Advance Information

Am29202™

Low-Cost RISC Microcontroller with IEEE-1284-Compliant Parallel Interface

Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

- **Completely integrated system for cost-sensitive embedded applications requiring high performance**
- **Full 32-bit RISC architecture offers faster instruction execution and higher performance.**
  - — 32-bit instruction/data bus
  - — 22-bit address bus
  - — 192 general-purpose registers
  - — Fully pipelined, three-address instruction architecture
  - — 104-Mbyte address space
  - — 12-, 16-, and 20-MHz operating frequencies
  - — 16 VAX MIPS sustained at 20 MHz
- **Glueless system interfaces with on-chip wait state control lower total system cost.**
  - — ROM controller supports four banks of ROM, each separately programmable for 8-, 16-, or 32-bit-wide interface.
  - — DRAM controller supports four banks of DRAM, each separately programmable for 16- or 32-bit-wide interface.
  - — 2-port peripheral interface adapter (PIA)
- **Two-channel DMA controller (one external) with queued reload for internal peripherals**
- **On-chip timer and interrupt controller**
- **IEEE Std 1284-1994-compliant parallel port interface (peripheral-side only) supports fast bidirectional data transfers.**
  - — Compatibility, Nibble, Byte, and ECP modes
  - — Supports Microsoft® Windows® Printing System
- **Bidirectional bit serializer/deserializer for direct connection to raster input and output devices**
- **12-line programmable I/O port (8 lines interruptible)**
- **DRAM page-mode support improves memory access time.**
- **On-chip DRAM mapping reduces memory requirements.**
- **Advanced debugging support**
  - — IEEE Std 1149.1-1990-compliant Standard Test Access Port and Boundary Scan Architecture (JTAG) for testing system hardware
  - — Instruction tracing
  - — UART serial port
- **Software and hardware development tools widely available from AMD® and Fusion29K® partners**
- **Binary compatibility with all 29K™ Family of RISC microcontrollers and microprocessors**
- **132-pin Plastic Quad Flat Pack (PQFP) package**

GENERAL DESCRIPTION

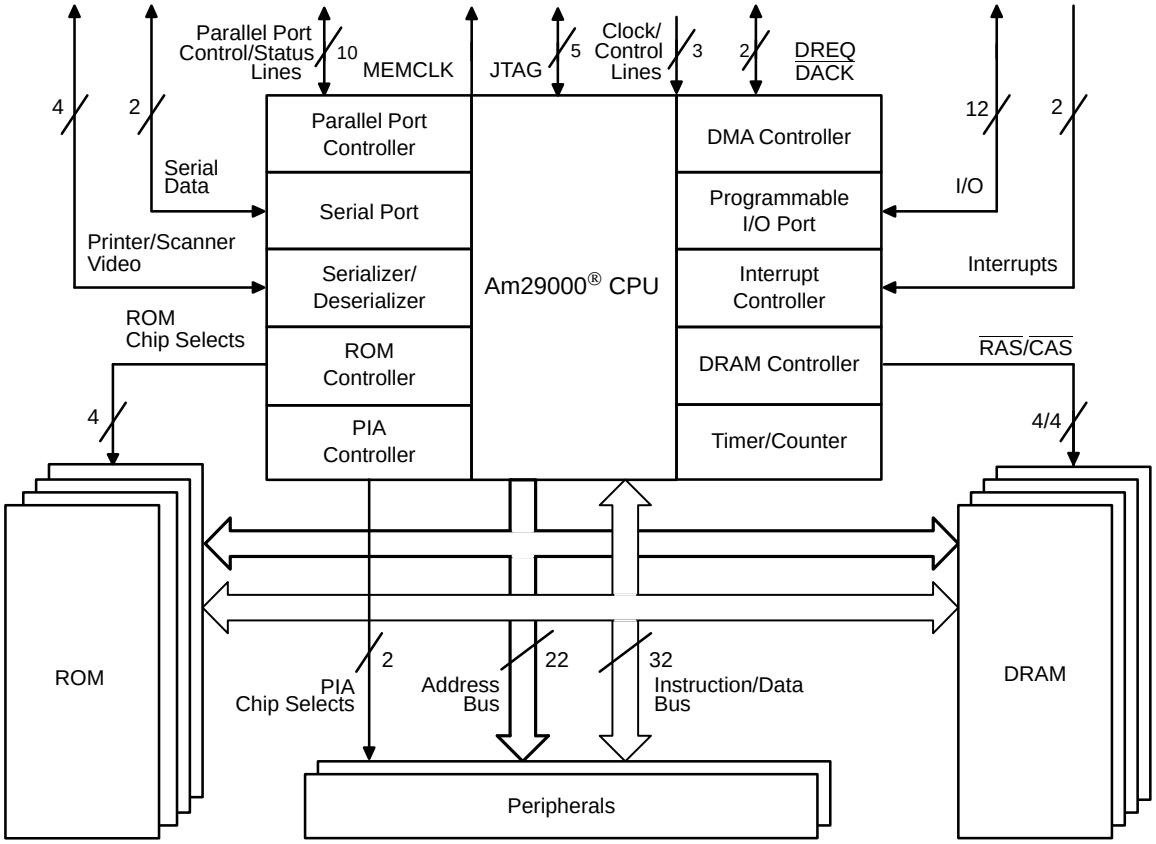
The Am29202™ RISC microcontroller is a highly integrated, 32-bit embedded processor implemented in complementary metal-oxide semiconductor (CMOS) technology. Based on the 29K architecture, the Am29202 microcontroller is part of a growing family of RISC microcontrollers, which includes the Am29200™ and Am29205™ microcontrollers, along with the high-performance Am29240™, Am29245™, and Am29243™ RISC microcontrollers. A feature summary of the Am29200 RISC microcontroller family is included in Table 1.

With its 32-bit instruction and data bus, the Am29202 microcontroller is functionally very similar to an Am29200 microcontroller, operating with a reduced pin count and fewer peripherals. The low-cost Am29202 microcontroller is well-suited for cost-sensitive embedded applications requiring the enhanced performance of a 32-bit instruction/data bus and an IEEE-1284-compliant parallel port interface. The Am29202 microcontroller incorporates a complete set of system facilities commonly found in printing, imaging, graphics, and other embedded applications.

The Am29202 microcontroller meets the common requirements of embedded applications such as laser printers, imaging applications, graphics processing, industrial control, and general purpose applications requiring high performance in a compact design. Specific applications include products based on Microsoft's Windows Printing System, such as personal and workgroup 600-DPI laser printers and midrange inkjet printers, as well as scanners and multifunction peripherals, among others.

Publication# **19716** Rev. **A** Amendment **/0**
Issue Date: **March 1995. WWW: 5/4/95**

Am29202 MICROCONTROLLER BLOCK DIAGRAM



CUSTOMER SERVICE

AMD's customer service network includes U.S. offices, international offices, and a customer training center. Expert technical assistance is available from AMD's worldwide staff of field application engineers and factory support staff.

Hotline, E-mail, and Bulletin Board Support

For answers to technical questions, AMD provides a toll-free number for direct access to our engineering support staff. For overseas customers, the easiest way to reach the engineering support staff with your questions is via fax with a short description of your question. AMD 29K Family customers also receive technical support through electronic mail. This worldwide service is available to 29K Family product users via the international Internet e-mail service. Also available is the AMD bulletin board service, which provides the latest 29K Family product information, including technical information and data on upcoming product releases.

Engineering Support Staff

(800) 292-9263, ext. 2	toll-free for U.S.
0031-11-1163	toll-free for Japan
(512) 602-4118	direct dial worldwide
44-(0)256-811101	U.K. and Europe hotline
(512) 602-5031	fax
epd.support@amd.com	e-mail

Bulletin Board

(800) 292-9263, ext. 1	toll-free for U.S.
(512) 602-7604	direct dial worldwide

Documentation and Literature

A simple phone call gets you free 29K Family information, such as data books, user's manuals, data sheets, application notes, the Fusion29K Partner Solutions Catalog and Newsletter, and other literature. Internationally, contact your local AMD sales office for complete 29K Family literature.

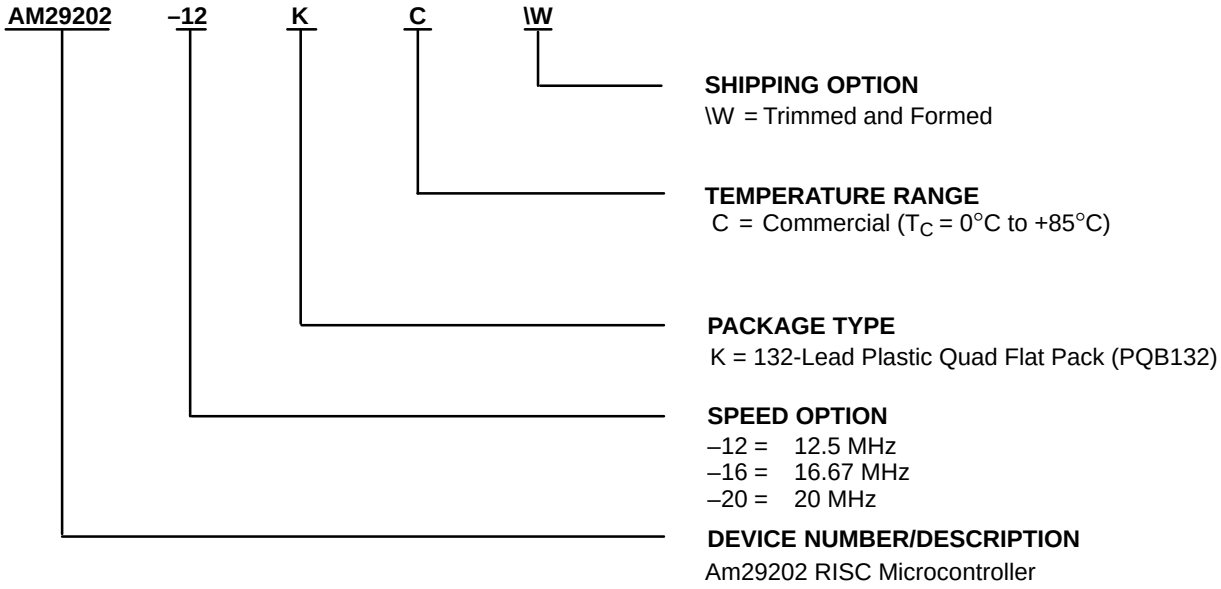
Literature Request

(800) 292-9263, ext. 3	toll-free for U.S.
(512) 602-5651	direct dial worldwide
(512) 602-7639	fax for U.S.
(800) 222-9323, option 1	AMD Facts-On-Demand™ fax information service toll-free for U.S.

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. Valid order numbers are formed by a combination of the elements below.



Valid Combinations	
AM29202–12	KC\W
AM29202–16	
AM29202–20	

Valid Combinations

Valid Combinations list configurations planned to be supported in volume. Consult the local AMD sales office to confirm availability of specific valid combinations, and check on newly released combinations.

Table 1. Product Comparison—Am29200 Microcontroller Family

FEATURE	Am29205 Controller	Am29202 Controller	Am29200 Controller	Am29245 Controller	Am29240 Controller	Am29243 Controller
Instruction Cache	—	—	—	4 Kbytes	4 Kbytes	4 Kbytes
Data Cache	—	—	—	—	2 Kbytes	2 Kbytes
Cache Associativity	—	—	—	2-way	2-way	2-way
Integer Multiplier	Software	Software	Software	Software	32 x 32-bit	32 x 32-bit
Memory Management Unit (MMU)	—	—	—	1 TLB 16 Entry	1 TLB 16 Entry	2 TLBs 32 Entry
Data Bus Width						
Internal	32 bits	32 bits	32 bits	32 bits	32 bits	32 bits
External	16 bits	32 bits	32 bits	32 bits	32 bits	32 bits
ROM Interface						
Banks	3	4	4	4	4	4
Width	8, 16 bits	8, 16, 32 bits	8, 16, 32 bits	8, 16, 32 bits	8, 16, 32 bits	8, 16, 32 bits
ROM Size (Max/Bank)	4 Mbytes	4 Mbytes	16 Mbytes	16 Mbytes	16 Mbytes	16 Mbytes
Boot-Up ROM Width	16 bits	8, 16, 32 bits	8, 16, 32 bits	8, 16, 32 bits	8, 16, 32 bits	8, 16, 32 bits
Burst-Mode Access	Not Supported	Not Supported	Supported	Supported	Supported	Supported
DRAM Interface						
Banks	4	4	4	4	4	4
Width	16 bits only	16, 32 bits	16, 32 bits	16, 32 bits	16, 32 bits	16, 32 bits
Size: 32-Bit Mode	—	16 Mbytes/bank	16 Mbytes/bank	16 Mbytes/bank	16 Mbytes/bank	16 Mbytes/bank
Size: 16-Bit Mode	8 Mbytes/bank	8 Mbytes/bank	8 Mbytes/bank	8 Mbytes/bank	8 Mbytes/bank	8 Mbytes/bank
Video DRAM	Not Supported	Not Supported	Supported	Supported	Supported	Not Supported
Access Cycles						
Initial/Burst	3/2	3/2	3/2	2/1	2/1	2/1
DRAM Parity	No	No	No	No	No	Yes
On-Chip DMA						
Width (ext. peripherals)	8, 16 bits	8, 16, 32 bits	8, 16, 32 bits	8, 16, 32 bits	8, 16, 32 bits	8, 16, 32 bits
Total Number of Channels	2	2	2	2	4	4
Externally Controlled	1	1	2	2	4	4
External Master Access	No	No	Yes	Yes	Yes	Yes
External Master Burst	No	No	No	Yes	Yes	Yes
External Terminate Signal	No	No	Yes	Yes	Yes	Yes
Double-Frequency CPU Option	No	No	No	No	Yes	Yes
Low Voltage Operation	No	No	No	Yes	Yes	Yes
Peripheral Interface Adapter (PIA)						
PIA Ports	2	2	6	6	6	6
Data Width	8, 16 bits	8, 16, 32 bits	8, 16, 32 bits	8, 16, 32 bits	8, 16, 32 bits	8, 16, 32 bits
Min. Cycles Access	3	3	3	1	1	1
Programmable I/O Port (PIO)						
Signals	8	12	16	16	16	16
Signals programmable for interrupt generation	8	8	8	8	8	8
Serial Ports						
Ports	1 Port	1 Port	1 Port	1 Port	2 Ports	2 Ports
$\overline{DSR}/\overline{DTR}$	PIO signals	PIO signals	Supported	Supported	1 Port Supported	1 Port Supported
Interrupt Controller						
External Interrupt Pins	2	2	4	4	4	4
External Trap and Warn Pins	0	0	3	3	3	3
Parallel Port Controller	Yes	Yes	Yes	Yes	Yes	Yes
32-Bit Transfer	No	Yes	Yes	Yes	Yes	Yes
IEEE-1284 Interface	No	Yes	No	No	No	No
JTAG Debug Support	No	Yes	Yes	Yes	Yes	Yes
Serializer/Deserializer	Yes	Yes	Yes	Yes	Yes	No
Pin Count and Package	100 PQFP	132 PQFP	168 PQFP	196 PQFP	196 PQFP	196 PQFP
Voltage						
V_{CC}	5 V	5 V	5 V	3.3 V or 5 V	3.3 V or 5 V	3.3 V or 5 V
I/O Tolerance	5 V	5 V	5 V	5 V	5 V	5 V
Processor Clock Rate	12, 16 MHz	12, 16, 20 MHz	16, 20 MHz	16 MHz	20, 25, 33 MHz	20, 25, 33 MHz

RELATED AMD PRODUCTS

29K Family Devices

Part No.	Description
Am29000®	32-bit RISC microprocessor
Am29005™	Low-cost 32-bit RISC microprocessor with no MMU and no branch target cache
Am29030™	32-bit RISC microprocessor with 8-Kbyte instruction cache
Am29035™	32-bit RISC microprocessor with 4-Kbyte instruction cache
Am29040™	32-bit RISC microprocessor with 8-Kbyte instruction cache and 4-Kbyte data cache
Am29050™	32-bit RISC microprocessor with on-chip floating point unit
Am29200™	32-bit RISC microcontroller
Am29205™	Low-cost 32-bit RISC microcontroller
Am29240™	32-bit RISC microcontroller with 4-Kbyte instruction cache and 2-Kbyte data cache
Am29243™	32-bit data RISC microcontroller with instruction and data caches and DRAM parity
Am29245™	Low-cost 32-bit RISC microcontroller with 4-Kbyte instruction cache

29K FAMILY DEVELOPMENT SUPPORT PRODUCTS

Contact your local AMD representative for information on the complete set of development support tools. The following software and hardware development products are available on several hosts:

- Optimizing compilers for common high-level languages
- Assembler and utility packages
- Source- and assembly-level software debuggers
- Target-resident development monitors
- Simulators
- Execution boards

THIRD-PARTY DEVELOPMENT SUPPORT PRODUCTS

The Fusion29K Program of Partnerships for Application Solutions provides the user with a vast array of products designed to meet critical time-to-market needs. Products and solutions available from the AMD Fusion29K Partners include

- Silicon products
- Software generation and debug tools
- Hardware development tools
- Board level products
- Laser printer solutions
- Networking and communication solutions
- Multiuser, kernel, and real-time operating systems
- Graphics solutions
- Manufacturing support
- Custom software consulting, support, and training

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

KEY FEATURES AND BENEFITS

The Am29202 microcontroller offers the performance of the Am29200 microcontroller with the slightly reduced feature set required for a smaller package. As an upgrade to the Am29205 microcontroller, the low-cost Am29202 microcontroller offers the enhanced performance of a 32-bit instruction/data bus and an IEEE-1284-compliant parallel interface.

IEEE-1284-Compliant Advanced Parallel Interface

The Am29202 microcontroller includes a new parallel port interface, called the Advanced Parallel Interface (API), that is compliant with IEEE Std 1284-1994. The IEEE-1284 standard specifies the operation of an extensible, bidirectional, multimode parallel interface that provides access to a variety of peripheral devices, such as printers, scanners, multifunction peripherals, storage devices, network interfaces, and others. This standard bidirectional protocol enables the development of new peripherals that can return significant data, as well as basic status, to the host.

AMD's implementation of this protocol on the Am29202 microcontroller supports a number of communications modes, allowing access to both high-speed and low-overhead communications. The supported modes include: Compatibility (standard Centronics) mode, Nibble (reverse) mode, Byte (reverse) mode, and ECP (bidirectional) mode.

The standard IEEE-1284 communications modes are supported using a mixture of hardware and software controls. Automatic hardware handshakes and hardware DMA support are provided in all modes except Nibble. Full software control provides easy access to input status information with a variety of software strategies, including polling, interrupt service, and DMA. The API supports peripheral-side designs only.

Windows Printing System Compatibility

Because of its high performance, full feature set, glueless interfaces, and low total system cost, the Am29202 microcontroller was chosen by Microsoft to be the reference hardware design for its Windows Printing System. The Windows Printing System provides substantial performance improvements for a new class of printers that are optimized for the Windows operating system.

These new printers utilize features of the IEEE-1284 parallel interface to provide a fast, bidirectional communication channel that improves the transfer of data between host and peripheral and also allows the printer to communicate status information back to the host PC. While not limited in functionality to a specific application, the Am29202 microcontroller has the performance and feature set ideally suited to meet the needs of these low- to mid-range laser printers.

