	2B8h	FEB8h	003Fh	*Res*										CI1P					
GMTD	2BAh	FEBAh	00FFh	*Res*								MON01T							
GMRD	2BCh	FEBCh	0000h	*Res*								MON01R							
GMRDP	2BEh	FEBEh	0000h	*Res*								MON01P							
Time Slot Assigner (TSA) Channel A Registers CC CH																			
TSACON	2C0h	FEC0h	0000h	EN	*Res*					MODE !		*Res*		FSCP	DRVLVL	*Res*	ESADJ		
TSASTART	2C2h	FEC2h	0000h	*Res*				BPSTART											
TSASTOP	2C4h	FEC4h	0000h	*Res*				BPSTOP											
Time Slot Assigner (TSA) Channel B Registers CC CH																			
TSBCON	2C8h	FEC8h	0000h	EN	*Res*					MODE !		*Res*		FSCP	DRVLVL	*Res*	ESADJ		
TSBSTART	2CAh	FECAh	0000h	*Res*				BPSTART											
TSBSTOP	2CCh	FECCh	0000h	*Res*				BPSTOP											
Time Slot Assigner (TSA) Channel C Registers CC																			
TSCCON	2D0h	FED0h	0000h	EN	*Res*					MODE !		*Res*		FSCP	DRVLVL	*Res*	ESADJ		
TSCSTART	2D2h	FED2h	0000h	*Res*				BPSTART											
TSCSTOP	2D4h	FED4h	0000h	*Res*				BPSTOP											
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Time Slot Assigner (TSA) Channel D Registers CC																			
TSDCON	2D8h	FED8h	0000h	EN	*Res*					MODE !		*Res*		FSCP	DRVLVL	*Res*	ESADJ		
TSDSTART	2DAh	FEDAh	0000h	*Res*				BPSTART											
TSDSTOP	2DCh	FEDCh	0000h	*Res*				BPSTOP											
Synchronous Serial Interface (SSI) Registers																			
SSSTAT	2F0h	FEF0h	0000h	ENHCTL	*Res*												RE/TE	DR/DT	PB
SSCON	2F2h	FEF2h	0400h	*Res*				CLKP	DENP	*Res*	MSBF	*Res*	CLKEXP			*Res*		DE1	DE0
SSTXD1	2F4h	FEF4h	0000h	*Res*								TXDATA							
SSTXD0	2F6h	FEF6h	0000h	*Res*								TXDATA							
SSRXD	2F8h	FEF8h	0000h	*Res*								RXDATA							
Interrupt Controller Registers																			
CH0CON	300h	FF00h	003Fh	*Res*												MSK	PR		
CH1CON	302h	FF02h	000Fh	*Res*											LTM	MSK	PR		
CH2CON	304h	FF04h	000Fh	*Res*										SRC !	LTM	MSK	PR		
CH3CON	306h	FF06h	000Fh	*Res*										SRC	LTM	MSK	PR		
CH4CON !	308h	FF08h	003Fh	*Res*												MSK	PR		
CH5CON !	30Ah	FF0Ah	003Fh	*Res*												MSK	PR		
CH6CON !	30Ch	FF0Ch	003Fh	*Res*												MSK	PR		
CH7CON !	30Eh	FF0Eh	003Fh	*Res*												MSK	PR		
CH8CON	310h	FF10h	000Fh	*Res*										SRC !	LTM	MSK	PR		
CH9CON	312h	FF12h	000Fh	*Res*										SRC	LTM	MSK	PR		
CH10CON	314h	FF14h	000Fh	*Res*										SRC	LTM	MSK	PR		
CH11CON	316h	FF16h	000Fh	*Res*										SRC	LTM	MSK	PR		
CH12CON	318h	FF18h	000Fh	*Res*											LTM	MSK	PR		
CH13CON	31Ah	FF1Ah	000Fh	*Res*											LTM	MSK	PR		
CH14CON	31Ch	FF1Ch	001Fh	*Res*												MSK	PR		
EOI	320h	FF20h	0000h	NSPEC	*Res*										S				
POLL	322h	FF22h	0000h	IREQ	*Res*										S				
POLLST	324h	FF24h	0000h	IREQ	*Res*										S				
IMASK	326h	FF26h	FFFFh	*Res*	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
PRIMSK	328h	FF28h	0007h	*Res*													PRM		
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INSERV	32Ah	FF2Ah	0000h	*Res*	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
REQST	32Ch	FF2Ch	0000h	*Res*	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
INTSTS	32Eh	FF2Eh	0000h	*Res*									DMA3	DMA2	DMA1	DMA0	TIM2	TIM1	TIM0
DMAHLT	330h	FF30h	0000h	DHLT	*Res*														
SHREQ	332h	FF32h	0000h	PIO35	PIO34	PIO33	PIO30	PIO29	PIO27	PIO15	PIO5	INT7	INT6	INT5	INT4	INT3	INT2	INT1	*Res*
SHMASK	334h	FF34h	FFFFh	PIO35	PIO34	PIO33	PIO30	PIO29	PIO27	PIO15	PIO5	INT7	INT6	INT5	INT4	INT3	INT2	INT1	*Res*
INTPOL	336h	FF36h	FFFFh	*Res*							INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
PIOPOL	338h	FF38h	FFFFh	PIO35	PIO34	PIO33	PIO30	PIO29	PIO27	PIO15	PIO5	*Res*							
Timer Registers																			
T0CON	340h	FF40h	0000h	EN	$\overline{\text{INH}}$	INT	RIU	*Res*						MC	RTG	P	EXT	ALT	CONT
T0CNT	342h	FF42h	0000h	TC															
T0CMPA	344h	FF44h	0000h	TC															
T0CMPB	346h	FF46h	0000h	TC															
T1CON	348h	FF48h	0000h	EN	$\overline{\text{INH}}$	INT	RIU	*Res*						MC	RTG	P	EXT	ALT	CONT
T1CNT	34Ah	FF4Ah	0000h	TC															
T1CMPA	34Ch	FF4Ch	0000h	TC															
T1CMPB	34Eh	FF4Eh	0000h	TC															
T2CON	350h	FF50h	0000h	EN	$\overline{\text{INH}}$	INT	*Res*							MC	*Res*				CONT
T2CNT	352h	FF52h	0000h	TC															
T2CMPA	354h	FF54h	0000h	TC															
Chip Select Registers																			
UMCS	3A0h	FFA0h	F01Bh F03Bh	*Res*	LB			*Res*				DA	UDEN	USIZ	*Res*		R2	R1	R0
LMCS	3A2h	FFA2h	0F1Bh	*Res*	UB			*Res*				DA	LDEN	LSIZ	*Res*		R2	R1	R0
PACS	3A4h	FFA4h	0073h	BA[19–11]									*Res*			R3	R2	R1	R0
MMCS	3A6h	FFA6h	7FDBh	BA[19–13]					*Res*					MCS0_ ONLY	*Res*		R2	R1	R0
MPCS	3A8h	FFA8h	8183h	*Res*	M[6–0]							*Res*	MS	OMSIZ	IOSIZ	R3	R2	R1	R0
DRAM Controller Registers																			
CDRAM	3AAh	FFAAh	0000h	*Res*				RC											
EDRAM	3ACh	FFACh	0000h	EN	*Res*			T											
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