Complete Set of Common System Peripherals

The Am29202 microcontroller minimizes system cost by incorporating a complete set of system facilities commonly found in embedded applications, eliminating the cost of additional components. The on-chip functions include: a ROM controller, a DRAM controller, a peripheral interface adapter, a DMA controller, a bidirectional serializer/deserializer, a programmable I/O port, an IEEE-1284-compliant parallel port interface, a serial port, an interrupt controller, and an IEEE-1149.1-compliant JTAG debug port.

The Am29202 microcontroller provides a glueless attachment to external ROMs, DRAMs, and other peripheral components. Processor outputs have edge-rate control that allows them to drive a wide range of load capacitances with low noise and ringing. This eliminates the cost of external logic and buffering.

The Am29202 microcontroller lets product designers capitalize on the very low system cost made possible by the integration of processor and peripherals. Many simple systems can be built using only the Am29202 microcontroller and external ROM and/or DRAM memory.

ROM Controller

The ROM controller supports four individual banks of ROM or other static memory. Each ROM bank has its own timing characteristics, and each bank may be of a different size: either 8, 16, or 32 bits wide. The ROM banks can appear as a contiguous memory area of up to 16 Mbytes in size. The ROM controller also supports writes to the ROM memory space for devices such as Flash EPROMs and SRAMs.

DRAM Controller

The DRAM controller supports four separate banks of dynamic memory, each of which can be a different size: either 16 or 32 bits wide. The DRAM banks can appear as a contiguous memory area of up to 64 Mbytes in size. To support system functions such as on-the-fly data compression and decompression, four 64-Kbyte regions of the DRAM can be mapped into a 16-Mbyte virtual address space.

Peripheral Interface Adapter (PIA)

The peripheral interface adapter allows for additional system features implemented by external peripheral chips. The PIA interface permits glueless interfacing from the Am29202 microcontroller to two external peripherals, each with a separate 4-Mbyte address space.

DMA Controller

The DMA controller in the Am29202 microcontroller provides two channels for transfer of data between the DRAM and internal peripherals and one channel for external transfers. One of the DMA channels is double buffered to relax the constraints on the reload time.

Interrupt Controller

The interrupt controller generates and reports the status of interrupts caused by on-chip peripherals.

Programmable I/O Port (PIO)

The Am29202 microcontroller's I/O port permits direct access to 12 individually programmable external input/output signals. Eight of these signals can be configured to cause interrupts. Four of these signals are shared with the IEEE-1284-compliant parallel port interface.

Serial Port

The serial port implements a full-duplex UART.

Serializer/Deserializer

The bidirectional bit serializer/deserializer (video interface) permits direct connection to a number of laser marking engines, video displays, or raster input devices such as scanners.

Performance Overview

The Am29202 microcontroller offers a significant margin of performance over CISC microprocessors in existing embedded designs, since the majority of processor features were defined for the maximum achievable performance at a very low cost. This section describes the features of the Am29202 microcontroller from the point of view of system performance.

Instruction Timing

The Am29202 microcontroller uses an arithmetic/logic unit, a field shift unit, and a prioritizer to execute most instructions. Each of these is organized to operate on 32-bit operands and provide a 32-bit result. All operations are performed in a single cycle.

The performance degradation of load and store operations is minimized in the Am29202 microcontroller by overlapping them with instruction execution, by taking advantage of pipelining, and by organizing the flow of external data into the processor so that the impact of external accesses is minimized.

Pipelining

Instruction operations are overlapped with instruction fetch, instruction decode and operand fetch, instruction execution, and result write-back to the register file. Pipeline forwarding logic detects pipeline dependencies and routes data as required, avoiding delays that might arise from these dependencies.

Pipeline interlocks are implemented by processor hardware. Except for a few special cases, it is not necessary to rearrange instructions to avoid pipeline dependencies, although this is sometimes desirable for performance.

Instruction Set Overview

The Am29202 microcontroller employs a three-address instruction set architecture. The compiler or assembly-language programmer is given complete freedom to allocate register usage. There are 192 general-purpose registers, allowing the retention of intermediate calculations and avoiding needless memory accesses. Instruction operands may be contained in any of the general-purpose registers, and the results may be stored into any of the general-purpose registers.

The instruction set contains 117 instructions that are divided into nine classes. These classes are integer arithmetic, compare, logical, shift, data movement, constant, floating point, branch, and miscellaneous. The floating-point instructions are not executed directly, but are emulated by trap handlers.

All directly implemented instructions are capable of executing in one processor cycle, with the exception of interrupt returns, loads, and stores.

Data Formats

The Am29202 microcontroller defines a word as 32 bits of data, a half-word as 16 bits, and a byte as 8 bits. The hardware provides direct support for word-integer (signed and unsigned), word-logical, word-Boolean, half-word integer (signed and unsigned), and character data (signed and unsigned).

Word-Boolean data is based on the value contained in the most significant bit of the word. The values TRUE and FALSE are represented by the MSB values 1 and 0, respectively.

Other data formats, such as character strings, are supported by instruction sequences. Floating-point formats (single and double precision) are defined for the processor; however, there is no direct hardware support for these formats in the Am29202 microcontroller.

Protection

The Am29202 microcontroller offers two mutually exclusive modes of execution, the user and supervisor modes, that restrict or permit accesses to certain processor registers and external storage locations.

The register file may be configured to restrict accesses to supervisor-mode programs on a bank-by-bank basis.

Page-Mode Memories

The Am29202 microcontroller uses the page-mode capability of common DRAMs to improve the access time in cases where page-mode accesses can be used. This is particularly useful in very low-cost systems with 16-bit-wide DRAMs, where the DRAM must be accessed twice for each 32-bit operand.

DRAM Mapping

The Am29202 microcontroller provides a 16-Mbyte region of virtual memory that is mapped to one of four 64-Kbyte blocks in the physical DRAM memory. This supports system functions such as on-the-fly data compression and decompression, allowing a large data structure such as a frame buffer to be stored in a compressed format while the application software operates on a region of the structure that is decompressed. Using a mechanism that is analogous to demand paging, system software moves data between the compressed and decompressed formats in a way that is invisible to the application software. This feature can greatly reduce the amount of memory required for printing, imaging, and graphics applications.

Interrupts and Traps

When the microcontroller takes an interrupt or trap, it does not automatically save its current state information in memory. This lightweight interrupt and trap facility greatly improves the performance of temporary interruptions such as simple operating-system calls that require no saving of state information.

In cases where the processor state must be saved, the saving and restoring of state information is under the control of software. The methods and data structures used to handle interrupts—and the amount of state information saved—may be tailored to the needs of a particular system.

Interrupts and traps are dispatched through a 256-entry vector table which directs the processor to a routine that handles a given interrupt or trap. The vector table may be relocated in memory by the modification of a processor register. There may be multiple vector tables in the system, though only one is active at any given time.

The vector table is a table of pointers to the interrupt and trap handlers and requires only 1 Kbyte of memory. The processor performs a vector fetch every time an interrupt or trap is taken. The vector fetch requires at least three cycles, in addition to the number of cycles required for the basic memory access.

Debugging and Testing

Software debugging on the Am29202 microcontroller is facilitated by the instruction trace facility and instruction breakpoints. Instruction tracing is accomplished by forcing the processor to trap after each instruction has been executed. Instruction breakpoints are implemented by the HALT instruction or by a software trap.

A scan interface compliant with IEEE Std 1149.1-1990 (JTAG) Standard Test Access Port and Boundary-Scan Architecture is provided to test system hardware in a production environment. It contains extensions that allow a hardware-development system to control and observe the processor without interposing hardware between the processor and system.

Complete Development and Support Environment

A complete development and support environment is vital for reducing a product's time-to-market. Advanced Micro Devices has created a standard development environment for the 29K Family of processors. In addition, the Fusion29K third-party support organization provides the most comprehensive customer/partner program in the embedded processor market.

Advanced Micro Devices offers a complete set of hardware and software tools for design, integration, debugging, and benchmarking. These tools, which are available now for the 29K Family, include the following:

- High C® 29K optimizing C compiler with assembler, linker, ANSI library functions, and 29K architectural simulator
- XRAY29K™ source-level debugger
- MiniMON29K™ debug monitor
- A complete family of demonstration and development boards

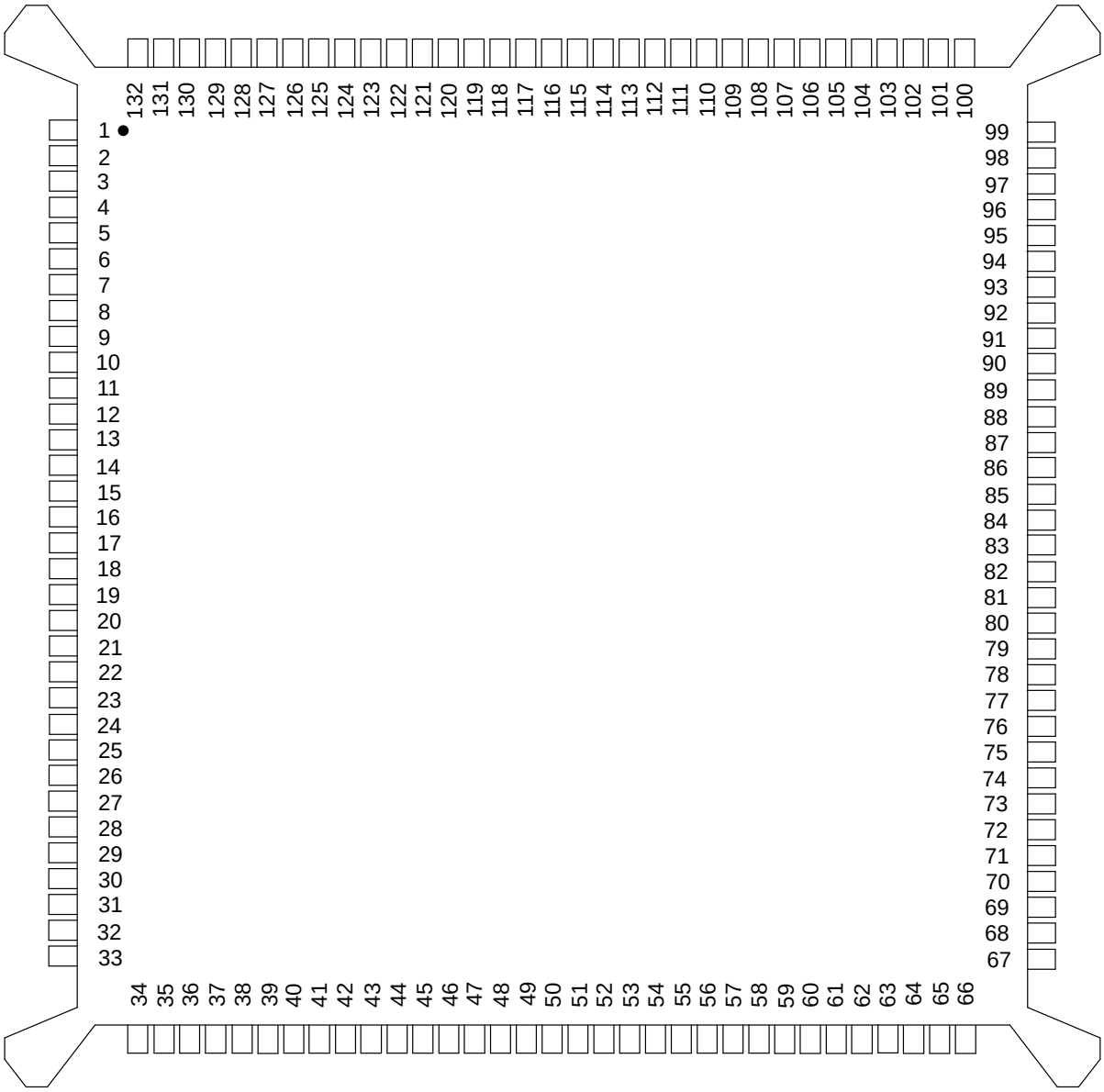
In addition, Advanced Micro Devices has developed a standard host interface (HIF) specification for operating system services, the Universal Debug Interface (UDI) for seamless connection of debuggers to ICEs and target hardware, and extensions for the UNIX common object file format (COFF).

This support is augmented by an engineering hotline, an on-line bulletin board, and field application engineers.

CONNECTION DIAGRAM

132-Lead Plastic Quad Flat Pack

Top View



Note:
Pin 1 is marked for orientation.

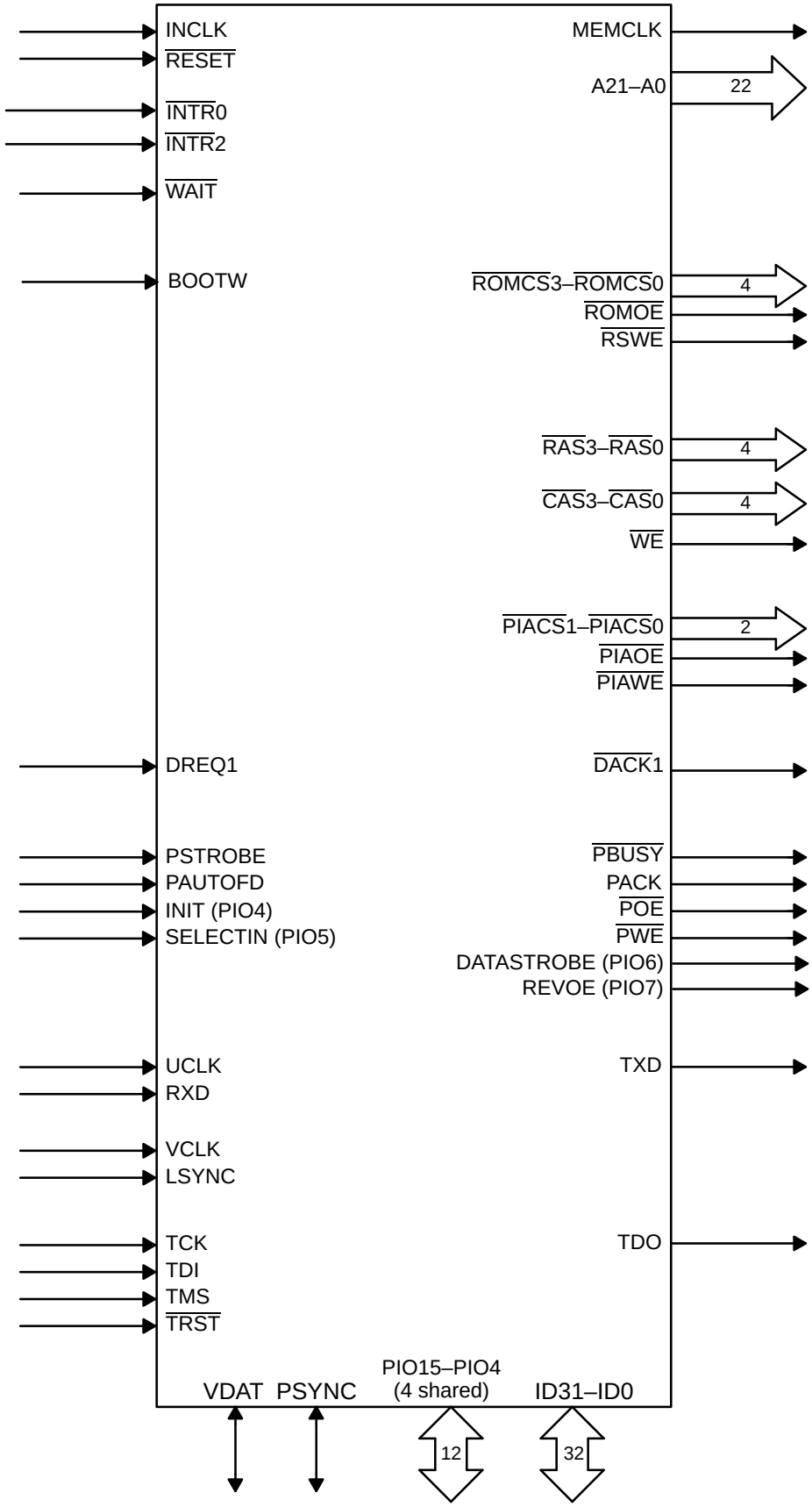
PQFP PIN DESIGNATIONS (Sorted by Pin Number)

Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
1	V_{CC}	**34**	ID7	**67**	$\overline{\text{PIAOE}}$	**100**	$\overline{\text{POE}}$
2	MEMCLK	**35**	ID6	**68**	$\overline{\text{DACK}}$1	**101**	PACK
3	V_{CC}	**36**	ID5	**69**	A21	**102**	$\overline{\text{PBUSY}}$
4	GND	**37**	ID4	**70**	A20	**103**	PIO15
5	INCLK	**38**	ID3	**71**	A19	**104**	PIO14
6	GND	**39**	ID2	**72**	A18	**105**	PIO13
7	V_{CC}	**40**	ID1	**73**	A17	**106**	GND
8	ID31	**41**	ID0	**74**	A16	**107**	V_{CC}
9	ID30	**42**	GND	**75**	A15	**108**	PIO12
10	ID29	**43**	V_{CC}	**76**	A14	**109**	PIO11
11	ID28	**44**	RXD	**77**	A13	**110**	PIO10
12	ID27	**45**	UCLK	**78**	A12	**111**	PIO9
13	ID26	**46**	TXD	**79**	A11	**112**	PIO8
14	ID25	**47**	$\overline{\text{ROMCS}}$3	**80**	GND	**113**	PIO7/REVOE
15	ID24	**48**	$\overline{\text{ROMCS}}$2	**81**	V_{CC}	**114**	PIO6/DATASTROBE
16	ID23	**49**	$\overline{\text{ROMCS}}$1	**82**	A10	**115**	PIO5/SELECTIN
17	ID22	**50**	$\overline{\text{ROMCS}}$0	**83**	A9	**116**	PIO4/INIT
18	ID21	**51**	$\overline{\text{RSWE}}$	**84**	A8	**117**	TDO
19	ID20	**52**	$\overline{\text{ROMOE}}$	**85**	A7	**118**	VDAT
20	ID19	**53**	$\overline{\text{RAS}}$3	**86**	A6	**119**	V_{CC}
21	ID18	**54**	$\overline{\text{RAS}}$2	**87**	A5	**120**	GND
22	ID17	**55**	$\overline{\text{RAS}}$1	**88**	A4	**121**	PSYNC
23	ID16	**56**	$\overline{\text{RAS}}$0	**89**	A3	**122**	DREQ1
24	V_{CC}	**57**	$\overline{\text{CAS}}$3	**90**	A2	**123**	$\overline{\text{INTR}}$0
25	GND	**58**	$\overline{\text{CAS}}$2	**91**	A1	**124**	$\overline{\text{INTR}}$2
26	ID15	**59**	$\overline{\text{CAS}}$1	**92**	A0	**125**	VCLK
27	ID14	**60**	$\overline{\text{CAS}}$0	**93**	V_{CC}	**126**	LSYNC
28	ID13	**61**	$\overline{\text{WE}}$	**94**	GND	**127**	TMS
29	ID12	**62**	V_{CC}	**95**	BOOTW	**128**	$\overline{\text{TRST}}$
30	ID11	**63**	GND	**96**	$\overline{\text{WAIT}}$/$\overline{\text{TRIST}}$	**129**	TCK
31	ID10	**64**	$\overline{\text{PIACS}}$1	**97**	PAUTOFD	**130**	TDI
32	ID9	**65**	$\overline{\text{PIACS}}$0	**98**	PSTROBE	**131**	$\overline{\text{RESET}}$
33	ID8	**66**	$\overline{\text{PIAWE}}$	**99**	$\overline{\text{PWE}}$	**132**	GND

PQFP PIN DESIGNATIONS (Sorted by Pin Name)

Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.
A0	92	GND	63	ID27	12	$\overline{\text{RAS0}}$	56
A1	91	GND	80	ID28	11	$\overline{\text{RAS1}}$	55
A2	90	GND	94	ID29	10	$\overline{\text{RAS2}}$	54
A3	89	GND	106	ID30	9	$\overline{\text{RAS3}}$	53
A4	88	GND	120	ID31	8	$\overline{\text{RESET}}$	131
A5	87	GND	132	INCLK	5	$\overline{\text{ROMCS0}}$	50
A6	86	ID0	41	$\overline{\text{INTR0}}$	123	$\overline{\text{ROMCS1}}$	49
A7	85	ID1	40	$\overline{\text{INTR2}}$	124	$\overline{\text{ROMCS2}}$	48
A8	84	ID2	39	LSYNC	126	$\overline{\text{ROMCS3}}$	47
A9	83	ID3	38	MEMCLK	2	$\overline{\text{ROMOE}}$	52
A10	82	ID4	37	$\overline{\text{PACK}}$	101	$\overline{\text{RSWE}}$	51
A11	79	ID5	36	PAUTOFD	97	RXD	44
A12	78	ID6	35	$\overline{\text{PBUSY}}$	102	TCK	129
A13	77	ID7	34	$\overline{\text{PIACS0}}$	65	TDI	130
A14	76	ID8	33	$\overline{\text{PIACS1}}$	64	TDO	117
A15	75	ID9	32	$\overline{\text{PIAOE}}$	67	TMS	127
A16	74	ID10	31	$\overline{\text{PIAWE}}$	66	$\overline{\text{TRST}}$	128
A17	73	ID11	30	PIO4/INIT	116	TXD	46
A18	72	ID12	29	PIO5/SELECTIN	115	UCLK	45
A19	71	ID13	28	PIO6/DATASTROBE	114	V_{CC}	1
A20	70	ID14	27	PIO7/REVOE	113	V_{CC}	3
A21	69	ID15	26	PIO8	112	V_{CC}	7
BOOTW	95	ID16	23	PIO9	111	V_{CC}	24
$\overline{\text{CAS0}}$	60	ID17	22	PIO10	110	V_{CC}	43
$\overline{\text{CAS1}}$	59	ID18	21	PIO11	109	V_{CC}	62
$\overline{\text{CAS2}}$	58	ID19	20	PIO12	108	V_{CC}	81
$\overline{\text{CAS3}}$	57	ID20	19	PIO13	105	V_{CC}	93
$\overline{\text{DACK1}}$	68	ID21	18	PIO14	104	V_{CC}	107
DREQ1	122	ID22	17	PIO15	103	V_{CC}	119
GND	4	ID23	16	$\overline{\text{POE}}$	100	VCLK	125
GND	6	ID24	15	PSTROBE	98	VDAT	118
GND	25	ID25	14	PSYNC	121	$\overline{\text{WAIT}}/\overline{\text{TRIST}}$	96
GND	42	ID26	13	$\overline{\text{PWE}}$	99	$\overline{\text{WE}}$	61

LOGIC SYMBOL



PIN DESCRIPTIONS

Note: *The UCLK signal must be tied High if the serial port is not used.*

Clocks

INCLK
Input Clock (input)

This is an oscillator input at twice the processor and system operating frequency. It can be driven at TTL levels.

MEMCLK
Memory Clock (output)

This is a clock output at one-half of the frequency of INCLK. Most processor outputs, and many inputs, are synchronous to MEMCLK. MEMCLK drives out with CMOS levels.

Processor Signals

A21–A0
Address Bus (output, synchronous)

The address bus supplies the byte address for all accesses, except for DRAM accesses. For DRAM accesses, multiplexed row and column addresses are provided on A14–A1. The signals A23–A22 and burst-mode devices are not supported on the Am29202 microcontroller.

ID31–ID0
Instruction/Data Bus (bidirectional, synchronous)

The instruction/data bus (ID bus) transfers instructions to, and data to and from the processor.

$\overline{\text{INTR}}$2, $\overline{\text{INTR}}$0
Interrupt Requests 2 and 0 (input, asynchronous, internal pull-up transistors)

These inputs generate prioritized interrupt requests. The interrupt caused by $\overline{\text{INTR}}$0 has the highest priority, and the interrupt caused by $\overline{\text{INTR}}$2 has the lower priority. The interrupt requests are masked in prioritized order by the Interrupt Mask field in the Current Processor Status Register and are disabled by the DA and DI bits of the Current Processor Status Register. These signals have special hardening against metastable states, allowing them to be driven with slow-transition-time signals. The $\overline{\text{INTR}}$3 and $\overline{\text{INTR}}$1 signals are not supported on the Am29202 microcontroller.

$\overline{\text{RESET}}$
Reset (input, asynchronous)

This input places the processor in the Reset mode. This signal has special hardening against metastable states, allowing it to be driven with a slow-rise-time signal.

$\overline{\text{WAIT}}$/$\overline{\text{TRIST}}$
Add Wait States/Three-State Control (input, synchronous, weak internal pull-up)

The $\overline{\text{WAIT}}$ signal may be asserted during a PIA, ROM, or DMA access to extend the access indefinitely. The $\overline{\text{WAIT}}$/$\overline{\text{TRIST}}$ pin is also used for three-state control during test. When asserted during a processor reset, all output pins go into a high impedance state. For normal operation, this pin must be pulled High during reset.

ROM Interface

BOOTW
Boot ROM Width (input, asynchronous)

This input configures the width of ROM Bank 0, so the ROM can be accessed before the ROM configuration has been set by the system initialization software. The BOOTW signal is sampled during and after a processor reset. If BOOTW is High before and after reset (tied High), the boot ROM is 32 bits wide. If BOOTW is Low before and after reset (tied Low), the boot ROM is 16 bits wide. If BOOTW is Low before reset and High after reset (tied to $\overline{\text{RESET}}$), the boot ROM is 8 bits wide. This signal has special hardening against metastable states, allowing it to be driven with a slow-rise-time signal and permitting it to be tied to $\overline{\text{RESET}}$.

$\overline{\text{ROMCS}}$3–$\overline{\text{ROMCS}}$0
ROM Chip Selects, Banks 3–0 (output, synchronous)

A Low level on one of these signals selects the memory devices in the corresponding ROM bank. $\overline{\text{ROMCS}}$3 selects devices in ROM Bank 3, and so on. The timing and access parameters of each bank are individually programmable.

$\overline{\text{ROMOE}}$
ROM Output Enable (output, synchronous)

This signal enables the selected ROM Bank to drive the ID bus. It is used to prevent bus contention when switching between different ROM banks or switching between a ROM bank and another device or DRAM bank.

$\overline{\text{RSWE}}$
ROM Space Write Enable (output, synchronous)

This signal is used to write an alterable memory in a ROM bank (such as an SRAM or Flash EPROM). $\overline{\text{RSWE}}$ supports only writes of width equal to or greater than the width of the memory, and the memory must be at least 16 bits wide. The $\overline{\text{CAS}}$3–$\overline{\text{CAS}}$0 signals can serve as individual byte strobes for writes to the ROM space, if ROM byte writes are enabled.

DRAM Interface

$\overline{\text{CAS}}$3–$\overline{\text{CAS}}$0
Column Address Strobes, Byte 3–0 (output, synchronous)

A High-to-Low transition on these signals causes the DRAM bank selected by $\overline{\text{RAS}}$3–$\overline{\text{RAS}}$0 to latch the column address and complete the access. To support byte and half-word writes, column address strobes are provided for individual DRAM bytes. $\overline{\text{CAS}}$3 is the column address strobe for the DRAMs, in all banks, attached to

ID31–ID24. $\overline{CAS}$2 is for the DRAMs attached to ID23–ID16, and so on. These signals are also used in other special DRAM cycles.

The $\overline{CAS}$3–$\overline{CAS}$0 signals can be enabled to act as individual byte strobes for byte writes to the ROM space. In this configuration, ROM accesses do not conflict with DRAM accesses or refresh even though $\overline{CAS}$3–$\overline{CAS}$0 may be used by both the ROM and DRAM.

$\overline{RAS}$3–$\overline{RAS}$0
Row Address Strobe, Banks 3–0 (output, synchronous)

A High-to-Low transition on one of these signals causes a DRAM in the corresponding bank to latch the row address and begin an access. $\overline{RAS}$3 starts an access in DRAM Bank 3, and so on. These signals are also used in other special DRAM cycles.

$\overline{WE}$
Write Enable (output, synchronous)

This signal is used to write the selected DRAM bank. "Early write" cycles are used so the DRAM data inputs and outputs can be tied to the common ID bus.

Peripheral Interface Adapter (PIA)

$\overline{PIACS}$1–$\overline{PIACS}$0
Peripheral Chip Selects, Regions 1–0 (output, synchronous)

These signals are used to select individual peripheral devices. DMA Channel 1 may be programmed to use $\overline{PIACS}$1. $\overline{PIACS}$5–$\overline{PIACS}$2 are not supported on the Am29202 microcontroller.

$\overline{PIAOE}$
Peripheral Output Enable (output, synchronous)

This signal enables the selected peripheral device to drive the ID bus.

$\overline{PIAWE}$
Peripheral Write Enable (output, synchronous)

This signal causes data on the ID bus to be written into the selected peripheral.

DMA Controller

$\overline{DACK}$1
DMA Acknowledge, Channel 1 (output, synchronous)

This signal acknowledges an external transfer on DMA Channel 1. DMA transfers can occur to and from internal peripherals independent of these acknowledgments. $\overline{DACK}$0 is not supported on the Am29202 microcontroller.

DREQ1
DMA Request, Channel 1 (input, asynchronous, internal pull-up)

This signal requests an external transfer on DMA Channel 1). This request is individually programmable to be either level- or edge-sensitive for either polarity of level or edge. DMA transfers can occur to and from internal peripherals independent of these requests. DREQ0 is not supported on the Am29202 microcontroller.

I/O Port

PIO15–PIO8, PIO7/REVOE, PIO6/DATASTROBE, PIO5/SELECTIN, PIO4/INIT
Programmable Input/Output (input/output, asynchronous)

The PIO signals are available for direct software control and inspection. PIO15–PIO8 may be individually programmed to cause processor interrupts. These signals have special hardening against metastable states, allowing them to be driven with slow-transition-time signals. PIO7–PIO4 are shared with the IEEE-1284-compliant parallel port interface. The I/O port has control of these lines when the parallel port is not enabled. The signals PIO3–PIO0 are not supported on the Am29202 microcontroller.

Advanced Parallel Interface (API)

Note: *For more complete descriptions of these signals and their use, see the functional description of the IEEE-1284-compliant parallel interface beginning on page 41.*

DATASTROBE/PIO6
(output, synchronous)

DATASTROBE causes incoming data from the host to be latched externally on the rising edge. It is generated by a number of different signals and edges in various IEEE-1284 modes.

INIT/PIO4
(input, asynchronous)

The INIT signal comes from the IEEE-1284 signal nInit/nReverseRequest and can optionally cause control interrupts on either edge.

PACK
Parallel Port Acknowledge (output, synchronous)

This signal is used by the microcontroller to acknowledge a transfer from the host or to indicate to the host that data has been placed on the port.

PAUTOFD
Parallel Port Autofeed (input, asynchronous)

This signal is directly input from the IEEE-1284 signal nAutofd/HostBusy/HostAck. It is used by the host in reverse-channel modes to signal reverse data strobe. It is also used in other contexts in various modes. PAUTOFD can optionally cause control interrupts on either edge.

$\overline{PBUSY}$
Parallel Port Busy (output, synchronous)

Output to the IEEE-1284 signal Busy/PtrBusy/PeriphAck, this signal comes from the API when it is enabled.

$\overline{\text{POE}}$
Parallel Port Output Enable (output, synchronous)

This signal enables latched data on the data bus, to be read by the processor under interrupt or DMA control.

PSTROBE
Parallel Port Strobe (input, asynchronous)

Directly input from the IEEE-1284 signal nStrobe/ HostClk, PSTROBE is used in some forward modes to generate data strobe assertions and to signal data presence. In other modes, PSTROBE signals something other than a data transfer. The PSTROBE signal can optionally cause control interrupts on either edge.

$\overline{\text{PWE}}$
Parallel Port Write Enable (output, synchronous)

This signal is used to latch the data bus for outgoing (peripheral-to-host) transmission.

REVOE/PIO7
(output, synchronous)

REVOE is used to drive latched output data in the reverse direction, from peripheral to host.

SELECTIN/PIO5
(input, asynchronous)

SELECTIN comes from the IEEE-1284 signal nSelectIn/1284Active. It transitions (along with PAUTOFD) to signal the request to negotiate an IEEE-1284 mode and to signal the termination from an IEEE-1284 mode. SELECTIN can optionally cause control interrupts on either edge.

Serial Port

UCLK
UART Clock (input)

This is an oscillator input for generating the UART (serial port) clock. To generate the UART clock, the oscillator frequency may be divided by any amount up to 65,536. The UART clock operates at 16 times the serial port's baud rate. As an option, UCLK may be driven with MEMCLK or INCLK. It can be driven with TTL levels. UCLK must be tied High if unused.

TXD
Transmit Data (output, asynchronous)

This output is used to transmit serial data.

RXD
Receive Data (input, asynchronous)

This input is used to receive serial data.

Video Interface

VCLK
Video Clock (input, asynchronous)

This clock is used to synchronize the transfer of video data. As an option, VCLK may be driven with MEMCLK or INCLK. It can be driven with TTL levels.

VDAT
Video Data (input/output, synchronous to VCLK)

This is serial data to or from the video device.

LSYNC
Line Synchronization (input, asynchronous)

This signal indicates the start of a raster line.

PSYNC
Page Synchronization (input/output, asynchronous)

This signal indicates the beginning of a raster page.

JTAG 1149.1 Boundary Scan Interface

TCK
Test Clock Input (asynchronous input, internal pull-up)

This input is used to operate the test access port. The state of the test access port must be held if this clock is held either High or Low. This clock is internally synchronized to MEMCLK for certain operations of the test access port controller, so signals internally driven and sampled by the test access port are synchronous to processor internal clocks.

TMS
Test Mode Select (input, synchronous to TCK, internal pull-up)

This input is used to control the test access port. If it is not driven, it appears High internally.

TDI
Test Data Input (input, synchronous to TCK, internal pull-up)

This input supplies data to the test logic from an external source. It is sampled on the rising edge of TCK. If it is not driven, it appears High internally.

TDO
Test Data Output (three-state output, synchronous to TCK)

This output supplies data from the test logic to an external destination. It changes on the falling edge of TCK. It is in the high-impedance state except when scanning is in progress.

$\overline{\text{TRST}}$
Test Reset Input (asynchronous input, internal pull-up)

This input asynchronously resets the test access port. This input places the test logic in a state such that no output driver is enabled. The $\overline{\text{TRST}}$ input must be asserted in conjunction with the $\overline{\text{RESET}}$ input for correct processor initialization, whether or not the JTAG port is used.

FUNCTIONAL DIFFERENCES

The Am29202 microcontroller is functionally very similar to the Am29200 microcontroller, operating with a reduced pin count and fewer peripherals. The major differences include a new IEEE-1284-compliant bidirectional parallel port interface, a new refresh scheme, a smaller address bus (22 bits), only one external DMA channel, no direct DMA, no video DRAM support, fewer PIOs, fewer PIAs, no burst-mode ROM, no external traps, fewer interrupt request pins, and a new JTAG scan path.

This large section (through page 70) describes the technical differences between the Am29202 and Am29200 microcontrollers and omits much of the information common to both processors. For a complete description of the technical features, on-chip peripherals, programming interface, and instruction set, please refer to the *Am29200 and Am29205 RISC Microcontrollers User's Manual* (order# 16362).

***Note**: All registers with bits designated as "reserved" should be programmed with 0s to ensure compatibility.*

Advanced Parallel Interface

The parallel interface on the Am29202 microcontroller is completely different from the one included on the Am29200 and Am29205 microcontrollers. The new port is called the Advanced Parallel Interface (API) and is described in considerable detail in the section "IEEE-1284-Compliant Advanced Parallel Interface," beginning on page 41.

Memory Map Changes

All addresses are in the microcontroller's instruction/data memory address space. The address space is partitioned as shown in Table 2. Internal peripheral registers are selected by offsets from address 80000000h. The address assignment of the various internal peripherals and controllers is shown in Table 3.

The address assignments for the parallel port registers have changed from those assigned for the Am29200 and Am29205 microcontrollers. The following register addresses are not supported on the Am29202 microcontroller:

Parallel Port Control Register	800000C0
Parallel Port Data Register	800000C4
Parallel Port Status Register	800000C8

Accesses to addresses that are not supported on the Am29202 microcontroller will generate an Unsupported Peripheral Address trap (see Table 4). The new address assignments for the Advanced Parallel Interface (API) registers are shown in Table 3.

Pin Changes for the Am29202 Microcontroller

The reduced pin count of the Am29202 microcontroller comes from having a smaller address bus and fewer ports on some of the peripherals. The following signals supported on the Am29200 microcontroller are not available on the Am29202 microcontroller.