Table A-1 Am186CC/CH/CU Microcontrollers Register Summary (Continued)

NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Programmable I/O (PIO) Registers																			
PIOMODE0	3C0h	FFC0h	0000h	PMODE 15	PMODE 14	PMODE 13	PMODE 12	PMODE 11	PMODE 10	PMODE 9	PMODE 8	PMODE 7	PMODE 6	PMODE 5	PMODE 4	PMODE 3	PMODE 2	PMODE 1	PMODE 0
PIODIR0	3C2h	FFC2h	1EFFh	PDIR15	PDIR14	PDIR13	PDIR12	PDIR11	PDIR10	PDIR9	PDIR8	PDIR7	PDIR6	PDIR5	PDIR4	PDIR3	PDIR2	PDIR1	PDIR0
PIODATA0	3C4h	FFC4h	????h	PDATA15	PDATA14	PDATA13	PDATA12	PDATA11	PDATA10	PDATA9	PDATA8	PDATA7	PDATA6	PDATA5	PDATA4	PDATA3	PDATA2	PDATA1	PDATA0
PIOSET0	3C6h	FFC6h	0000h	PSET15	PSET14	PSET13	PSET12	PSET11	PSET10	PSET9	PSET8	PSET7	PSET6	PSET5	PSET4	PSET3	PSET2	PSET1	PSET0
PIOCLR0	3C8h	FFC8h	0000h	PCLR15	PCLR14	PCLR13	PCLR12	PCLR11	PCLR10	PCLR9	PCLR8	PCLR7	PCLR6	PCLR5	PCLR4	PCLR3	PCLR2	PCLR1	PCLR0
PIOMODE1	3CAh	FFCAh	0000h	PMODE 31	PMODE 30	PMODE 29	PMODE 28	PMODE 27	PMODE 26	PMODE 25	PMODE 24	PMODE 23	PMODE 22	PMODE 21	PMODE 20	PMODE 19	PMODE 18	PMODE 17	PMODE 16
PIODIR1	3CCh	FFCCh	9FFFh	PDIR31	PDIR30	PDIR29	PDIR28	PDIR27	PDIR26	PDIR25	PDIR24	PDIR23	PDIR22	PDIR21	PDIR20	PDIR19	PDIR18	PDIR17	PDIR16
PIODATA1	3CEh	FFCEh	????h	PDATA31	PDATA30	PDATA29	PDATA28	PDATA27	PDATA26	PDATA25	PDATA24	PDATA23	PDATA22	PDATA21	PDATA20	PDATA19	PDATA18	PDATA17	PDATA16
PIOSET1	3D0h	FFD0h	0000h	PSET31	PSET30	PSET29	PSET28	PSET27	PSET26	PSET25	PSET24	PSET23	PSET22	PSET21	PSET20	PSET19	PSET18	PSET17	PSET16
PIOCLR1	3D2h	FFD2h	0000h	PCLR31	PCLR30	PCLR29	PCLR28	PCLR27	PCLR26	PCLR25	PCLR24	PCLR23	PCLR22	PCLR21	PCLR20	PCLR19	PCLR18	PCLR17	PCLR16
PIOMODE2	3D4h	FFD4h	0000h	PMODE 47	PMODE 46	PMODE 45	PMODE 44	PMODE 43	PMODE 42	PMODE 41	PMODE 40	PMODE 39	PMODE 38	PMODE 37	PMODE 36	PMODE 35	PMODE 34	PMODE 33	PMODE 32
PIODIR2	3D6h	FFD6h	FFF1h	PDIR47	PDIR46	PDIR45	PDIR44	PDIR43	PDIR42	PDIR41	PDIR40	PDIR39	PDIR38	PDIR37	PDIR36	PDIR35	PDIR34	PDIR33	PDIR32
PIODATA2	3D8h	FFD8h	????h	PDATA47	PDATA46	PDATA45	PDATA44	PDATA43	PDATA42	PDATA41	PDATA40	PDATA39	PDATA38	PDATA37	PDATA36	PDATA35	PDATA34	PDATA33	PDATA32
PIOSET2	3DAh	FFDAh	0000h	PSET47	PSET46	PSET45	PSET44	PSET43	PSET42	PSET41	PSET40	PSET39	PSET38	PSET37	PSET36	PSET35	PSET34	PSET33	PSET32
PIOCLR2	3DCh	FFDCh	0000h	PCLR47	PCLR46	PCLR45	PCLR44	PCLR43	PCLR42	PCLR41	PCLR40	PCLR39	PCLR38	PCLR37	PCLR36	PCLR35	PCLR34	PCLR33	PCLR32
Reset Configuration Register																			
RESCON	3DEh	FFDEh	????h	RCD15	RCD14	RCD13	RCD12	RCD11	RCD10	RCD9	RCD8	RCD7	RCD6	RCD5	RCD4	RCD3	RCD2	RCD1	RCD0
Watchdog Timer Register																			
WDTCON	3E0h	FFE0h	C180h	ENA	WRST	RSTFLAG	NMIFLAG	*Res*			EXRST	ES							
Miscellaneous Registers																			
SYSCON	3F0h	FFF0h	0000h	*Res*		DSDEN	PWD	DISMEM	DISIO	ITF4 !		EXSYNC !	*Res*			DISCLK	*Res*		
PRL	3F4h	FFF4h	4001h	PRL															
RELOC	3FEh	FFFEh	20FCh	DUAL	*Res*		M/$\overline{IO}$	R[19–10]										*Res*	
NAME	OFFSET	DEFAULT LOCATION	DEFAULT VALUE	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