- Processor signals: A23–A22, R/$\overline{W}$, $\overline{WARN}$, $\overline{INTR}$1, $\overline{INTR}$3, $\overline{TRAP}$1–$\overline{TRAP}$0, STAT2–STAT0
- ROM interface signals: $\overline{BURST}$
- DRAM interface signals: $\overline{TR}$/$\overline{OE}$
- PIA signals: $\overline{PIACS}$5–$\overline{PIACS}$2
- DMA signals: DREQ0, $\overline{DACK}$0, TDMA, $\overline{GREQ}$, $\overline{GACK}$
- I/O port signals: PIO3–PIO0
- Serial port signals: $\overline{DSR}$, $\overline{DTR}$

Table 2. Internal Peripheral Address Ranges

Address Range (hexadeximal)	Selection	Maximum Physical Size
00000000–03FFFFFF	ROM Banks (all)	16 Mbyte
40000000–43FFFFFF	DRAM Banks (all)	64 Mbyte
50000000–50FFFFFF	Mapped DRAM Banks (all)	16 Mbyte
60000000–63FFFFFF	VDRAM transfers	Not Supported
80000000–800000FC	Internal peripherals/controllers	—
90000000–90FFFFFF	PIA Region 0 ($\overline{PIACS}$0)	4 Mbyte
91000000–91FFFFFF	PIA Region 1 ($\overline{PIACS}$1)	4 Mbyte
92000000–92FFFFFF	PIA Region 2 ($\overline{PIACS}$2)	Not Supported
93000000–93FFFFFF	PIA Region 3 ($\overline{PIACS}$3)	Not Supported
94000000–94FFFFFF	PIA Region 4 ($\overline{PIACS}$4)	Not Supported
95000000–95FFFFFF	PIA Region 5 ($\overline{PIACS}$5)	Not Supported
—all others—	Reserved	

Table 3. Internal Peripheral Address Assignments

Peripheral	Address (hexadecimal)	Register
ROM Controller	80000000	ROM Control Register
	80000004	ROM Configuration Register
DRAM Controller	80000008	DRAM Control Register
	8000000C	DRAM Configuration Register
DRAM Mapping Unit	80000010	DRAM Mapping Register 0
	80000014	DRAM Mapping Register 1
	80000018	DRAM Mapping Register 2
	8000001C	DRAM Mapping Register 3
Peripheral Interface Adapter	80000020	PIA Control Register 0
	80000024	PIA Control Register 1♦
Interrupt Controller	80000028	Interrupt Control Register
DMA Channel 0	80000030	DMA0 Control Register
	80000034	DMA0 Address Register
	80000070	DMA0 Address Tail Register
	80000038	DMA0 Count Register
	8000003C	DMA0 Count Tail Register
DMA Channel 1	80000040	DMA1 Control Register
	80000044	DMA1 Address Register
	80000048	DMA1 Count Register
Serial Port	80000080	Serial Port Control Register
	80000084	Serial Port Status Register
	80000088	Serial Port Transmit Holding Register
	8000008C	Serial Port Receive Buffer Register
	80000090	Baud Rate Divisor Register
Advanced Parallel Interface	800000A0	Advanced Parallel Control Register
	800000A4	Advanced Parallel Status Register
	800000A8	Advanced Parallel Interrupt Mask Register
	800000AC	Advanced Parallel Interrupt Status Register
	800000B0	Advanced Parallel Data Register
Programmable I/O Port	800000D0	PIO Control Register
	800000D4	PIO Input Register
	800000D8	PIO Output Register
	800000DC	PIO Output Enable Register
Video Interface	800000E0	Video Control Register
	800000E4	Top Margin Register
	800000E8	Side Margin Register
	800000EC	Video Data Holding Register
	—all others—	Reserved

Note:

♦ *PIA Control Register 1 is reserved on the Am29202 microcontroller.*

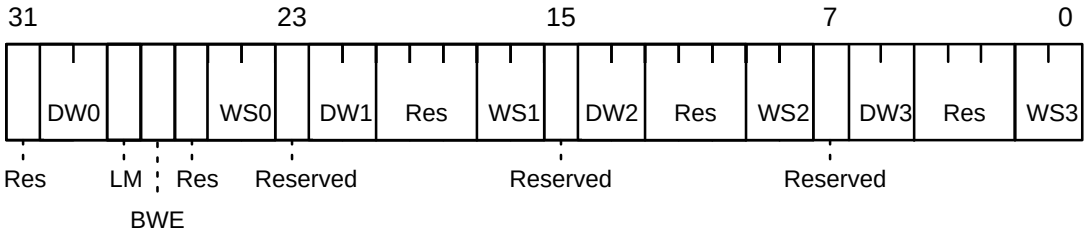


Figure 1. ROM Control Register

ROM CONTROLLER

The on-chip ROM controller provides a glueless interface to static memory devices such as ROM, EPROM, SRAM, Flash EPROM, and memory-mapped peripherals.

The ROM interface on the Am29202 microcontroller accommodates up to four banks of static memory space. These banks can be 8, 16, or 32 bits wide, with a maximum address space of 4 Mbytes per bank, instead of the 16 Mbytes supported by the Am29200 microcontroller.

Burst-mode ROM accesses are not supported on the Am29202 microcontroller, since the $\overline{\text{BURST}}$ pin is not present.

ROM Control Register (RMCT, Address 80000000)

The ROM Control Register (Figure 1) controls the access of ROM Banks 0 through 3. Bits controlling burst-mode on the Am29200 microcontroller are now reserved on the Am29202 microcontroller.

Bit 31: Reserved

Bits 30–29: Data Width, Bank 0 (DW0)—This field indicates the width of the ROM in Bank 0, as follows:

DW0	ROM Width
00	32 bits
01	8 bits
10	16 bits
11	Reserved

Bit 28: Large Memory (LM)—This bit controls the size of the ROM banks and the total size of the ROM address space. If the LM bit is 0, each ROM bank is up to 1 Mbyte in size, for placement within a 4 Mbyte total ROM address space. If the LM bit is 1, each ROM bank is up to 4 Mbytes in size, for placement within a 16-Mbyte total ROM address space.

Bit 27: Byte Write Enable (BWE)—This bit controls whether or not the $\overline{\text{CAS}}$3–$\overline{\text{CAS}}$0 signals are used as byte strobes during writes to the ROM address space. If BWE is 0, the $\overline{\text{CAS}}$3–$\overline{\text{CAS}}$0 signals are not active during ROM writes (unless there is a hidden refresh at the same time). If BWE is 1, the $\overline{\text{CAS}}$3–$\overline{\text{CAS}}$0 signals are used as byte strobes during a ROM write with hidden refresh prohibited during a ROM read or write.

Bit 26: Reserved

Bits 25–24: Wait States, Bank 0 (WS0)—This field specifies the number of wait states in a ROM access (i.e., the number of cycles in addition to one cycle required to access the ROM). Zero-wait-state cycles are supported for ROM reads. Writes to the ROM address space have a minimum of one wait state, even when wait states are programmed at zero.

Other bits of this register have a definition similar to DW0 and WS0 for ROM Banks 1 through 3.

DRAM CONTROLLER

The Am29202 microcontroller directly supports DRAM devices without any additional components, providing RAS and CAS generation, address multiplexing, and refresh generation. The on-chip DRAM controller utilizes page-mode accesses and CAS-before-RAS refresh to extract maximum performance from DRAM devices.

The DRAM interface accommodates up to four banks of DRAM that can be configured as a contiguous memory. Each bank is individually configurable in width. In addition, four 64-Kbyte regions of the DRAM can be mapped into a 16-Mbyte virtual address space.

For random accesses, the DRAM controller provides a fixed access time of 3 cycles plus 1 cycle of RAS precharge after each access. Sequential accesses use the DRAM page mode with 3 cycles for the first access, followed by 2 cycles for each additional access, followed by 1 cycle of precharge.

To support a lower pin count, several signals used by the Am29200 microcontroller for DRAM interfacing are not available on the Am29202 microcontroller.

The $\overline{TR}/\overline{OE}$ signal for normal DRAM output enable and video DRAM transfer is not available on the Am29202 microcontroller. Any DRAM with an $\overline{OE}$ line should have this line either tied to the appropriate $\overline{CAS}$ signal, or tied directly to ground (asserted) to always be enabled. This will not cause any circuit contention, since the DRAM's internal logic gates the external OE signal with the device's internal chip select from the processor's RAS. Video DRAM transfers are not supported on the Am29202 microcontroller.

DRAM Control Register (DRCT, Address 80000008)

The DRAM Control Register (Figure 2) controls the access to and refresh of DRAM Banks 0 through 3.

Bit 31: Page-Mode DRAM, Bank 0 (PG0)—When this bit is 1, burst-mode accesses to DRAM Bank 0 are performed using page-mode accesses for all but the first access. When this bit is 0, page-mode accesses are not performed.

Bit 30: Data Width, Bank 0 (DW0)—This field indicates the width of the DRAM in Bank 0, as follows:

DW Value	DRAM Width
0	32 bits
1	16 bits

Bit 29: Disable Bank Refresh, Bank 0 (DBR0)—When this bit is 1, DRAM refresh does not occur for DRAM Bank 0.

Bit 28: Large Memory (LM)—This bit controls the size of the DRAM banks and the total size of the DRAM address space. If the LM bit is 0, each DRAM bank is up to 4 Mbytes in size, for placement within a 16 Mbyte total DRAM address space.

If the LM bit is 1, each DRAM bank is up to 16 Mbytes in size, for placement within a 64-Mbyte total DRAM address space.

Other bits of this register have a definition similar to PG0, DBR0, DW0 for DRAM Banks 1 through 3.

Bit 15: Static-Column DRAM (SC)—When this bit is 1, page-mode accesses to the DRAM are performed using static-column accesses. Static column accesses differ from page-mode cycles only in that $\overline{CAS}3$–$\overline{CAS}0$ are held Low throughout a read access. The timing of the access is not affected, and write accesses are not affected. When this bit is 0, normal page-mode accesses are performed, if enabled.

Bits 14–9: Reserved

Bits 8–0: Refresh Rate (REFRATE)—This field indicates the number of MEMCLK cycles between DRAM refresh intervals. A DRAM refresh interval is the time required to refresh all enabled DRAM banks. CAS-before-RAS cycles are performed, overlapped in the background with other non-DRAM accesses when possible. If one or more banks have not been refreshed

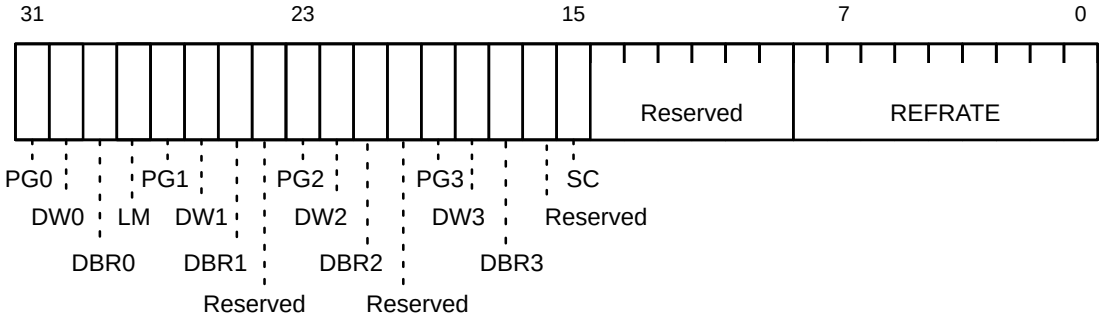


Figure 2. DRAM Control Register

in the background when the REFRATE interval expires twice, the processor forces a panic-mode refresh of the unrefreshed banks.

The minimum REFRATE count for the Am29202 microcontroller is 16. A zero in the REFRATE field disables refresh. On reset, this field is initialized to the value 1FFh.

Refresh Control Changes

Two basic improvements have been made to the refresh control mechanism over that implemented on the Am29200 microcontroller.

- Panic refresh has been redefined.
- Selectable DRAM refresh by bank has been added.

Panic Refresh

A panic refresh will occur when the REFRATE field in the DRAM Control Register has decremented to 0 twice. Rather than immediately refreshing all unrefreshed banks at the first available opportunity, these banks are refreshed in sequence, with each 4-cycle CAS-before-RAS refresh separated by 12 cycles from the next refresh. Having gaps between the refreshes allows DMA transfers to occur in between. The maximum interval between refreshes on the same (enabled) bank is:

((number of rows/bank) – 1) * REFRATE +
(16 MEMCLK cycles * number of enabled banks)

This assumes each DRAM bank can be refreshed within 16 cycles of the preceding bank's refresh.

Hidden or non-panic refreshes are unchanged from the Am29200 microcontroller.

Selectable DRAM Refresh

Since unused DRAM banks need no refresh, the Am29202 microcontroller provides a selectable DRAM bank refresh option. This feature eliminates unnecessary refresh cycles, reducing the likelihood of a panic refresh and speeding up the refresh process. Four new bits have been added to the DRAM Control Register to support this feature. A single bit is present for each DRAM bank and, when set high, disables that DRAM bank from being refreshed. All DRAM banks are enabled for DRAM refresh at processor reset.

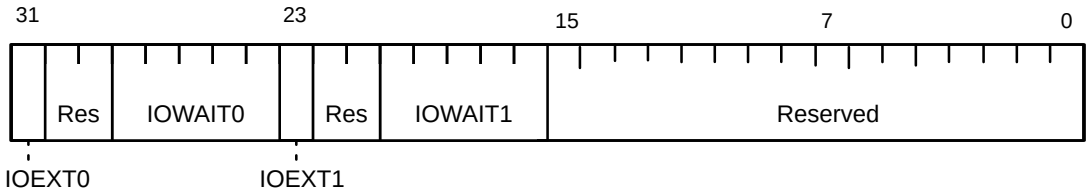


Figure 3. PIA Control Register 0

PERIPHERAL INTERFACE ADAPTER (PIA)

PIA space on the microcontroller is divided into regions, each of which can be directly attached to an off-chip peripheral device. The microcontroller's dedicated PIA chip select signals will assert a peripheral device's chip select pin when the associated PIA region on the microcontroller is read or written.

With two PIA chip select signals and a smaller address bus, the Am29202 microcontroller supports up to two peripheral devices, each with its own 22-bit memory space, for a maximum size of 4 Mbytes per PIA region. The $\overline{\text{PIACS}}$5–$\overline{\text{PIACS}}$2 signals are not supported on the Am29202 microcontroller.

PIA Control Register 0/1 (PICT0/1, Address 80000020/24)

The PIA Control Register 0 (Figure 3) controls the access to PIA Regions 0 and 1 on the Am29202 microcontroller. The PIA Control Register 1 is not available on the Am29202 microcontroller, since this product does not support PIA Regions 2, 3, 4, or 5.

Bit 31: Input/Output Extend, Region 0 (IOEXT0)—If this bit is one, the end of a PIA access is extended by one cycle after $\overline{\text{PIAOE}}$ is deasserted or by two cycles after $\overline{\text{PIAWE}}$ is deasserted. This provides one additional cycle of output disable time or data hold time for reads and writes, respectively.

Bits 30–29: Reserved

Bits 28–24: Input/Output Wait States, Region 0 (IOWAIT0)—This field specifies the number of wait states taken by an access to PIA Region 0. An I/O read cycle takes at least three cycles (two wait states), and an I/O write cycle takes at least four cycles (three wait states). If the IOWAIT0 field specifies an insufficient number of wait states for an access (for example, IOWAIT0 = 00010b for a write), the processor takes the required minimum number of wait states instead of the specified number.

Other bits perform similar functions to IOEXT0 and IOWAIT0 for PIA Region 1.

Bits 15–0: Reserved. These bits are reserved on the Am29202 microcontroller and should be written with 0s to ensure compatibility.

DMA CONTROLLER

The Am29202 microcontroller supports two types of DMA transfers: internal and external transfers. Direct DMA transfers between an external device and DRAM using an address supplied by the external device are not supported on the Am29202 microcontroller since the $\overline{\text{GREQ}}$ and $\overline{\text{GACK}}$ pins are not available on this device.

Internal DMA transfers can be requested by the parallel port, serial port, and video interface. Each of these internal peripherals has a field in its control register for specifying which of the two DMA channels is to be used for the transfer. The DMA-enable field for the IEEE-1284-compliant parallel port is DMAMODE in the Advanced Parallel Control (APCT) Register.

External DMA transfers are requested by off-chip peripherals using DREQ1.

DMA Channel 0 (DMA0) is available to internal peripherals only. DMA Channel 1 (DMA1) can be requested by either internal or external peripherals.

The Am29200 microcontroller signals DREQ0, $\overline{\text{DACK}}$0, $\overline{\text{GREQ}}$, $\overline{\text{GACK}}$, and TDMA are not supported on the Am29202 microcontroller.

DMA0 Control Register (DMCT0, Address 80000030)

The DMA0 Control Register (Figure 4) controls DMA Channel 0 on the Am29202 microcontroller. DMA Channel 0 on the Am29202 microcontroller is available for transfers between internal peripherals and DRAM only; external transfers are not supported.

Bits 31–24: Reserved

Bits 23–22: Data Width (DW)—This field indicates the width of the data transferred by the DMA channel, as follows:

DW Value	DMA Transfer Width
00	32 bits
01	8 bits
10	16 bits
11	32 bits, address unchanged

The value DW=11 is used to repeatedly transfer a fixed pattern from a single DRAM location to a peripheral. For example, it can be used with the video shifter to display a blank area of a printed page without requiring that a memory buffer be allocated for the blank area.

Bits 21–10: Reserved

Bit 9: Transfer Up/Down (UD)—This bit controls the addressing of memory for the series of DMA transfers. If the UD bit is 1, the DMA address (in the DMA0 Address Register) is incremented after each transfer. If the UD bit is 0, the DMA address is decremented after each transfer. The amount by which the address is incremented or decremented is determined by the width of the transfer.

DW Value	Address Increment/Decrement
00 (32 bits)	+/– 4
01 (8 bits)	+/– 1
10 (16 bits)	+/– 2
11 (32 bits)	+/– 0

Bit 8: Read/Write (RW)—This bit controls whether the DMA transfer is to or from the DRAM. If the RW bit is 1, the DMA channel transfers data from the DRAM to the peripheral. If the RW bit is 0, the DMA channel transfers data from the peripheral to the DRAM.

Bit 7: Enable (EN)—This bit enables the DMA channel to perform transfers. A 1 enables transfers, and a 0 disables transfers.

Bit 6: Reserved

Bit 5: Count Terminate Enable (CTE)—This bit, when 1, causes the DMA channel to terminate the transfer when the DMACNT field of the DMA Count Register decrements past zero. If this bit is 0, the DMA transfer does not terminate, though the DMA channel still decrements the count after every transfer.

Bit 4: Queue Enable (QEN)—This bit, when 1, enables the DMA queuing feature (which is implemented only on DMA Channel 0). DMA queuing allows the DMA0 Address Register and DMA0 Count Register to be reloaded automatically at the end of a DMA transfer from the DMA0 Address Tail Register and the DMA0

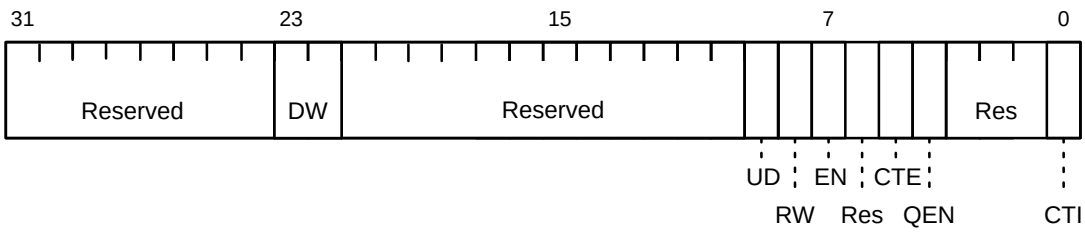


Figure 4. DMA0 Control Register

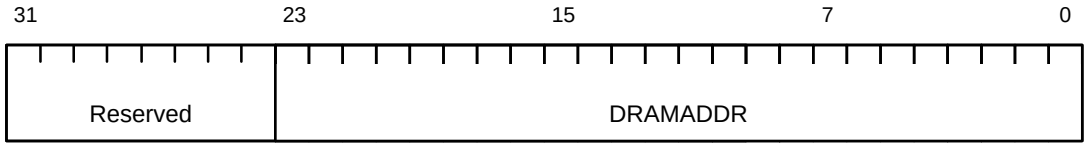


Figure 5. DMA0 Address Register

Count Tail Register, respectively. Queuing permits a second transfer to start immediately after a first transfer has terminated, greatly reducing the response-time requirement for software to set up and start the second transfer. When this bit is 0, DMA queuing is disabled, and the DMA0 Address Register and DMA0 Count Register are set directly to initiate a transfer.

Bits 3–1: Reserved

Bit 0: Count Terminate Interrupt (CTI)—The CTI bit is used to report that the DMA channel has generated an interrupt because of count termination. If the CTE bit is one and the DMACNT field decrements past zero, the CTI bit is set and a processor interrupt occurs.

DMA0 Address Register (DMAD0, Address 80000034)

The DMA0 Address Register (Figure 5) contains the addresses for a transfer by DMA Channel 0.

Bits 31–24: Reserved

Bits 23–0: DRAM Address (DRAMADDR)—This field contains the DRAM address for the next DMA transfer to or from the DRAM. The DRAMADDR field is incremented or decremented (based on the UD bit) by an amount determined by the width of the DMA transfer. The increment or decrement amount is 1 for a byte transfer, 2 for a halfword transfer, and 4 for a word transfer. To support repeated transfers from the same word, the address can be left unchanged.

The DRAMADDR field wraps from the value 000000h to FFFFFFh when decremented and from FFFFFFh to 000000h when incremented. Addresses must be aligned with the data width of the transfer.

DMA1 Control Register (DMCT1, Address 80000040)

DMA1 Control Register (Figure 6) controls DMA Channel 1. Queuing is not implemented on DMA Channel 1.

Bit 31: DMA Extend (DMAEXT)—The DMAEXT bit serves a function very similar to the IOEXTx bits in the PIA Control registers. This bit is set to provide an additional cycle of output disable time for a read or an additional cycle of data hold time for a write.

Bits 30–29: Reserved

Bits 28–24: DMA Wait States (DMAWAIT)—This field specifies the number of wait states taken by an external access by DMA Channel 1. An external DMA read cycle takes at least three cycles (two wait states) and an external DMA write cycle takes at least four cycles (three wait states). If the DMAWAIT field specifies an insufficient number of wait states for an access (for example, DMAWAIT = 00010b for a write), the processor takes the required minimum number of wait states instead of the specified number.

Bits 23–22: Data Width (DW)—This field indicates the width of the data transferred by the DMA channel, as follows:

DW Value	DMA Transfer Width
00	32 bits
01	8 bits
10	16 bits
11	32 bits, address unchanged

The value DW=11 is used to repeatedly transfer a fixed pattern from a single DRAM location to a peripheral.

Bits 21–20: DMA Request Mode (DRM)—This field indicates how external DMA requests are signaled by DREQ1, as follows:

DRM Value	DREQ1 Request
00	Active Low
01	Active High
10	High-to-Low transition
11	Low-to-High transition

The DRM field is set to 00 by a processor reset.

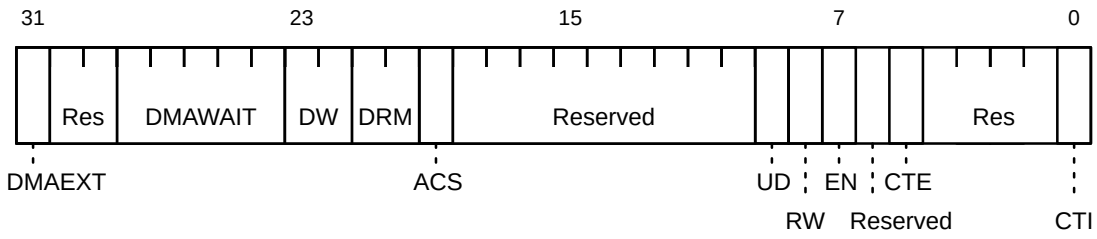


Figure 6. DMA1 Control Register

Bit 19: Assert Chip Select (ACS)—This bit controls whether DMA Channel 1 asserts $\overline{\text{PIACS}}$1 during an external peripheral access. If the ACS bit is 1, the DMA channel asserts $\overline{\text{PIACS}}$1; if the ACS bit is 0, the DMA channel does not assert $\overline{\text{PIACS}}$1.

Bits 18–10: Reserved

Bit 9: Transfer Up/Down (UD)—This bit controls the addressing of memory for the series of DMA transfers. If the UD bit is 1, the DMA address (in the DMA1 Address Register) is incremented after each transfer. If the UD bit is 0, the DMA address is decremented after each transfer. The amount by which the address is incremented or decremented is determined by the width of the transfer, as follows:

DW Value	Address Increment/Decrement
00 (32 bits)	+/– 4
01 (8 bits)	+/– 1
10 (16 bits)	+/– 2
11 (32 bits)	+/– 0

Bit 8: Read/Write (RW)—This bit controls whether the DMA transfer is to or from the DRAM. If the RW bit is 1, the DMA channel transfers data from the DRAM to the peripheral. If the RW bit is 0, the DMA channel transfers data from the peripheral to the DRAM.

Bit 7: Enable (EN)—This bit enables the DMA channel to perform transfers. A 1 enables transfers, and a 0 disables transfers.

Bit 6: Reserved

Bit 5: Count Terminate Enable (CTE)—This bit, when 1, causes the DMA channel to terminate the transfer when the DMACNT field of the DMA Count Register decrements past zero. If this bit is 0, the CTE field does not terminate the DMA transfer, though the DMA channel still decrements the count after every transfer.

Bits 4–1: Reserved

Bit 0: Count Terminate Interrupt (CTI)—The CTI bit is used to report that the DMA channel has generated an interrupt because of count termination. If the CTE bit is one and the DMACNT field decrements past zero, the CTI bit is set and a processor interrupt occurs.

PROGRAMMABLE I/O PORT

The I/O port permits direct programmable access of up to twelve external PIO signals, as either inputs, outputs, or open-drain outputs. When used as inputs, eight of these signals, PIO15–PIO8, can be programmed to cause edge- or level-sensitive interrupts. The Am29202 microcontroller supports PIO signals PIO15–PIO4.

Four PIO signals (PIO7–PIO4) are shared with the IEEE-1284-compliant parallel port interface. The access to these additional IEEE-1284-specific input and output signals is controlled by the parallel port. To use REVOE and DATASTROBE as outputs and SELECTIN and INIT as inputs, the POCT, PIN, POUT, and POEN registers must be configured before the parallel interface is enabled.

When the parallel port is enabled, it has control of the shared signals. The I/O port has control of the lines when the parallel port is not enabled or after a processor reset. The I/O port signals may be read at any time, irrespective of the parallel port ownership of those signals.

PIO Control Register (POCT, Address 800000D0)

The PIO Control Register (Figure 7) controls interrupt generation and determines the polarity of PIO15–PIO4. Note that IRM15, value 11 is now reserved; it cannot be used to signal a change in the parallel port configuration to the host.

Bits 31–30: Interrupt Request Mode, PIO15 (IRM15)—This field enables PIO15 to generate an interrupt and indicates whether PIO15 is level- or edge-sensitive in generating the interrupt. The IRM15 field controls PIO15 as follows:

IRM15 Value	PIO15 Interrupt
00	Interrupt disabled
01	Level-sensitive
10	Edge-sensitive
11	Reserved

The INVERT field (see below) further conditions interrupt generation. If the INVERT bit for PIO15 is 0, an interrupt, if enabled, is generated by a High level on PIO15 (level-sensitive) or on a Low-to-High transition (edge-sensitive) of PIO15. If the INVERT bit for PIO15 is 1, an interrupt, if enabled, is generated by a Low level on PIO15 (level-sensitive) or on a High-to-Low transition (edge-sensitive) of PIO15.

Bits 29–16: IRM14 through IRM8—The IRM14–IRM8 fields enable interrupts and specify level- or edge-sensitivity for PIO14–PIO8, respectively. These fields are identical in definition to IRM15.

Bits 15–4: PIO Inversion (INVERT)—This field determines how the level on each PIO signal is reflected in the PIO Input and PIO Output Registers, and how interrupts are generated. The most significant bit of the INVERT field determines the sense of PIO15, the next bit determines the sense of PIO14, and so on. A 0 in this field causes the internal and external sense of the respective PIO signal to be noninverted; a High external level is reflected as a 1 internally, and a Low is reflected as a 0 internally. A 1 in this field causes the internal and external sense of the respective PIO signal to be inverted; a High external level is reflected as a 0 internally, and a Low is reflected as a 1 internally.

Bits 3–0: Reserved. These bits are reserved on the Am29202 microcontroller and should be written with 0s to ensure compatibility.

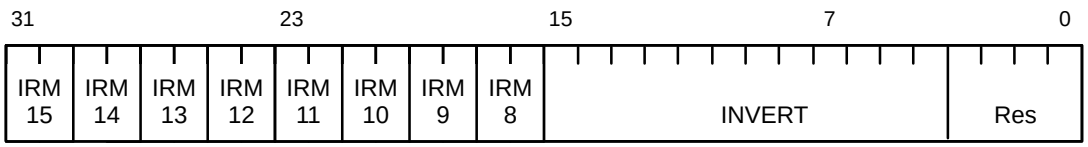


Figure 7. PIO Control Register

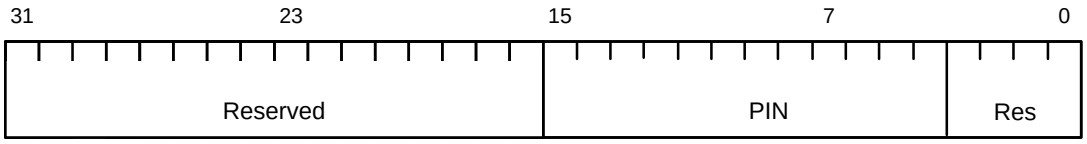


Figure 8. PIO Input Register

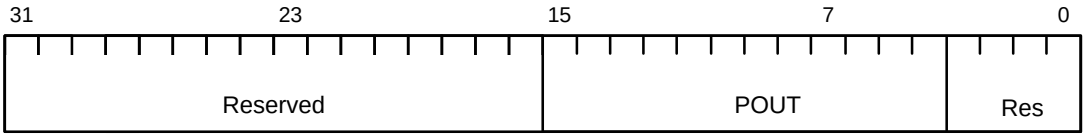


Figure 9. PIO Output Register

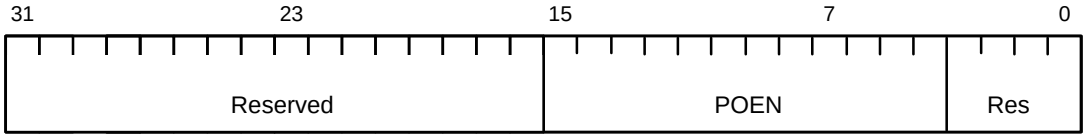


Figure 10. PIO Output Enable Register

PIO Input Register (PIN, Address 800000D4)

The PIO Input Register (Figure 8) reflects the external levels of PIO15–PIO4.

Bits 31–16: Reserved

Bits 15–4: PIO Input (PIN)—This field reflects the levels on each PIO signal. The most significant bit of the PIN field reflects the level on PIO15, the next bit reflects the level on PIO14, and so on. The correspondence between levels and bits in this register is controlled by the INVERT field.

Bits 3–0: Reserved. These bits are reserved on the Am29202 microcontroller and will be read as 0s.

PIO Output Register (POUT, Address 800000D8)

The PIO Output Register (Figure 9) determines the levels driven on the PIO signals, for those signals enabled to be driven by the PIO Output Enable Register.

Bits 31–16: Reserved

Bits 15–4: PIO Output (POUT)—This field determines the levels on each PIO signal, if so enabled by the PIO Output Enable Register. The most significant bit of the POUT field determines the level on PIO15, the next bit determines the level on PIO14, and so on. The correspondence between levels and bits in this register is controlled by the INVERT field.

Bits 3–0: Reserved. These bits are reserved on the Am29202 microcontroller and should be written with 0s to ensure compatibility.

PIO Output Enable Register (POEN, Address 800000DC)

The PIO Output Enable Register (Figure 10) determines whether or not the PIO signals are driven as outputs.

Bits 31–16: Reserved

Bits 15–4: PIO Output Enable (POEN)—This field determines whether each PIO signal is driven as an output. The most significant bit of the POEN field determines whether PIO15 is driven, the next bit determines whether PIO14 is driven, and so on. A 1 in a bit position enables the respective signal to be driven according to the associated POUT and INVERT bits, and a 0 disables the signal as an output.

Bits 3–0: Reserved. These bits are reserved on the Am29202 microcontroller and should be written with 0s to ensure compatibility.

SERIAL PORT

The on-chip serial port is a UART that permits full-duplex, bidirectional data transfer using the RS-232 standard. Serial port registers provide a programmable baud rate generator, odd/even parity capability, choice of word length, a test mode, and DMA access.

The operations of the serial port are similar on the Am29200 and Am29202 microcontrollers, except that the $\overline{DSR}$ and $\overline{DTR}$ handshake signals are not available on the Am29202 microcontroller. These functions, if needed, can be recreated with available PIO signals.

Serial Port Control Register (SPCT, Address 80000080)

The Serial Port Control Register (Figure 11) controls both the transmit and receive sections of the serial port.

Bits 31–27: Reserved

Bit 26: Loopback (LOOP)—Setting this bit places the serial port in the loopback mode. In this mode, the TXD output is set High and the Transmit Shift Register is connected to the Receive Shift Register. Data transmitted by the transmit section is immediately received by the receive section. The loopback mode is provided for testing the serial port.

Bit 25: Send Break (BRK)—Setting this bit causes the serial port to send a break, which is a continuous Low level on the TXD output for a duration of more than one frame transmission time. The transmitter can be used to time the frame by setting the BRK bit when the transmitter is empty (indicated by the TEMT bit of the Serial Port Status Register), writing the Serial Port Transmit Holding Register with data to be transmitted, and then waiting until the TEMT bit is set again before resetting the BRK bit.

Bits 24–22: Reserved

Bits 21–19: Parity Mode (PMODE)—This field specifies how parity generation and checking are performed during transmission and reception (the value "x" is a don't care):

PMODE Value	Parity Generation and Checking
0xx	No parity bit in frame
100	Odd parity (odd number of 1s in frame)
101	Even parity (even number of 1s in frame)
110	Parity forced/checked as 1
111	Parity forced/checked as 0

Bit 18: Stop Bits (STP)—A 0 in this bit specifies that one stop bit is used to signify the end of a frame. A 1 in this bit specifies that two stop bits are used to signify the end of a frame.

Bits 17–16: Word Length (WLGN)—This field indicates the number of data bits transmitted or received in a frame, as follows:

WLGN Value	Word Length
00	5 bits
01	6 bits
10	7 bits
11	8 bits

Data words of less than eight bits are right-justified in the Transmit Holding Register and Receive Buffer Register.

Bits 15–10: Reserved

Bits 9–8: Transmit Mode (TMODE)—This field enables data transmission and controls the operational mode of the serial port for the transmission of data, as follows:

TMODE Value	Effect on Transmit Section
00	Disabled
01	Generate interrupt requests for service
10	Generate DMA Channel 0 requests
11	Generate DMA Channel 1 requests

Requests for service are requests to write the Transmit Holding Register with data to be transmitted. Placing the transmit section into the disabled state causes all internal state machines to be reset and holds the transmit section in an idle state with TXD High. Serial port programmable registers are not affected when the transmit section is disabled.

Bits 7–3: Reserved

Bit 2: Receive Status Interrupt Enable (RSIE)—This bit enables the serial port to generate an interrupt because of an exception during reception. If this bit is 1 and the serial port receives a break or experiences a framing error, parity error, or overrun error, the serial port generates a Receive Status interrupt.

Bits 1–0: Receive Mode (RMODE)—This field enables data reception and controls the operational mode of the serial port for the reception of data:

RMODE Value	Effect on Receive Section
00	Disabled
01	Generate interrupt requests for service
10	Generate DMA Channel 0 requests
11	Generate DMA Channel 1 requests

Requests for service are requests to read data from the Receive Buffer Register. Placing the receive section into the disabled state causes all internal state machines to be reset and holds the receive section in an idle state. Serial port programmable registers are not affected when the receive section is disabled.

Serial Port Status Register (SPST, Address 80000084)

The Serial Port Status Register (Figure 12) indicates the status of the transmit and receive sections of the port.

Bits 31–11: Reserved

Bit 10: Transmitter Empty (TEMT)—This bit is 1 when the transmitter has no data to transmit and the Transmit Shift Register is empty. This indicates to software that it is safe to disable the transmit section.

Bit 9: Transmit Holding Register Empty (THRE) When the THRE bit is 1, the Transmit Holding Register does not contain valid data and can be written with data to be transmitted. When the THRE bit is 0, the Transmit Holding Register contains valid data not yet copied to the Transmit Shift Register for transmission and cannot be written. If so enabled by the TMODE field, the THRE bit causes an interrupt or DMA request when it is set. The THRE bit is reset automatically by writing the Transmit Holding Register. This bit is read-only, allowing other bits of the Serial Port Status Register to be written (for example, resetting the BRKI bit) without interfering with the data request.

Bit 8: Receive Data Ready (RDR)—When the RDR bit is 1, the Receive Buffer Register contains data that has been received on the serial port, and can be read to obtain the data. When the RDR bit is 0, the Receive Buffer Register does not contain valid data. If so enabled by the RMODE field, the RDR bit causes an interrupt or DMA request when it is set. The RDR bit is reset automatically by reading the Receive Buffer Register.

Bits 7–4: Reserved

Bit 3: Break Interrupt (BRKI)—The BRKI bit is set to indicate that a break has been received. If the RSIE bit is 1, the BRKI bit being set causes a Receive Status interrupt. The BRKI bit should be reset by the Receive Status interrupt handler.

Bit 2: Framing Error (FER)—This bit is set to indicate that a framing error occurred during reception of data. If the RSIE bit is 1, the FER bit being set causes a Receive Status interrupt. The FER bit should be reset by the Receive Status interrupt handler.

Bit 1: Parity Error (PER)—This bit is set to indicate that a parity error occurred during reception of data. If the RSIE bit is 1, the PER bit being set causes a Receive Status interrupt. The PER bit should be reset by the Receive Status interrupt handler.

Bit 0: Overrun Error (OER)—This bit is set to indicate that an overrun error occurred during reception of data. If the RSIE bit is 1, the OER bit being set causes a Receive Status interrupt. The OER bit should be reset by the Receive Status interrupt handler.

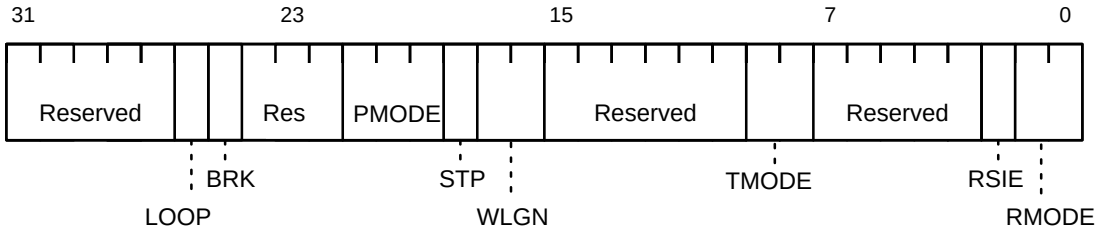


Figure 11. Serial Port Control Register

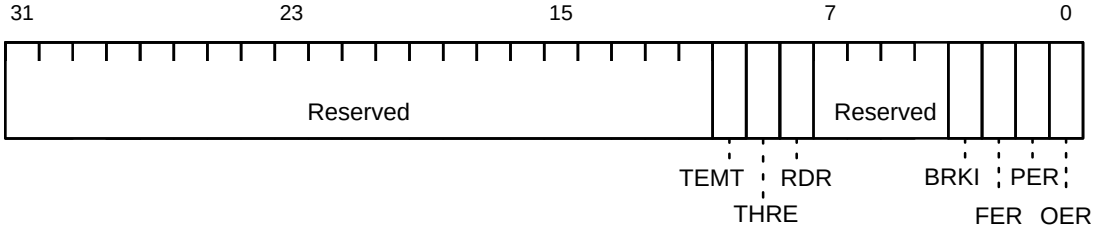


Figure 12. Serial Port Status Register

INTERRUPTS AND TRAPS

The Am29202 microcontroller employs a lightweight interrupt and trap facility that does not automatically save its current state in memory. Saving and restoring state information is under software control. Interrupts and traps are dispatched using a vector table that can be relocated in memory (see Table 4).