! *= See the register or bit description in the Am186™CC/CH/CU Microcontrollers Register Set Manual, order #21916, for controller-specific details.*

A

A bus
Nonmultiplexed address bus.

ACK
Acknowledgment.

AD bus
Multiplexed address and data bus.

ADCCP
Advanced data communication control procedures.

ADSL
Asymmetrical digital subscriber line. *See* DSL.

ANSI
American national standards institute.

asynchronous
Pertaining to two or more processes that do not depend on the occurrence of specific events such as common timing signals.

asynchronous transmission
Data transmission in which each information character is individually synchronized (usually by the use of start and stop elements). *Compare to* synchronous transmission *and* isochronous transmission.

AT interface
A method of communicating with and controlling modems. Developed by Hayes Microcomputer Products, the AT Command Set has become a de facto standard most modems are designed to use.

B

bit stuffing
Adding bits to a transmitted message to round out a fixed frame or to break up a pattern of data bits that could be misconstrued as control codes. *Also called* zero-bit insertion. *Compare to* bit unstuffing.

bit unstuffing
Deleting bits from a received message to remove any bits added to round out a fixed frame or to break up a pattern of data bits that could be misconstrued as control codes. *Also called* zero-bit deletion. *Compare to* bit stuffing.

break
During serial communications, a constant Low signal on the receive data line for one frame time or greater. In the Am186CC/CH/CU microcontrollers, this is reported as a zero character with the framing error (FER) and break (BRK) status bits set in the (H)SPSTAT register.

BRI
Basic rate interface.

buffer
(1) A routine or storage space used to compensate for a difference in rate of data flow, or time of occurrence of events, when transferring data from one device to another. (2) A portion of storage space used to temporarily hold input or output data.

buffer queue
A block of memory to which data is written or from which data is read during a DMA transfer. Software specifies the length and base address of the buffer queue. The DMA transfer writes data to each byte or word of the buffer queue until it reaches the end of the transfer or the end of the buffer queue. *Compare to* circular buffer.

buffer descriptor ring
See descriptor ring.

bulk transfer
A nonperiodic data transmission process that typically consists of large bursts of information. Bulk transmission is typically used for a transfer that can use any available bandwidth and also can be delayed until bandwidth is available.

byte
A group of eight adjacent binary digits (bits).

C

CCIT
International telegraph and telephone consultative committee.

circular buffer
A block of memory to which data is written or from which data is read during a DMA transfer. Software specifies the length and base address of the circular buffer. The DMA transfer writes data to or reads data from each byte or word of the circular buffer. If the transfer reaches the end of the buffer, the DMA control hardware points

back to the beginning of the buffer and continues writing or reading data. Sometimes called a ring buffer. *Compare to* buffer queue.