External traps, $\overline{\text{WARN}}$, $\overline{\text{INTR}}$3, and $\overline{\text{INTR}}$1 are not supported on the Am29202 microcontroller. PIO signals can be used as additional interrupts when more inputs are required. Two new bits are defined in the Interrupt Control Register to support the IEEE-1284 parallel port.

Current Processor Status Register (CPS, Register 2)

This protected special-purpose register (see Figure 13) controls the behavior of the processor and its ability to recognize exceptional events. The IM field values have changed for the Am29202 microcontroller.

Bits 31–18: Reserved

Bit 17: Timer Disable (TD)—When the TD bit is 1, the Timer interrupt is disabled. When this bit is 0, the Timer interrupt depends on the value of the IE bit of the Timer Reload Register. Note that Timer interrupts may be disabled by the DA bit regardless of the value of either TD or IE. The intent of this bit is to provide a means of disabling Timer interrupts without having to perform a nonatomic read-modify-write operation on the Timer Reload Register.

Bits 16–15: Reserved

Bit 14: Interrupt Pending (IP)—This bit allows software to detect the presence of interrupts while the interrupts are disabled. The IP bit is set if an interrupt request is active, but the processor is disabled from taking the resulting interrupt due to the value of the DA, DI, or IM bits. If all interrupt requests are subsequently deactivated while still disabled, the IP bit is reset.

Bits 13–12: Trace Enable, Trace Pending (TE, TP)—The TE and TP bits implement a software-controlled, instruction single-step facility. Single stepping is not implemented directly, but rather emulated by trap sequences controlled by these bits. The value of the TE bit is copied to the TP bit whenever an instruction completes execution. When the TP bit is 1, a Trace trap occurs.

Bit 11: Trap Unaligned Access (TU)—The TU bit enables checking of address alignment for external data-memory accesses. When this bit is 1, an Unaligned Access trap occurs if the processor either generates an address for an external word not aligned on a word address-boundary (i.e., either of the least significant two bits is 1) or generates an address for an external half-word not aligned on a half-word address boundary (i.e., the least significant address bit is 1). When the TU bit is 0, data-memory address alignment is ignored.

Alignment is ignored for input/output accesses. The alignment of instruction addresses is also ignored (unaligned instruction addresses can be generated only by indirect jumps). Interrupt/trap vector addresses always are aligned properly by the processor.

Bit 10: Freeze (FZ)—The FZ bit prevents certain registers from being updated during interrupt and trap processing, except by explicit data movement. The affected registers are: Channel Address, Channel Data, Channel Control, Program Counter 0, Program Counter 1, Program Counter 2, and the ALU Status Register.

When the FZ bit is 1, these registers hold their values. An affected register can be changed only by a Move-To-Special-Register instruction. When the FZ bit is 0, there is no effect on these registers and they are updated by processor instruction execution as described in this manual.

The FZ bit is set whenever an interrupt or trap is taken, holding critical state in the processor so it is not modified unintentionally by the interrupt or trap handler.

If the Freeze (FZ) bit of the Current Processor Status Register is reset from 1 to 0, two cycles are required before all program state is reflected properly in the registers affected by the FZ bit. This implies that interrupts and traps cannot be enabled until two cycles after the FZ bit is reset for proper sequencing of program state. There is no delay associated with setting the FZ bit from 0 to 1.

Bits 9–8: Reserved

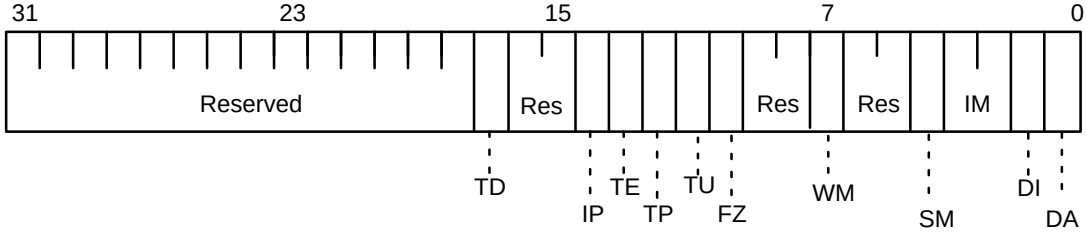


Figure 13. Current Processor Status Register

Bit 7: Wait Mode (WM)—The WM bit places the processor in the Wait mode. When this bit is 1, the processor performs no operations. The Wait mode is reset by an interrupt or trap for which the processor is enabled, or by the assertion of the $\overline{\text{RESET}}$ pin.

Bits 6–5: Reserved

Bit 4: Supervisor Mode (SM)—The SM bit protects certain processor context, such as protected special-purpose registers. When this bit is 1, the processor is in the Supervisor mode and access to all processor context is allowed. When this bit is 0, the processor is in the User mode and access to protected processor context is not allowed. An attempt to access (either read or write) protected processor context causes a Protection Violation trap.

Bits 3–2: Interrupt Mask (IM)—The IM field is an encoding of the processor priority with respect to external interrupts. The interpretation of the interrupt mask is specified as follows:

IM Value	Result
00	$\overline{\text{INTR}}$0 enabled
01	$\overline{\text{INTR}}$0 enabled
10	$\overline{\text{INTR}}$2 and $\overline{\text{INTR}}$0 enabled
11	$\overline{\text{INTR}}$2, $\overline{\text{INTR}}$0, and internal peripheral interrupts enabled

Note that the $\overline{\text{INTR}}$0 interrupt cannot be disabled by the IM field.

Bit 1: Disable Interrupts (DI)—The DI bit prevents the processor from being interrupted by internal peripheral requests and by external interrupt requests $\overline{\text{INTR}}$2 and $\overline{\text{INTR}}$0. When this bit is 1, the processor ignores all internal and external interrupts. However, internal traps, Timer interrupts, and Trace traps may be taken. When this bit is 0, the processor takes any interrupt enabled by the IM field, unless the DA bit is 1.

Bit 0: Disable All Interrupts and Traps (DA)—The DA bit prevents the processor from taking any interrupts and most traps. When this bit is 1, the processor ignores interrupts and traps. When the DA bit is 0, all traps are taken; interrupts are taken if otherwise enabled.

Interrupt Control Register (ICT, Address 80000028)

Two new bits, APDI and APCI, have been added to Interrupt Control Register (Figure 14) to support the IEEE-1284-compliant parallel port. The APDI interrupt occurs when a hardware handshake-supported data transfer mode receives (or is ready to transmit) a data byte and when interrupts are desired instead of DMA. The APCI interrupt occurs for a variety of combined mask-selectable events requiring processor intervention, such as mode changes and status input. Semi-automatic and manual modes utilize APCI interrupts for some phase transitions, including data handling.

Bits 31–28: Reserved

Bit 27: Video Interrupt (VDI)—A 1 in this bit indicates the video interface has generated an interrupt request.

Bits 26–24: Reserved

Bits 23–16: I/O Port Interrupt (IOPI)—A 1 in this field indicates the respective PIO signal has generated an interrupt request. A 1 in the most significant bit of the IOPI field indicates PIO15 has caused an interrupt, the next bit indicates PIO14 has caused an interrupt, and so on.

Bit 15: Reserved

Bit 14: DMA Channel 0 Interrupt (DMA0I)—A 1 in this bit indicates DMA Channel 0 has generated an interrupt request.

Bit 13: DMA Channel 1 Interrupt (DMA1I)—A 1 in this bit indicates DMA Channel 1 has generated an interrupt request.

Bit 12: Advanced Parallel Port Data Transfer Interrupt (APDI)—A 1 in this bit indicates that the IEEE-1284 parallel port interface is requesting a data transfer.

Writing a 1 to the APDS clears the data transfer request status that caused the APDI. This is usually not necessary, because a read or write (whichever is appropriate in a particular mode) to the Advanced Parallel Data Reg-

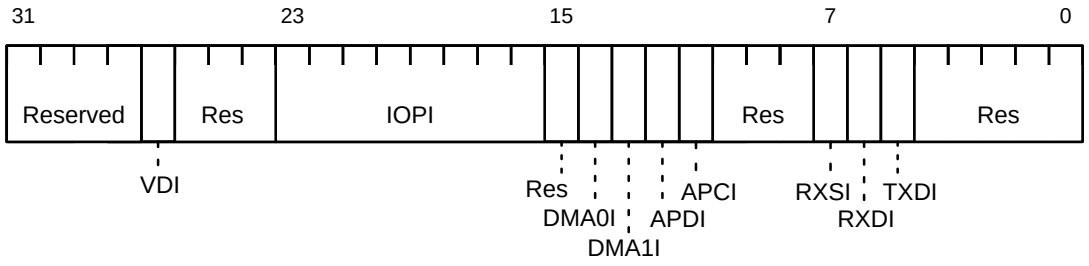


Figure 14. Interrupt Control Register

ister automatically clears the APDS condition. Clearing the APDI bit when the DMAMODE bit in the APCT Register is set is undefined and not recommended.

Bit 11: Advanced Parallel Port Control Condition Interrupt (APCI)—A 1 in this bit indicates that the IEEE-1284 parallel port interface has detected a valid control condition.

This bit is the OR of all the control condition interrupt bits in the APIS register, (ECI, DEVINITI, INITHLI, INITLHI, SELINHLI, SELINLHI, PAUTOHLI, PAUTOLHI, PSTBHL and PSTBLHI), which are in turn masked from the control condition bits in the APST register. The APCI bit must be cleared separately from the bit or bits that caused the APCI interrupt.

Writing a 1 to any of the control status bits in the APST Register or to the interrupt bits in the APIS Register will clear the condition that caused the interrupt (if masked). All of the bits require such a condition clear action, except for the device initialization interrupt: It is cleared when the parallel port interface is returned to IEEE-1284 Compatibility mode.

Bits 10–8: Reserved

Bit 7: Serial Port Receive Status Interrupt (RXSI)—A 1 in this bit indicates the serial port has generated an interrupt request because of the status of the receive logic.

Bit 6: Serial Port Receive Data Interrupt (RXDI)—A 1 in this bit indicates the serial port has generated an interrupt request because receive data is ready.

Bit 5: Serial Port Transmit Data Interrupt (TXDI)—A 1 in this bit indicates the serial port has generated an interrupt request because the Transmit Holding Register is empty.

Bits 4–0: Reserved

Vector Numbers

When an interrupt or trap is taken, the processor determines an 8-bit vector number associated with the interrupt or trap. The vector number gives the number of a vector table entry. The physical address of the vector table entry is generated by replacing bits 9–2 of the value in the Vector Area Base Address Register with the vector number.

Vector numbers are either predefined or specified by an instruction causing the trap. The assignment of vector numbers is shown in Table 4 (vector numbers are in decimal notation).

An Unsupported Peripheral Address trap has been added for the Am29202 microcontroller. A vector 6 trap will occur for accesses to peripheral addresses other than those listed in Table 2.

Table 4. Vector Number Assignments

Number	Type of Trap or Interrupt	Cause
0	Illegal Opcode	Executing undefined instruction[1]
1	Unaligned Access	Access on unnatural boundary, TU = 1
2	Out-of-Range	Overflow or underflow
3–4	Reserved	
5	Protection Violation	Invalid User-mode operation[2]
6	Unsupported Peripheral Address	Access to unsupported address
7	Reserved	
8	User Instruction Mapping Miss	No DRAM mapping for access
9	User Data Mapping Miss	No DRAM mapping for access
10	Supervisor Instruction Mapping Miss	No DRAM mapping for access
11	Supervisor Data Mapping Miss	No DRAM mapping for access
12–13	Reserved	
14	Timer	Timer Facility
15	Trace	Trace Facility
16	$\overline{\text{INTR}}$0	$\overline{\text{INTR}}$0 input
17	Reserved	
18	$\overline{\text{INTR}}$2	$\overline{\text{INTR}}$2 input
19	Internal	Internal peripheral
20–21	Reserved	
22	Floating-Point Exception	Unmasked floating-point exception[3]
23	Reserved	
24–29	Reserved for instruction emulation (opcodes D8–DD)	
30	MULTM	MULTM instruction
31	MULTMU	MULTMU instruction
32	MULTIPLY	MULTIPLY instruction
33	DIVIDE	DIVIDE instruction
34	MULTIPLU	MULTIPLU instruction
35	DIVIDU	DIVIDU instruction
36	CONVERT	CONVERT instruction
37	SQRT	SQRT instruction
38	CLASS	CLASS instruction
39–41	Reserved for instruction emulation (opcode E7–E9)	
42	FEQ	FEQ instruction
43	DEQ	DEQ instruction
44	FGT	FGT instruction
45	DGT	DGT instruction
46	FGE	FGE instruction
47	DGE	DGE instruction
48	FADD	FADD instruction
49	DADD	DADD instruction
50	FSUB	FSUB instruction
51	DSUB	DSUB instruction
52	FMUL	FMUL instruction

Notes:

1. *This vector number also results if an external device removes* $\overline{\text{INTR}}$x *before the corresponding interrupt or trap is taken by the processor.*
2. *Some Supervisor-mode operations cause Protection Violations to facilitate virtualization of certain operations.*
3. *The Floating-Point Exception trap is not generated by the processor hardware. It is generated by the software that implements the virtual arithmetic interface.*

Table 4. Vector Number Assignments (continued)

Number	Type of Trap or Interrupt	Cause
53	DMUL	DMUL instruction
54	FDIV	FDIV instruction
55	DDIV	DDIV instruction
56	Reserved for instruction emulation (opcode F8)	
57	FDMUL	FDMUL instruction
58–63	Reserved for instruction emulation (opcode FA–FF)	
64–255	ASSERT and EMULATE instruction traps (vector number specified by instruction)	See Note 4

Notes: (continued)

4. Some of Vector Numbers 64–255 are reserved for software compatibility. These are documented in the Host Interface (HIF) Specification (order# 11539) available from AMD.

Sequencing of Interrupts and Traps

To resolve conflicts, interrupts and traps are taken according to the priority shown in Table 5. In this table, interrupts and traps are listed in order of decreasing priority. Interrupts and traps fall into one of two categories depending on the timing of their occurrence relative to instruction execution. The column labels *Inst* and *Async* have the following meanings:

- *Inst*—Generated by the execution or attempted execution of an instruction.
- *Async*—Generated asynchronous to and independent of the instruction being executed, although it may be a result of an instruction executed previously.

The principle for interrupt and trap sequencing is that the highest priority interrupt or trap is taken first. Other interrupts and traps either remain active until they can be taken or they are regenerated when they can be taken. This is accomplished depending on the type of interrupt or trap, as follows:

1. All traps in Table 5 with priority 8 or 9 are regenerated by the re-execution of the causing instruction.
2. Most of the interrupts and traps of priority 3 through 7 must be held by external hardware until they are taken. The exceptions to this are listed in item 3.
3. The exceptions to item 2 are the Timer interrupt and the Trace trap. These are caused by bits in various registers in the processor and are held by these registers until taken or cleared. The two relevant bits are the Interrupt (IN) bit of the Timer Reload Register for Timer interrupts and the Trace Pending (TP) bit of the Current Processor Status Register for Trace traps.
4. All traps of priority 1 and 2 in Table 5, except for the Unaligned Access trap, are not regenerated. These traps are mutually exclusive and are given high priority because they cannot be regenerated: They must be taken if they occur. If one of these traps occurs at the same time as a reset, it is not taken and its occurrence is lost.
5. The Unaligned Access trap is regenerated internally when an external access is restarted by the Channel Address, Channel Data, and Channel Control registers. Note that this trap is not necessarily exclusive to the traps discussed in item 4 above.

Exception Reporting and Restarting

The *PC1* column in Table 5 describes the value held in the Program Counter 1 Register (PC1) when the interrupt or trap is taken. For traps in the *Inst* category, PC1 contains either the address of the instruction causing the trap, indicated by *Curr,* or the address of the instruction following the instruction causing the trap, indicated by *Next*.

For interrupts and traps in the *Async* category, PC1 contains the address of the first instruction not executed due to the taking of the interrupt or trap. This is the next instruction to be executed upon interrupt return, as indicated by *Next* in the PC1 column.

Table 5. Interrupt and Trap Priority Table

Priority	Type of Interrupt or Trap	Inst/Async	PC1	Channel Regs
1 (Highest)	User-Mode Data Mapping Miss	Inst	Next	All
	Supervisor-Mode Data Mapping Miss	Inst	Next	All
	Unsupported Peripheral Address	Inst	Next	All
2	Unaligned Access	Inst	Next	All
	Out-of-Range	Inst	Next	N/A
	Assert Instructions	Inst	Next	N/A
	Floating-Point Instructions	Inst	Next	N/A
	Integer Multiply/Divide Instructions	Inst	Next	N/A
	EMULATE	Inst	Next	N/A
3	$\overline{\text{INTR}}$0	Async	Next	Multiple
4	$\overline{\text{INTR}}$2	Async	Next	Multiple
5	Internal peripheral interrupts	Async	Next	Multiple
6	Timer	Async	Next	Multiple
7	Trace	Async	Next	Multiple
8	User-mode Inst Mapping Miss	Inst	Curr	N/A
	Supervisor-mode Inst Mapping Miss	Inst	Curr	N/A
9 (Lowest)	Illegal Opcode	Inst	Curr	N/A
	Protection Violation	Inst	Curr	N/A

DEBUGGING AND TESTING

The Am29202 microcontroller provides debugging and testing features at both the hardware and software levels. Instruction tracing and instruction breakpoints are supported. However, the processor status outputs STAT2–STAT0 are not available on the Am29202 microcontroller.

A JTAG-compliant test access port facilitates system testing in a production environment. A new main data scan path for the Am29202 microcontroller is provided below. The ICTEST1 and ICTEST2 data paths are unchanged.

Main Data Path

Table 6 shows a 160-cell path used to access the processor pins. This path is divided into five sets of cells. Where applicable, each set has a cell that enables the outputs of the set to be driven on the processor's pins. These cells are not connected to a processor pin. Some of these cells affect outputs not normally enabled and disabled during normal system operation.

The sets of cells are divided logically as follows: 1) clocks, requests, and reset, 2) miscellaneous peripheral control signals, 3) memory and peripheral controls, 4) instruction/data bus. Note that the $\overline{\text{GREQ}}$, $\overline{\text{GACK}}$, STAT2–STAT0, R/$\overline{\text{W}}$, and $\overline{\text{TR}}$ pins are included in the main scan path for special emulation devices only; these external pins are not included on the Am29202 microcontroller.

Table 6. Main Data Scan Path

Bit	Cell Name	Comments
1	MEMCLK	
2	$\overline{\text{RESET}}$	
3	LSYNC	
4	VCLK	
5	$\overline{\text{INTR}}$2	
6	$\overline{\text{INTR}}$0	
7	DREQ1	
8	$\overline{\text{GREQ}}$	
9	**TOPDRV**	Enables the drivers for PSYNC through PWE
10	PSYNCI	PSYNC input
11	PSYNCO	PSYNC output
12	VDATI	VDAT input
13	VDATO	VDAT output
14	PIOI4	PIO4 input
15	PIOO4	PIOO4 output
16	PIOI5	PIO5 input
17	PIOO5	PIO5 output
.	.	
.	.	
36	PIOI15	PIO15 input
37	PIOO15	PIO15 output
38	$\overline{\text{PBUSY}}$	
39	PACK	
40	$\overline{\text{POE}}$	
41	$\overline{\text{PWE}}$	
42	PSTROBE	
43	PAUTOFD	
44	$\overline{\text{WAIT}}$	
45	BOOTW	
46	**ABIDRV**	Enables the driving of the A21–A0 outputs
47	A0	
48	A1	
.	.	
.	.	
68	A21	

Table 6. Main Data Scan Path (continued)

Bit	Cell Name	Comments
69	**BOTDRV**	Enables the drivers for $\overline{\text{DACK}}$1 through RXD
70	$\overline{\text{DACK}}$1	
71	R/$\overline{\text{W}}$	
72	$\overline{\text{PIAOE}}$	
73	$\overline{\text{PIAWE}}$	
74	$\overline{\text{PIACS}}$0	
75	$\overline{\text{PIACS}}$1	
76	$\overline{\text{GACK}}$	
77	$\overline{\text{TR}}$	
78	$\overline{\text{WE}}$	
79	$\overline{\text{CAS}}$0	
80	$\overline{\text{CAS}}$1	
81	$\overline{\text{CAS}}$2	
82	$\overline{\text{CAS}}$3	
83	$\overline{\text{RAS}}$0	
84	$\overline{\text{RAS}}$1	
85	$\overline{\text{RAS}}$2	
86	$\overline{\text{RAS}}$3	
87	$\overline{\text{ROMOE}}$	
88	$\overline{\text{RSWE}}$	
89	$\overline{\text{ROMCS}}$0	
90	$\overline{\text{ROMCS}}$1	
91	$\overline{\text{ROMCS}}$2	
92	$\overline{\text{ROMCS}}$3	
93	TXD	
94	UCLK	
95	RXD	
96	**DBIDRV**	Enables the ID bus drivers
97	IDI0	ID0 input
98	IDO0	ID0 output
99	IDI1	ID1 input
100	IDO1	ID1 output
.	.	
.	.	
159	IDI31	ID31 input
160	IDO31	ID31 output

Note:
Drive-enable cells are shown in boldface.

IEEE-1284-COMPLIANT ADVANCED PARALLEL INTERFACE

The Am29202 microcontroller offers a new parallel port interface that is compliant with the IEEE Std 1284-1994 *Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers.*

The new Advanced Parallel Interface (API) replaces the parallel interface included on the Am29200 and Am29205 microcontrollers (referred to in this data sheet as the classic port).

Note: *This data sheet has been written with the assumption that the reader has a thorough and complete understanding of the IEEE-1284 standard and all the electrical and timing specifications it contains. IEEE Std 1284-1994 can be ordered directly from the IEEE by calling 1-800-678-IEEE (US) or 1-908-981-1393 and requesting document #SH17335.*

Upgrading Hardware and Software

The Advanced Parallel Interface has been designed to minimize the hardware and software changes required to upgrade from the classic port; however, some changes will be required to upgrade.

The maximum external system circuitry needed to implement the API is shown in Figure 15. The parallel port does not attach directly to the microcontroller, but is attached to the interface via buffers. To support the new IEEE-1284-compliant parallel port, data must be latched in the interface using a bidirectional bus-driver/latch such as a 74ALS652. The handshaking signals, PSTROBE, PAUTOFD, SELECTIN, INIT, PACK, and $\overline{\text{PBUSY}}$, are connected to the microcontroller via simple interface circuits. The inputs PSTROBE, PAUTOFD, SELECTIN, and INIT should be connected to the processor via a Schmitt-trigger inverter such as a 74HCT14, and the outputs PACK and $\overline{\text{PBUSY}}$ should be connected to the host via an inverter/driver such as a 74LS240. $\overline{\text{PERROR}}$, $\overline{\text{SELECT}}$, and FAULT are driven by software through programmer-defined PIOs or PIAs.

The API also requires software/driver changes, since the programmable registers and their addresses have changed from those used on the classic port. These changes are described later in this section.

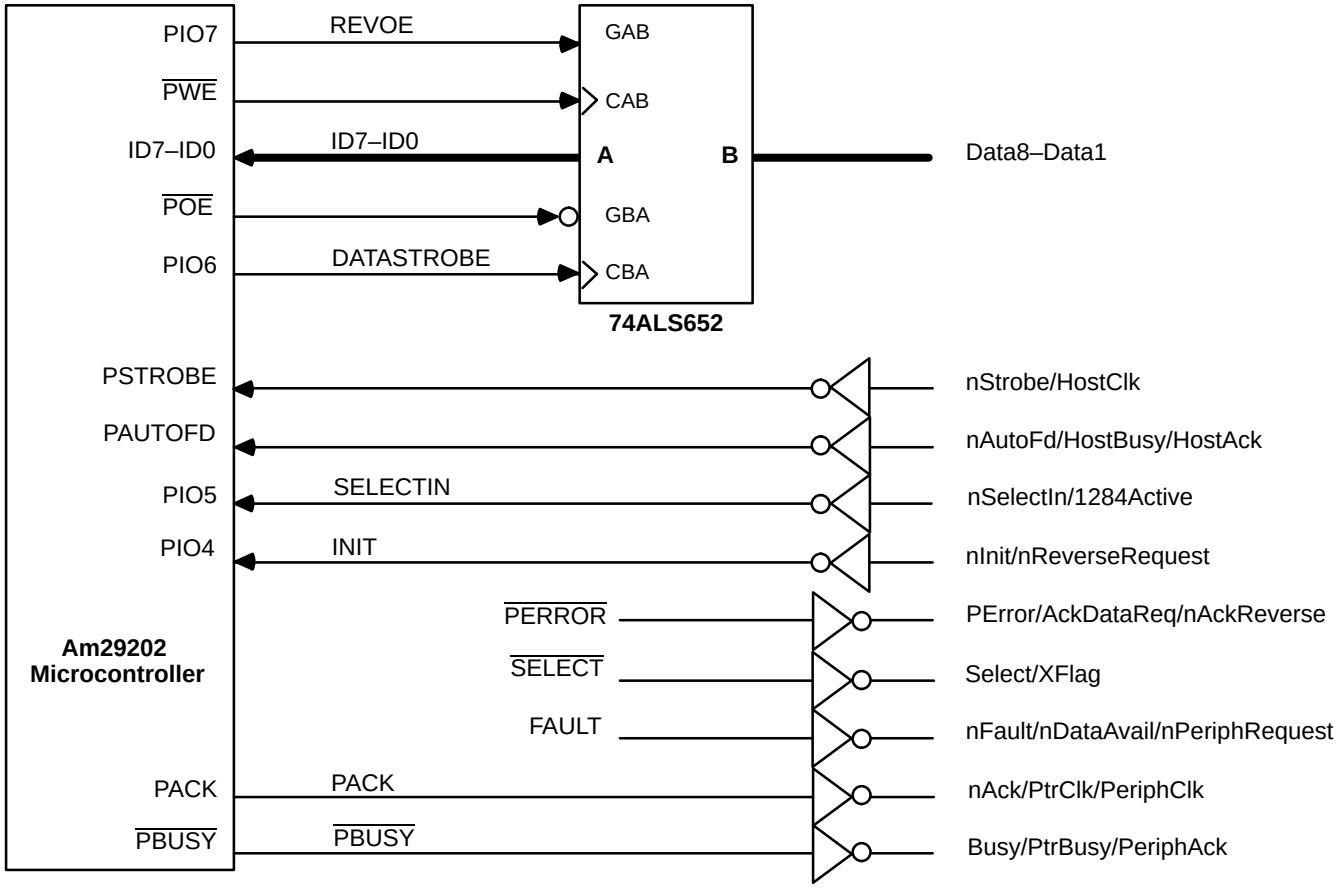


Figure 15. Maximum External System Design

Minimal System Design

While the new parallel interface on the Am29202 microcontroller adds considerable functionality, it is not required that the port be operated in full IEEE-1284 compliance. Using a subset of the hardware and the register set, the designer can set up the API to operate in a mode similar to that of the classic port on the Am29200 and Am29205 microcontrollers. A minimal system design for this configuration is shown in Figure 16.

Host-to-peripheral data must be latched in the interface using a three-state latch such as a 74LS374. The handshaking signals, PSTROBE, PAUTOFD, PACK, and $\overline{\text{PBUSY}}$, are connected to the microcontroller via simple interface circuits. The inputs PSTROBE and PAUTOFD should be connected to the processor via a Schmitt-trigger inverter such as a 74HCT14, and the outputs PACK and $\overline{\text{PBUSY}}$ should be connected to the host via an inverter such as a 74LS240.

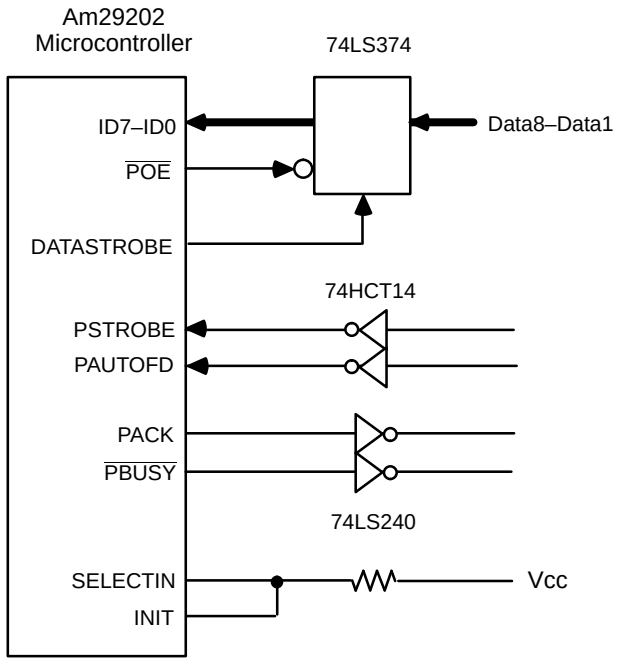


Figure 16. Minimal System Design

Software can then minimally control the API to operate in a manner similar to the classic port on the Am29200 and Am29205 microcontrollers. This is accomplished by configuring the interface for Compatibility mode (by setting APMODE to 1) and leaving it there. Interrupts will be received for data transfer in the forward direction.

OVERVIEW

The IEEE-1284 standard specifies the operation of an extensible, bidirectional, multimode parallel interface, providing access to a variety of peripheral devices, such as printers, scanners, storage devices, and network interfaces. It supports several different communications modes that allow access to both high-speed and low-overhead communications, providing a path for data to be sent from the peripheral device to the host and reducing the amount of user interaction required to operate a peripheral. AMD's implementation of the IEEE-1284 standard on the Am29202 microcontroller provides:

- **Compatibility, Nibble, Byte, and ECP modes**—Support for peripheral-side operation in these modes (host-side designs are not supported).
- **Automatic hardware handshakes**—Correctly timed requests to support data transfers that match IEEE-1284 protocols; automatic in all modes except Nibble.
- **Hardware DMA support** in all modes except Nibble.
- **External control lines**—Access to IEEE-1284 control lines through existing classic port signals and PIO lines. Registers and control logic are provided to easily support other required mode lines and controls in software.
- **Software control**—Easy access to input status information with a variety of software strategies, including polling, interrupt service, and DMA.
- **Windows Printing System compatibility**—The Am29202 microcontroller was chosen by Microsoft as its hardware reference platform for this software.

The Am29202 microcontroller supports the standard IEEE-1284 communications modes using a mixture of hardware and software controls.

Hardware controls include fast automatic data transfer handshakes and real-time status lines, as well as interrupts and pollable status for software-driven operations.

Operations not directly supported in hardware include: mode transitions of any type, IEEE-1284 negotiation and termination, mode approvals and denials, and Nibble mode data transmission. These operations are handled using interrupts and application software.

A special control interrupt in the ICT Register is provided to manage mode changes, negotiation, and termination. Using this APCI control interrupt, software modifies an API interrupt mask register at each stage of an IEEE-1284 transition, selecting the edge required for the next interrupt (as well as doing the work required at that phase transition). This structure allows for easy control of modes without processor delay loops or polling. Data handling is facilitated by a pollable status bit and a second dedicated interrupt (APDI) in the ICT Register.

Table 7. Feature Comparison of Supported IEEE-1284 Communication Modes

FEATURE	IEEE-1284 MODES			
	Compatibility (Centronics)	Nibble	Byte	ECP
Data Path	Forward (Host-to-peripheral)	Reverse (Peripheral-to-host)	Reverse (Peripheral-to-host)	Forward (Host-to-peripheral) Reverse (Peripheral-to-host)
Bidirectional	No (Note 1)	No (Note 2)	No (Note 2)	Yes
Full-Word Transfer	Yes	No	No	Forward mode only
Hardware Handshaking	Automatic	Semi-automatic	Automatic	Automatic
Hardware DMA Support	Yes	No	Yes	Yes

Notes:

1. Bidirectional when used with Nibble or Byte mode with transfer direction controlled by host.

2. Bidirectional when used with Compatibility mode. These two modes cannot be active simultaneously.

Communication Modes

AMD's implementation of the IEEE-1284 standard on the Am29202 microcontroller supports the following IEEE-1284 modes (see Table 7).

- **Compatibility Mode**—Provides an asynchronous, byte-wide forward (host-to-peripheral) channel with data and status lines used according to their original (Centronics) definitions. Compatibility mode is backward compatible with many existing devices, including the PC parallel port and the classic parallel port on the Am29200 and Am29205 microcontrollers.
- **Nibble Mode**—Provides an asynchronous, reverse (peripheral-to-host) channel, under control of the host. Data bytes are transmitted as two sequential, four-bit nibbles using four peripheral-to-host status lines. Nibble mode is used with Compatibility mode to implement a bidirectional channel. These two modes cannot be active simultaneously.
- **Byte Mode**—Provides an asynchronous, byte-wide reverse (peripheral-to-host) channel using the eight data lines of the interface for data and the control/status lines for handshaking. Byte mode is used with Compatibility mode to implement a bidirectional channel, with transfer direction controlled by the host, when the host and peripheral both support bidirectional use of the data lines. The two modes cannot be active simultaneously.
- **Extended Capabilities Port (ECP) Mode**—Provides an asynchronous, byte-wide, bidirectional channel. For faster forward transfers, an interlocked handshake replaces Compatibility mode's minimum timing requirements for its interface signals. A control line is provided to distinguish between command and data transfers. A command may optionally be used to indicate data compression or a channel address (determined by the application).

Mode selection is made by the application software, based on mode requests made by the external IEEE-1284 host. These mode requests are called IEEE-1284 negotiations and are attempts to communicate beyond the base level (Compatibility mode). The negotiations, responses, and mode changes are all moderated by the application software (an IEEE-1284 driver), on interrupts caused by the IEEE-1284 interface hardware, in response to IEEE-1284 activity on the interface.

EXTERNAL SIGNALS

Note: *All IEEE-1284 interface signal levels discussed in this document are inverted, from the IEEE-1284 cable to the peripheral circuitry at the processor, and vice versa (see Figure 15). Signal names shown in this document in all upper case letters (e.g., PSTROBE) are Am29202 microcontroller signals; they represent IEEE Std 1284-1994 signal names (e.g., nStrobe) that have been inverted at the processor/interface chip terminal.*

While many signals are named differently in each IEEE-1284 communication mode, the primary reference in this data sheet is to the Am29202 microcontroller signal name at the pin. To facilitate reference to timing diagrams in the IEEE standard document, the inverted IEEE-1284 Compatibility mode signal name is sometimes listed in this data sheet in parentheses following the Am29202 microcontroller signal name, e.g., PSTROBE (nStrobe). Table 8 maps the Am29202 microcontroller signal names to all those used in the IEEE-1284 standard.

Dedicated Signal Lines

- **PACK (output)**
  Output through an inverting buffer to nAck/PtrClk/PeriphClk, this signal is active when the API is enabled.
- **PAUTOFD (input)**
  Input via an inverting buffer from nAutofd/HostBusy/HostAck, PAUTOFD is used by the host in reverse-channel modes to signal reverse data strobe. It is also used in other contexts in various modes. The PAUTOFD signal can optionally cause control interrupts on either edge.
- **$\overline{\text{PBUSY}}$ (output)**
  Output through an inverting buffer to Busy/PtrBusy/PeriphAck, this signal comes from the Advanced Port when it is enabled.
- **$\overline{\text{POE}}$ (output)**
  Made active by a read from address 800000B0, this signal enables latched data on the data bus, to be read by the processor under interrupt or DMA control.
- **PSTROBE (input)**
  Input via an inverting buffer from nStrobe/HostClk, the PSTROBE signal is used in some forward modes to generate data strobe assertions and to signal data presence. In other modes, PSTROBE signals something other than a data transfer. The PSTROBE signal can optionally cause control interrupts on either edge.
- **$\overline{\text{PWE}}$ (output)**
  Made active by a write to address 800000B0, this signal is used to latch the data bus for outgoing (peripheral-to-host) transmission.

Mode-Allocated PIO Lines

Extended IEEE-1284 modes require dedicated control signal lines for their operation. Some of these lines appear as outputs or are read as inputs from existing PIO lines. The function of these pins changes from general-purpose PIO to specific-purpose IEEE-1284 control line while the API is enabled. When the API is enabled, the API has control of the signal lines. If the API is not enabled, the PIO port has control of the lines.

- **DATASTROBE/PIO6 (output)**
  The DATASTROBE line causes forward data to be latched in the external forward data latch during appropriate modes. This line supplies a strobe pulse with timing dependent upon the current API mode and controlled by the APMODE field.
- **INIT/PIO4 (input)**
  The INIT signal comes via an inverting buffer from nInit/nReverseRequest and can optionally cause control interrupts on either edge.
- **REVOE/PIO8 (output)**
  The REVOE signal is controlled by hardware and is used in Byte and ECP modes to force the data latch/buffer to drive data in the peripheral-to-host direction. This signal is used when the peripheral device has control of the IEEE-1284 data bus. Because of strict IEEE-1284 specifications on reinitialization, this signal must be driven directly.
- **SELECTIN/PIO5 (input)**
  The SELECTIN line comes via an inverting buffer from nSelectIn/1284Active. It transitions (along with PAUTOFD) to signal the request to negotiate an IEEE-1284 mode and to signal the termination from an IEEE-1284 mode. This line can optionally cause control interrupts on either edge.

Software-Driven Status Lines

Only the signals that are required in hardware for IEEE-1284 transfers are included in the mode-allocated PIOs. Other parallel IEEE-1284 status lines that are used during status outputs, mode transitions, or slow modes only are driven by software through programmer-defined parallel lines. PIAs or PIOs are acceptable, since these are outputs modified by software only.