CO
Central office.

codec
Coder-decoder. Also referred to as a compressor-decompressor. Any technology used to encode (or compress) and decode (or decompress) data, which can be done with hardware, software, or any combination of the two. Typically used for digital audio or video data streams.

control endpoint
A USB endpoint used to transfer USB commands and device configuration data between the host and device. The control endpoint is common to, and is required by, all USB device class specifications. The control endpoint features are not programmable. *Compare to* interrupt endpoint *and* data endpoint.

CPU
Central processing unit. The control unit or microprocessor of a computer system.

CRC
Cyclic redundancy check. A check performed on data to see if an error has occurred in transmitting, reading, or writing the data. The result of a CRC is typically stored or transmitted with the checked data. The stored or transmitted result is compared to a CRC value calculated for the data to determine if an error has occurred.

CTR
Clear to receive.

CTS/RTR
Clear-to-send/ready-to-receive. A symmetrical interface between two serial ports that provides hardware flow control when both ports are sending and receiving data. The $\overline{\text{CTS}}$ signal of each port is connected to the $\overline{\text{RTR}}$ signal of the other port. When the transmitter sends a $\overline{\text{CTS}}$ signal and the receiver sends a $\overline{\text{RTR}}$ signal, data can be transferred from the transmitter to the receiver between the ports.

D

data endpoint
A USB endpoint used to transfer data from the host to the device, or vice versa. Each data endpoint is individually programmable as to direction (IN or OUT relative to the host), transfer type (bulk, isochronous, or interrupt), and maximum packet size. *Compare to* control endpoint *and* interrupt endpoint.

data transparency
A data stream that happens to contain a data sequence that is the same as a flag, mark, or abort sequence is disguised during transmission so it is not misconstrued as an actual flag, mark, or abort. *See also* bit stuffing *and* bit unstuffing.

DCE
Data communications equipment. Any device that connects a computer to a network, such as a modem. *See also* raw DCE.

default address
An address defined by the USB specification and used by a USB device when it is first powered on or reset. The default address is 00h.

descriptor ring
A block of memory that the CPU and software use to control and describe data buffers.

destination-synchronized transfer
See synchronized transfer.

device
See USB device.

device address
The address of a device on the USB. The device address is the default address when the USB device is first powered on or reset. Hubs and functions are assigned a unique device address by USB software.

DMA
Direct memory access. A means of transferring data from a source (a device or block of memory) directly to a destination (also a device or block of memory) without passing the information through the processor. *See also* general-purpose DMA *and* SmartDMA channel.

DMA latency
The time period between the DMA request generation and the actual running of the bus cycles associated with the DMA transfer. *See also* latency, interrupt latency, *and* HOLD latency.

DMA mode
One of three modes supported by the Am186CC/CH/CU microcontrollers for serial communications. In DMA mode, software programs the DMA transfer registers, then the DMA hardware performs the entire transfer with no software intervention except for error-handling. *Compare to* polled mode *and* interrupt mode.

DMA transfer
A unit of work involving the transferring of data into or out of memory using DMA capabilities.

DRAM
Dynamic random access memory. A type of computer memory that employs a system of transistors and capacitors to retain data. DRAM is slower and less dependable than static RAM because the capacitors cannot maintain an electrical charge and need to be refreshed every millisecond, but it is cheaper, takes up less space, and uses less power. *Compare to* SRAM.

DSL
Digital subscriber line. A modem technology that increases the digital speed of ordinary telephone lines by a substantial factor over common V.34 (33600 bps) modems. DSL modems may provide symmetrical or asymmetrical (ADSL) operation. Asymmetrical provides faster downstream speeds and is suited for internet usage and video on demand, where the heaviest transmission requirement is from the provider to the customer.

DSL uses packet switching technology that operates independently from the voice telephone system. This allows the telephone companies to provide digital service and not lock up voice circuits for long calls. Because of this, DSL is not as well suited to videoconferencing as is ISDN. ISDN is circuit switched, which keeps the line open and connected throughout the session.

DTE
Data terminal equipment. A hardware component connected to some type of communications device. A PC is a piece of data terminal equipment; a modem is a communications device.

duplex
The ability of a serial communications connection to transmit data in both directions. *See also* half duplex *and* full duplex. *Compare to* simplex.

E

EDO
Extended data out.

endpoint
See USB endpoint.

endpoint address
The combination of a device address and an endpoint number on a USB device.

endpoint number
A unique pipe endpoint on a USB device.

EOM
End of message.

even parity
See parity.

external reset
The reset of the Am186CC/CH/CU microcontrollers initiated by asserting the $\overline{\text{RES}}$ signal. Also called a power-on reset. *Compare to* internal reset *and* system reset.

F

FCS
Frame check sequence. The FCS contains the generated CRC code for the frame being transmitted. All data transmitted between the opening and closing flags (excluding inserted 0s) is included in the CRC calculation. The transmitter appends the calculated CRC to the end of the frame just before the closing flag.

FIFO
First in first out. (1) Describes a method of processing data in the order in which it is received. (2) A block of memory or other storage used as a first-in-first-out buffer.

FIFO high-water mark
See FIFO threshold.

FIFO threshold
A system-dependent, software threshold value that indicates action should be taken so data is not lost from the FIFO buffer.

fly-by-transfer
During a SmartDMA channel transfer, the read and write operations execute in a single bus cycle, instead of the two cycles required during a general-purpose DMA transfer.

frame
The unit of information transferred across a data link. Typically, there are control frames for link management and information frames for the transfer of message data.

frame synchronization (frame sync)
During an HDLC transfer, the process of signaling the beginning of the frame with a start flag and the end of the frame with a stop flag.

framing error
In asynchronous serial communication, a condition resulting from the receiver losing bit count alignment with the transmitter. In this situation, if the last bit of a unit (a frame) is a zero, the receiver may read that bit as the start bit of the next frame, thus the term framing error.

full duplex
The ability of a serial communications connection to transmit data in both directions at the same time. *See also* duplex *and* simplex. *Compare to* half duplex.

G

GCI
General circuit interface, also called IOM-2. One of the external interfaces supported by the Am186CC communications controller HDLC channels. GCI is an interface specification developed jointly by Alcatel, Italtel, GPT, and Siemens. This specification defines an industry standard serial bus for interconnecting telecommunications integrated circuits. The standard covers linecard, NT1, and terminal architectures for Integrated Services Digital Network (ISDN) applications. The Am186CC communications controller supports the terminal version of GCI. GCI on the Am186CC communications controller supports polled and interrupt modes, but does not support DMA mode. *See* IOM-2.

general-purpose DMA
The term used to describe standard or typical DMA processing as opposed to DMA processing using the SmartDMA channels in the Am186CC/CH/CU microcontrollers. *See also* DMA *and* SmartDMA.

H

half duplex
The ability of a serial communications connection to transmit data in both directions, but not at the same time. *Compare to* full duplex.

hardware interrupt
Any one of the maskable interrupts, or an NMI or watchdog timer interrupt. When a hardware interrupt is generated, the IF flag is cleared unless in polled mode. *Compare to* software interrupt.

HDLC
High-level data link control. A very common bit-oriented data link protocol (OSI layer 2) issued by ISO. Similar protocols are ADCCP, LAP-B, and SDLC. The Am186CC communications controller provides four HDLC channels. The HDLC channels support full-duplex transfers in polled, interrupt, and DMA modes.

HOLD latency
The time between a HOLD request and the HOLD acknowledge.

host
The host computer system where the USB host controller is installed. This includes the host hardware platform (CPU, bus, etc.) and the operating system in use.

I

ICE
In-circuit emulator. A device for testing and programming an integrated circuit outside of any actual system in which the device will be used.

internal peripherals
Components on a microcontroller integrated circuit other than the embedded CPU that provide control over some specific function. On the Am186CC/CH/CU microcontrollers, internal peripherals would include but not be limited to the HDLC controller, the DMA controller, the USB peripheral controller, and the DRAM controller. For lists of the internal peripherals in the Am186CC/CH/CU microcontrollers, see “Features” on page 1-1.

internal reset
The reset of the Am186CC communications controller initiated by the watchdog timer. *Compare to* external reset *and* system reset.

interrupt
A command or signal that tells the processor to stop what it is doing and wait for further instruction. The interrupt may require the processor to suspend its current job and perform another function that is more pressing.

interrupt channel
The group of logic that is comprised of a control register, an in-service bit, a request bit, and a mask bit. The interrupt channel controls the behavior of a maskable interrupt.

interrupt endpoint
A USB endpoint used for small data transfers that in the past have been interrupt-driven. The interrupt endpoint is polled at a regular programmable interval to allow the device to transfer interrupt data such as event notification, keyboard characters, and pointing device coordinates to the host. *Compare to* control endpoint *and* data endpoint.

interrupt latency
The time period between an interrupt request and the servicing of the interrupt. *See also* latency, DMA latency, *and* HOLD latency.

interrupt mode
One of three modes supported by the Am186CC communications controller for serial communications. In interrupt mode, software performs other tasks until an interrupt tells it to service a serial channel. *Compare to* DMA mode *and* polled mode.

interrupt source
Any source (internal or external) that can request an interrupt. This can be a physical pin, or an on-chip peripheral.

interrupt transfer
One of four USB transfer types. Interrupt transfers have the following characteristics: small data, nonperiodic, low frequency, and bounded latency. They are device-initiated communications typically used to notify the host of device service needs.

interrupt type
An eight-bit number assigned to each discrete interrupt (see Table 7-3 on page 7-12). Each interrupt type does not need a unique interrupt channel; one interrupt channel can support more than one interrupt type. However, if two interrupt types are supported by one channel, then those two types have the same level of programmable priority.

interrupt vector address
Equals the interrupt type times four and is the location in the interrupt vector table that stores the address of the interrupt service routine for each interrupt type.

interrupt vector table
A memory area of 1 Kbyte beginning at address 00h that contains up to 256 four-byte interrupt vector addresses.

IOM-2
ISDN-oriented modular interface, revision 2. *See* GCI.

ISDN
Integrated services digital network. A telecommunications network that allows for digital voice, video, and data transmissions. ISDN replaces the analog telephone system with a fast and efficient digital communications network. ISDN lines contain two channels: a B channel, which has a 64-Kbit/s data transmission rate, and a D channel, which has either a 16-Kbit/s or 64-Kbit/s transmission rate. When the two lines are used together, transmitted data can travel at 128 Kbit/s.

isochronous transmission
A data transmission process in which there is always an integral number of unit intervals between any two significant instants. *Compare to* synchronous transmission *and* asynchronous transmission.

isochronous transfer
One of four USB transfer types. Isochronous transfers are used when working with isochronous data. Isochronous transfers provide periodic, continuous communication between host and device.

ISR
Interrupt service routine. The software executed when the interrupt processing unit receives an interrupt request. The interrupt vector points to this code.

L

LANCE
Local area network controller for ethernet.

LAP-B
Link access procedure, balanced.

LAP-D
Link access procedure, D channel.

latency
A time period for an event to cause another event. *See also* interrupt latency, DMA latency, *and* HOLD latency.

LSB
Least significant bit.

M

maskable interrupt
An interrupt that can be enabled (unmasked) or disabled (masked) by setting or clearing a bit in the appropriate mask register. Maskable interrupts as a group are enabled and disabled by setting or clearing the Interrupt-Enable Flag (IF) in the Processor Status Flags (FLAGS) register. Nonmaskable interrupts are not affected by this bit setting.

message pipe
A pipe that transfers data using a request/data/status paradigm. The data has an imposed structure that allows requests to be reliably identified and communicated.

MSB
Most significant bit.

multidrop
A communication configuration in which more than two stations share a transmission path. A typical multidrop configuration has a number of secondary devices (e.g., terminals) and a primary device (e.g., host computer) on the same path or line.

multiplexed mode
The connection of an HDLC channel to an external interface through a TSA. In multiplexed mode, an HDLC channel can be connected to a PCM highway or GCI interface. *Compare to* nonmultiplexed mode.

multiplexed signal
A signal that shares a pin with at least one other signal.

multipoint
See multidrop.

mux
Abbreviation for multiplexer.

N

NACK
Negative acknowledgment.

nibble
Half a byte (four bits).

NMI
Nonmaskable interrupt. An interrupt that cannot be disabled (masked).

nonmultiplexed mode
The connection of an HDLC channel directly to an external interface without going through a TSA. In nonmultiplexed mode, an HDLC channel can be connected to a raw DCE interface. *Compare to* multiplexed mode.

NRZ
Non-return to zero.

NRZI
Non-return to zero, inverted.

O

odd parity
See parity.

ONCE
On-circuit emulation.

OSI
Open systems interconnection. An ISO standard for worldwide communications that defines a framework for implementing protocols in seven layers. Control is passed from one layer to the next.

overall priority
Each interrupt source has an overall priority number that is used only to arbitrate between two interrupt sources that have priority requests pending with the same programmable priority level. Overall priority is not used if the programmable priority is sufficient to resolve the pending highest-priority request.

P

PABX
Private automatic branch exchange.

packet
A self-contained message unit transmitted through a communications network. Typically, the transmitter breaks a longer message into packets to avoid the network performance degradation caused by long messages. A packet contains three parts: control information (source, destination address, length), the data to be transmitted, and error detection and correction bits. A packet may be made up of one or more frames.

packet buffer
The logical buffer used by a USB device for sending or receiving a single packet. This determines the maximum packet size the device can send or receive.

packet ID
A field in a USB packet that indicates the type of packet, and by inference the format of the packet and the type of error detection applied to the packet.

parity
An error-checking procedure for checking the accuracy of serial data streams based on whether the number of 1 bits is even or odd. A parity bit is added to each group of data bits in a transmission. In *even parity*, the parity bit is set to 1 whenever it is needed to bring the total number of 1 bits to an even number. In *odd parity*, the parity bit is set to 1 whenever it is needed to bring the total number of 1 bits to an odd number.

PCB
Peripheral control block. Each 16-bit read/write peripheral register is in the internal 1-Kbyte peripheral control block (PCB). Registers are physically located in the peripheral devices they control, but they are addressed as a single 1-Kbyte block. This block is located in either memory or I/O space, at the location pointed to by the Peripheral Control Block Relocation (RELOC) register. Because the base address of the block can change, the address of each register is specified as an offset from the RELOC register, rather than as an absolute address. The register address is found by adding the offset to the base address to determine the physical location in memory or I/O space.

The PCB base address can be set to any even 1-Kbyte boundary in memory or I/O space (i.e., the lower 10 bits of the base address must be 0). The RELOC register resides in the last register address of the PCB, at offset 03FEh. At reset, the base of the PCB is set to FC00h in I/O space. This places the RELOC register at FFFEh.

PCM
Pulse code modulation. A technique for converting analog signals into digital form that is widely used by the telephone companies in their T1 circuits. In North America and Japan, PCM samples the analog waves 8,000 times per second and converts each sample into an 8-bit number, resulting in a 64-Kbit/s data stream. The sampling rate is twice the 4-KHz bandwidth required for a toll-quality conversation.

PCM highway
Pulse code modulation highway. One of the external interfaces supported by the Am186CC communications controller HDLC channels.

pin
Refers to a physical wire on a chip which is available externally. *Compare to* signal.

pinstrap
A pinstrap is used to enable or disable features based on the state of the pin during an external reset. The pinstrap must be held in its desired state for at least 4.5 clock cycles after the deassertion of the $\overline{RES}$ signal. Note that the pinstraps are sampled in an external reset only (when the $\overline{RES}$ signal is asserted) not during an internal watchdog-timer generated reset.

PIO
Programmable input/output. Physical pins on the Am186CC communications controller that can be used for any purpose the system designer requires. The signal on a PIO pin can be sampled through a register and can be driven High or Low by setting or clearing the associated bit in the appropriate register.

pipe
A logical abstraction representing the association between an endpoint on a USB device and software on the host. A pipe has several attributes; for example, a pipe may transfer data as streams (stream pipe) or messages (message pipe).

polled mode
One of three modes supported by the Am186CC communications controller for serial communications. In polled mode, software reads a status register in a loop, and reads received data or transmits data depending on the status register indicator bits. *Compare to* interrupt mode *and* DMA mode.

port
Point of access to or from a system or circuit. For USB, the point where a USB device is attached.

POTS
Plain old telephone service.

power-on reset
See external reset.

PPP
Point to point protocol.

PRI
Primary rate interface.

programmable priority
Each interrupt channel has eight levels of programmable priority that are set in the channel's control register. Programmable priority determines which interrupt to service when two interrupts are requested at the same time. An interrupt service routine is interrupted by another interrupt request of equal or higher programmable priority, as long as the Interrupt-enable Flag (IF) in the Processor Status Flags (FLAGS) register is set. For more information about setting the FLAGS register, see the *Am186™CC/CH/CU Microcontrollers Register Set Manual*, order #21916. If the programmable priority levels are equal, the overall priority number is used.

PWD
Pulse width demodulation.

R

raw DCE
One of the external interfaces supported by the Am186CC communications controller HDLC channels. Raw DCE is a synchronous serial bus generally used in modem and other high-speed serial applications. Raw DCE runs at up to 10 Mbit/s. The Am186CC communications controller implementation requires transmit (TCLK) and receive (RCLK) clock inputs, and has receive data (RXD), transmit data (TXD), and the Clear-To-Send (CTS) and Ready-To-Receive (RTR) flow control signals.

receiver
The portion of logic for an HDLC channel, SmartDMA channel, or UART that processes information coming into the Am186CC communications controller.

reset
See external reset*,* internal reset*, and* system reset.

ring buffer
See circular buffer.

router
The part of a communications network that receives transmissions and forwards them to their destinations using the shortest route available. Data may travel through multiple routers on the way to their destination.

RTR
Ready-to-receive. *See* CTS/RTR.

RTS
Ready-to-send.

S

SCIT
Special circuit interface for terminals.

SDLC
Synchronous data link control. A data transmission protocol used by networks using Systems Network Architecture (a communications format, advanced by IBM, used on local-area networks to allow multiple systems access to centralized data). SDLC defines the format used to transmit the data traveling over network lines.

short frame
During an HDLC transfer, a frame containing a number of bytes between the start and stop flags that is less than the minimum length specified in the MINRL bit field of the HxRCON0 register.

signal
Refers to the electrical signal that flows across a pin. *Compare to* pin.

simplex
The ability of a serial communications connection to transmit data in one direction only. *Compare to* duplex.

SLIC
Subscriber line interface circuit.

SLAC™
Subscriber line audio-processing circuit.

SmartDMA™ channel
An AMD proprietary technique for increasing the performance of DMA transfers. SmartDMA channels provide a method for the transmission and reception of data across multiple memory buffers and a sophisticated buffer-chaining mechanism. These channels are always used in pairs: transmitter and receiver. The transmit channels can only transfer data from memory to a peripheral; the receive channels can only transfer data from a peripheral to memory. *See also* DMA *and* general-purpose DMA.

SOHO
Small office/home office.

SRAM
Static random access memory. A type of semiconductor memory that preserves stored information as long as there is enough power flow to keep the device running. SRAM does not need refreshing like DRAM. SRAM is faster and more dependable, but also is more expensive, takes up more space, and uses more power than DRAM.

software exception
A software interrupt that occurs when an instruction causes a particular condition in the processor.

software interrupt
An interrupt initiated by the INT or INTO software instruction, or by a software exception. A software interrupt does not affect the IF flag in the FLAGS register. *Compare to* hardware interrupt.

source-synchronized transfer
See synchronized transfer.

SSI
Synchronous serial interface. An AMD proprietary technology for providing half-duplex, bidirectional data transfers at transfer rates of up to 25 Mbit/s with a 50-MHz CPU clock. The SSI supports only polled mode, not interrupt or DMA modes.

stream pipe
A pipe that transfers data as a stream of samples with no defined USB structure.

synchronization type
A classification that characterizes an isochronous endpoint's capability to connect to other isochronous endpoints.

synchronized transfer
A transfer of information in which the transmitter and receiver coordinate their operations with a clock signal or some other technique so the receiver knows when the next piece of information is available from the transmitter. In DMA operations, a synchronized transfer takes place when either the source of the data (source-synchronized) or the destination of the data (destination-synchronized) generates a DRQ to request the transfer. *Compare to* unsynchronized transfer.

synchronous transmission
Data transmission in which the time of occurrence of each signal representing a bit is related to a fixed time frame. Typically, the transmitter sends a clock signal along with the data so the receiver knows when to receive each bit. *Compare to* asynchronous transmission *and* isochronous transmission.

system reset
The reset of the Am186CC/CH/CU microcontrollers (the CPU plus the internal peripherals) as well as any external peripherals connected to the RESOUT pin. An external reset always causes a system reset; an internal reset can optionally cause a system reset. *Compare to* internal reset *and* external reset.

T

TDM
Time-division multiplex. A method of transmitting multiple signals (data, voice, and/or video) simultaneously over one communications medium by interleaving a piece of each signal one after another.

TIC
Terminal interchip communication.

top of FIFO
The memory address or register where the next item in a first-in-first-out buffer can be read.

trace interrupt
The trace interrupt is the highest priority interrupt. It is a software interrupt in that it is initiated by software, but unlike other software interrupts, it does clears the IF flag.

transparency
See data transparency.

transparent mode
A mode of operation for an HDLC transmit channel that transmits the data exactly as it appears in the FIFO. Transparent mode does no bit stuffing, no framing with flags, and does not support CRC. Transparent mode is useful for transmitting raw data streams such as audio data (for use with a codec or DSP).

transaction
The delivery of service to an endpoint. A transaction consists of a token packet, an optional data packet, and an optional handshake packet. Specific packets are allowed or required based on the transaction type.

transceiver
A transmitter/receiver that can send and accept information.

transfer
One or more bus transactions to move information between a software client and its function.

transfer type
Determines the characteristics of the data flow between a software client and its function. Four USB transfer types are defined: control, interrupt, bulk, and isochronous.

transmitter
The portion of logic for an HDLC channel or SmartDMA channel that sends information out from the Am186CC/CH/CU microcontrollers.

TSA
Time-slot assigner. The portion of logic in an HDLC channel that directs data from the HDLC channel to an external communication interface or vice versa. A TSA's main function is to allow the transmission and reception of data to and from an individual HDLC by providing the appropriate HDLC clock and clock enable signals during its programmed time slot within an 8-KHz frame. The Am186CC/CH/CU microcontrollers support the following external interfaces: raw DCE and PCM Highway. In addition, the Am186CC communications controller supports GCI.

U

UART
Universal asynchronous receiver/transmitter. A device that provides full-duplex, bidirectional data transfer in RS-232 format. The Am186CC/CH/CU microcontrollers have a UART that supports speeds up to 115.2 Kbaud and a High-Speed UART that supports speeds up to 460 Kbaud. The UARTs support full-duplex transfers in polled, interrupt, and DMA modes.

unsynchronized transfer
A transfer of information in which the transmitter sends data without regard for any signal or other indication from the receiver. During an unsynchronized transfer in DMA operations, DRQ is always asserted; and the transfer takes place continually until the correct number of transfers occur. *Compare to* synchronized transfer.

USB
Universal serial bus. USB is an industry standard extension to the PC architecture that provides an easy-to-use port for connecting up to 127 peripheral devices at transfer rates up to 12 Mbit/s. The USB specification supports isochronous (real-time) data transfers for voice, audio, and compressed video; bulk data transfers for devices such as printers and terminal adapters; and interrupt data transfers for event-driven devices such as pointing devices and keyboards.

The USB portion of the Am186CC and Am186CU microcontrollers supports half-duplex transfers in polled, interrupt, and DMA modes.

USB device
A logical or physical entity that performs a function. The actual entity described depends on the context of the reference. At the lowest level, the term device may refer to a single hardware component, as in a memory device. At a higher level, it may refer to a collection of hardware components that perform a particular function, such as a USB interface device. At an even higher level, the term device may refer to the function performed by an entity attached to the USB; for example, a data/FAX modem device. Devices may be physical, electrical, addressable, and logical.

USB endpoint
A uniquely identifiable portion of a USB device that is the source or sink of information in a communication flow between the host and device. Each endpoint is supported by a first-in-first-out buffer (FIFO). The FIFO is a temporary storage location for the data that is passed between the microcontroller's CPU or memory bus and the integrated USB peripheral controller. *See also* control endpoint, data endpoint, *and* interrupt endpoint.

V

very short frame
During an HDLC transfer, a frame containing less than two bytes (zero or one) between the start and stop flags.

W

wait state
A pause in a microprocessor's clock cycles that allows for differences in speed between one component and others in a computer (such as input/output devices or RAM). Wait states are common in systems where the microprocessor has a much higher clock speed than other components, requiring the latter to play catch up. During a wait state, the microprocessor idles for one or more cycles while data comes in from RAM or other components. Wait states also are not uncommon between buses and devices connected to the bus.

WAN
Wide area network.

word
In the x86 environment, a group of 16 adjacent binary digits (bits) or two bytes.

Z

zero-bit deletion
See bit unstuffing.

zero-bit insertion
See bit stuffing.

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z