- **FAULT (output)**
  This signal is driven by software and output to nFault/nDataAvail/nPeriphRequest.
- **$\overline{\text{PERROR}}$ (output)**
  This signal is driven by software and output to PError/AckDataReq/nAckReverse.
- **$\overline{\text{SELECT}}$ (output)**
  This signal is driven by software and output to Select/XFlag.

Table 8. IEEE-1284 Parallel Interface Signal Names by Mode

Am29202 Microcontroller Signal Name [1] (Inverted from IEEE-1284 bus interface)	**Signal Names As Specified in IEEE Std 1284-1994**			
	Compatibility Mode	**Nibble Mode**	**Byte Mode**	**ECP Mode**
PSTROBE	nStrobe	—	HostClk	HostClk
PAUTOFD	nAutoFd	HostBusy	HostBusy	HostAck
SELECTIN	nSelectIn	1284Active	1284Active	1284Active
INIT	nInit	—	—	nReverseRequest
ID7–ID0	Data8–Data1	—	Data8–Data1 [3]	Data8–Data1 [3]
PACK	nAck	PtrClk	PtrClk	PeriphClk
$\overline{\text{PBUSY}}$	Busy	PtrBusy/ Data4, Data8	PtrBusy	PeriphAck
$\overline{\text{PERROR}}$	PError	AckDataReq/ Data3, Data7	AckDataReq	nAckReverse
$\overline{\text{SELECT}}$	Select	XFlag/ Data2, Data6	XFlag	XFlag
FAULT	nFault	nDataAvail/ Data1, Data5	nDataAvail	nPeriphRequest
DATASTROBE [2]	DATASTROBE	—	—	DATASTROBE [4]
REVOE [2]	—	—	REVOE	REVOE [5]

Notes:

1. *The primary form of reference in this data sheet is to the Am29202 microcontroller signal name at the pin,shown in all upper case letters. To facilitate reference to timing diagrams in the IEEE-1284 standard document, the inverted IEEE-1284 Compatibility mode signal name is sometimes shown in parentheses following the Am29202 microcontroller signal name, e.g., PSTROBE (nStrobe).*
2. *These signals are not called out in the IEEE-1284 Std document. However, they are used on the Am29202 microcontroller in the modes shown.*
3. *When reversed by REVOE, these lines are bidirectional.*
4. *Used on the Am29202 microcontroller in ECP Forward mode only.*
5. *Used on the Am29202 microcontroller in ECP Reverse mode only.*

REGISTERS

The parallel port interface is controlled through its five registers, which are summarized in Table 9.

- **The Advanced Parallel Control (APCT) Register** is used to enable and control the API, to change modes, and to directly or indirectly control operation of the internal hardware. This register can be written with control data, and the status of those bit fields can be read back.
- **The Advanced Parallel Status (APST) Register** supplies real-time status information on the operation of the API and its incoming and outgoing signals. This register is comprised of three types of signals: status signals from within API hardware, real-time snooping bits of the output and input signals used by the API, and the real-time values of the interruptible condition bits prior to being masked into the Advanced Parallel Interrupt Status Register (APIS). These values are then OR'd into the Interrupt Control (ICT) Register for the microcontroller via the APCI (Advanced Parallel Control Interrupt). The APST signals provide access to the status from polling routines or interrupt service.
- **The Advanced Parallel Interrupt Mask (APIM) Register** specifies the particular IEEE-1284 signal inputs that are combined to cause the next APCI interrupt; there also exists a mask for the single APDI data interrupt. (In this context, the mask allows the corresponding signal to pass through.) This allows easy programmer control as each IEEE-1284 transition occurs. Also, a special ECP Forward mode Command Interrupt and a Device Initialization interrupt is included in the interruptible signals.
- **The Advanced Parallel Interrupt Status (APIS) Register** reports the APST condition bits that have been masked in the APIM Register. This register is used by interrupt service routines to determine the condition that caused the latest APCI interrupt. Writing to this register with a 1 in each appropriate bit position clears the condition latch for each interrupt. Also writing to the same bit positions in the APST Register clears the same condition bits, thus allowing polling routines to easily clear the same bits as interrupt routines would do in the APIS.
- **The Advanced Parallel Data (APDT) Register** is a special register decode that addresses the external data latch. The register does not exist internal to the API; it causes the generation of the $\overline{\text{POE}}$ or $\overline{\text{PWE}}$ signals that read external data, or latch it, respectively.

Table 9. Parallel Port Register Summary

Register Name	Mnemonic	Function	Address
Advanced Parallel Control	APCT	Reads and writes values of control bits	800000A0
Advanced Parallel Status	APST	Reads interface status; reads interrupt edge bits; clears interrupts	800000A4
Advanced Parallel Interrupt Mask	APIM	Reads and writes mask bit values	800000A8
Advanced Parallel Interrupt Status	APIS	Reads enabled interrupt bits and clears interrupts	800000AC
Advanced Parallel Data	APDT	Reads external latched parallel input data	800000B0

Note:
The address assignments for these registers are different from those assigned to the classic port registers on the Am29200 and Am29205 microcontrollers.

Advanced Parallel Control Register (APCT, Address 800000A0)

The Advanced Port is controlled via the Advanced Parallel Control Register (Figure 17). It contains the APMODE field, DMA channel select, and various control bits. All bits read back their written states, except for the AFAS bit, which reads back 0.

Bit 31: Internal Reverse Output Enable (INTREVOE) Setting this bit to 1 forces the external signal REVOE High. REVOE changes the data direction of the external bus buffer/latch device to peripheral-to-host to drive the IEEE-1284 bus with data that has been captured from the processor. When the external parallel data direction must be reversed, software can modify this signal on entering and exiting different IEEE-1284 modes or submodes.

REVOE is disabled by internal hardware when the interface receives a Device Initialization condition (signaled by INIT and SELECTIN asserted).

The return to Compatibility mode automatically releases the REVOE disable. Software should determine and write the proper condition of INTREVOE before returning to Compatibility mode.

There is no effect from a Device Initialization condition if INTREVOE is set to 0.

Bits 30–28: Reserved

Bit 27: Background Status Defer (BSD)—Background Status Defer is used in Nibble mode to disable a portion of the semi-automatic handshaking, allowing status signaling back to the host.

While disabled, BSD allows PACK (nAck) deassertion semi-automatic handshakes to occur with an indefinite number of transfers.

When written to a 1, BSD disables the next PACK deassertion handshake-completion mechanism. When written to a 0, it releases the automatic handshake, but only after a delay of $T_{ACKDELAY}$. This delay allows the required data setup time for status information before allowing the PACK deassertion to occur.

Bit 26: Advanced Full Word Transfer (AFWT)—When AFWT is set to 0, the data transfer logic will generate one data transfer request per input byte, and the external APDT will be defined as 8 bits wide. When AFWT is set to 1 and the API is set to Compatibility or ECP Forward modes, the data transfer logic will generate a data transfer request cycle every fourth PSTROBE (nStrobe), and the external APDT will be defined as 32 bits wide (4 transfers).

The processor may read the BC field of the APST Register to determine the number of complete handshakes that have occurred since the last full word transfer. The partial value in BC is cleared by clearing AFWT.

External logic must be used to concatenate the four forward transfer bytes into a single 32-bit big-endian packed word.

Bit 25: Command Polarity Expected (CPE)—CPE is used with DMA in ECP Forward mode only to allow data and command bytes to be handled differently.

If the command bit received with an ECP data byte does not equal CPE, then APDS is asserted and DMA is requested normally.

If the command bit equals CPE, ECS is asserted, and, if masked (enabled), ECI will cause an APCI interrupt in the ICT Register. The ECS condition is not cleared automatically when the command byte is read; it must be cleared in the APIS or APST register. The automatic handshake is not completed when the command byte is read; it is completed only when ECS is cleared.

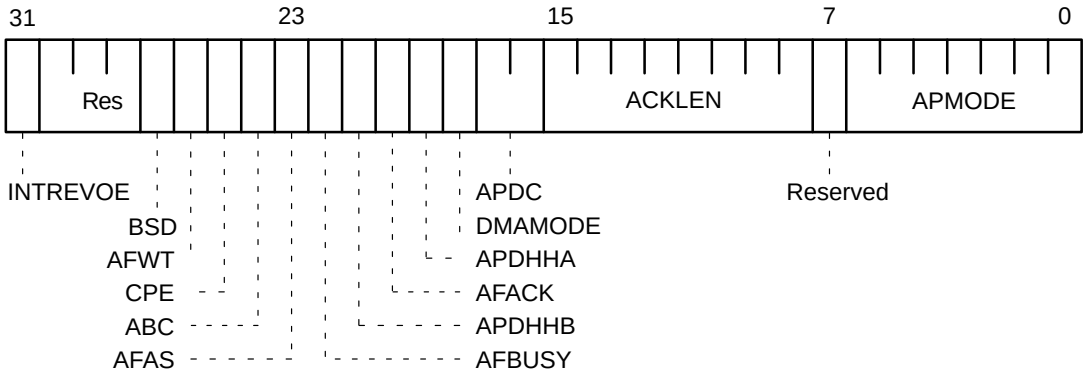


Figure 17. Advanced Parallel Control Register

Bit 24: Asynchronous Busy Control (ABC)—ABC is used to force $\overline{\text{PBUSY}}$ (Busy) asserted, during Compatibility mode only. ABC set to 1 forces $\overline{\text{PBUSY}}$ asserted. ABC set to 0 stops forcing $\overline{\text{PBUSY}}$ asserted, and allows $\overline{\text{PBUSY}}$ to return to the existing Compatibility-mode busy status. Whenever the peripheral can no longer accept a byte, this bit is set by software to asynchronously force $\overline{\text{PBUSY}}$ asserted.

ABC is only applicable in Compatibility mode; $\overline{\text{PBUSY}}$ transitions that are requested in other modes occur normally and are output on $\overline{\text{PBUSY}}$. Once the API is returned to Compatibility mode, the ABC-generated (or latch-full-generated) $\overline{\text{PBUSY}}$ will still be in effect.

Bit 23: Asynchronous Force Ack Set (AFAS)—AFAS is used to generate special pulses during Compatibility mode negotiations and delayed PACK (nAck) assertion edges in Nibble mode. AFAS always reads back 0.

In Compatibility mode, a write of 1 to AFAS forces a PACK pulse of length T_{ACKLEN} to be generated immediately. This will not normally be required because reading data from the Advanced Parallel Data Register will generate a correctly timed PACK pulse automatically (whether from an interrupt-driven instruction or from DMA). This AFAS assertion can be used to force a special PACK pulse needed in negotiation.

In Nibble mode, a write of 1 to AFAS will cause a delay of length $T_{ACKDELAY}$, and then a PACK assertion only. A PAUTOFD (nAutoFd) assertion (normal handshake) from the host then automatically clears the PACK status in this mode.

Bit 22: Advanced Force Busy (AFBUSY)—AFBUSY is an optional control bit for the $\overline{\text{PBUSY}}$ pin in the Advanced Port. Whenever APDHHB is 1, AFBUSY set to 1 forces an active level on $\overline{\text{PBUSY}}$ and AFBUSY set to 0 forces an inactive level on $\overline{\text{PBUSY}}$. The polarity from AFBUSY to $\overline{\text{PBUSY}}$ is inverted. This bit should be set to the proper condition before activating APDHHB.

AFBUSY is used when the API is in a mode that does not support hardware handshaking or in a handshaking mode where direct control is required.

If APDHHB is 1, AFBUSY directly controls the level driven on $\overline{\text{PBUSY}}$, whether or not the API is active. This allows the $\overline{\text{PBUSY}}$ pin to be used for an alternate output function if the parallel port is not used.

Bit 21: Advanced Port Disable Hardware Handshake Busy (APDHHB)—When the API is enabled, APDHHB set to 1 transfers control of $\overline{\text{PBUSY}}$ to the AFBUSY (Advanced Force Busy) register bit. APDHHB set to 0 allows API hardware handshake logic to control $\overline{\text{PBUSY}}$. The AFBUSY bit must be set to the proper condition before activating APDHHB.

Bit 20: Advanced Force Ack (AFACK)—AFACK is an optional control bit for the PACK pin in the Advanced Port. Whenever APDHHA is set to 1, AFACK set to 1 forces an active level on PACK and AFACK set to 0 forces an inactive level on PACK. The polarity from AFACK to PACK is not inverted. This bit should be set to the proper condition before activating APDHHA.

AFACK is used when the API is in a mode that does not support hardware handshaking or in a handshaking mode where direct control is required (such as negotiation).

If APDHHA is 1, AFACK directly controls the level driven on PACK, whether or not the API is active. This allows the PACK pin to be used for an alternate output function if the parallel port is not used.

Bit 19: Advanced Port Disable Hardware Handshake Ack (APDHHA)—When the API is enabled, APDHHA set to 1 transfers control of PACK to the AFACK (Advanced Force Ack) register bit. APDHHA set to 0 allows API hardware handshake logic to control PACK. The AFACK bit should be set to the proper condition before activating APDHHA.

When APDHHA is 1, the internal PACK logic will not start a PACK cycle (delayed or pulsed). This allows transitions back to internal PACK control without spurious pulses. For this reason, APDHHA should only be cleared when returning to Compatibility mode.

Bit 18: DMA Mode (DMAMODE)—DMAMODE controls which mechanism services a data transfer request condition (APDS) for modes that feature hardware handshaking. When set to 1, DMA transfers are enabled to a channel selected by the APDC field. When set to 0, DMAMODE enables interrupts on APDI (if masked).

Bits 17–16: Advanced Port DMA Channel Select (APDC)—APDC selects the DMA channel used to request a data transfer. If the API is enabled, DMAMODE is set to 1, and a data transfer request (APDS) occurs, then the DMA request of channel specified by the APDC field will be asserted.

APDC Channel	APDC1	APDC0
Channel 0	0	0
Channel 1	0	1
Reserved	1	x

Bits 15–8: Ack Length/Ack Delay (ACKLEN/ACKDELAY)—This field has two contexts: ACKLEN in Compatibility mode and ACKDELAY in all reverse modes. The period of time represented by this field is measured in MEMCLK cycles and is proportional to clock speed.

In Compatibility mode, ACKLEN is the length of the PACK (nAck) pulse generated by the automatic handshakes (or when AFAS is asserted for manual PACK control). When a data byte is read from the APDT Register or AFAS is written to a 1, a pulse is generated automatically on the PACK output of length T_{ACKLEN}. For proper operation, this field's minimum count is 1, and the maximum is 255.

In reverse modes, when a data byte is read or when AFAS is written to a 1, PACK is asserted after a delay of length $T_{ACKDELAY}$. ACKDELAY is the delay value from the time data is written to the Advanced Parallel Data Register to the time the PACK signal is generated (signaling data transfer) automatically in hardware. It provides a minimum data setup time from when data is output to when the PACK active edge signals the host that the transfer is ready. The minimum value specified for this time in the IEEE standard is 500 ns. The number of cycles that this value represents will vary with the processor clock frequency.

Bit: 7 Reserved

Bits 6–0: Advanced Parallel Mode (APMODE)—The value in APMODE (see Table 10) sets the operating mode of the API including all the automatic functions, such as data transfer request timing, PACK pulse delay and length timing, DATASTROBE source, PIO allocation, DMA direction, and $\overline{\text{PBUSY}}$ (Busy) context changes. The mode selected will remain in effect until changed. Mode changes are immediate when written. APMODE is cleared at reset time.

When set to 0, the API is disabled, and the PIO port has control of the shared signal lines. Interrupts from the API are disabled when APMODE is 0, whether their individual masks are set or not. The HL and LH status conditions for INIT, SELECTIN, PSTROBE, and PAUTOFD are not available in the APST Register when APMODE is 0.

Table 10. APMODE Values

APMODE	Mode Description	Handshake Mode	DMA Support	PIOs Allocated
0	Disabled	None	None	None
1	Compatibility Mode	Automatic	Yes	INIT/PIO4 SELECTIN/PIO5 DATASTROBE/PIO6
2	Nibble Mode (and ID)	Semi-Automatic	No	INIT/PIO4 SELECTIN/PIO5
3	Byte Mode (and ID)	Automatic	Yes	INIT/PIO4 SELECTIN/PIO5 REVOE/PIO7
4	ECP Forward Mode	Automatic	Yes	INIT/PIO4 SELECTIN/PIO5 DATASTROBE/PIO6
5	ECP Reverse Mode (and ID)	Automatic	Yes	INIT/PIO4 SELECTIN/PIO5 REVOE/PIO7

Advanced Parallel Status Register (APST, Address 800000A4)

The status bits for the real-time signals used in communication and negotiation are available in the Advanced Parallel Status Register (Figure 18).

All edge-detection and other control condition status bits for use in polling or interrupting can be read in this register, along with the data transfer status bit. These include the eight different control input signal conditions: high-to-low and low-to-high edge detection for each of the signals PSTROBE, PAUTOFD, SELECTIN, and INIT; the ECP Command condition, the Device Initialization condition, and the Data Transfer Request condition.

All condition status bits (bits 15-0) are reset-only. Writing a 1 clears the condition, and writing a 0 does not affect the bit. Writing to read-only status does not affect the bits.

The HL and LH status conditions for INIT, SELECTIN, PSTROBE, and PAUTOFD are not available in the APST Register when APMODE is 0.

Bit 31: PBUSY—This is the real-time value of the outgoing $\overline{\text{PBUSY}}$ signal. This signal directly follows the $\overline{\text{PBUSY}}$ pin.

Bit 30: PACK—This is the real-time value of the outgoing PACK signal. This signal directly follows the PACK pin.

Bit 29: INIT—This is the real-time value of the INIT input pin. INIT is input to the device on PIO4.

Bit 28: SELECTIN—This is the real-time value of the SELECTIN input pin. SELECTIN is input to the device on PIO5.

Bit 27: PAUTOFD—This is the real-time value of the PAUTOFD input pin.

Bit 26: PSTROBE—This is the real-time value of the PSTROBE input pin.

Bits 25–22: Reserved

Bits 21–20: Byte Count (BC)—When AFWT is set to 1, the Byte Count field contains the number of bytes that have been received by the external logic for concatenation into a full-word transfer. This count is useful in handling partial-word transfers, such as data streams that are a non-AFWT modulo length, or when an ECP command occurs. BC is a read-only field and is cleared when AFWT is cleared.

Bits 19–16: Reserved

Bit 15: Advanced Port Data Transfer Status (APDS)—APDS signals the readiness for a byte (or full word, if AFWT is set to 1) of data to be transferred, irrespective of the data transfer method programmed, and may be polled.

If DMAMODE is 0 and APDM is 1, then APDI is asserted and an APDI interrupt occurs in the ICT Register. (Note that there is no APDI bit in the Advanced Parallel Interrupt Status Register; an APDS assertion masked by APDM causes an APDI interrupt in the ICT directly.)

If DMAMODE is 1 and APDS is 1, then a DMA request will be made to the channel specified by the APDC bit.

Writing a 1 to this bit will clear the data transfer request condition, although this is normally not recommended, since the data transfer request will be cleared automatically when data is read or written to the Advanced Parallel Data Register. Clearing the APDS bit when DMAMODE is true is undefined and not recommended.

Bit 14: ECP Command Status (ECS)—ECS signals an ECP Forward mode command byte within the data stream. This condition blocks the completion of the ECP Forward data handshake to allow the processor time to interpret the command before more data is accepted. This handshake is held off only for command bytes; data bytes are transferred via APDI or DMA.

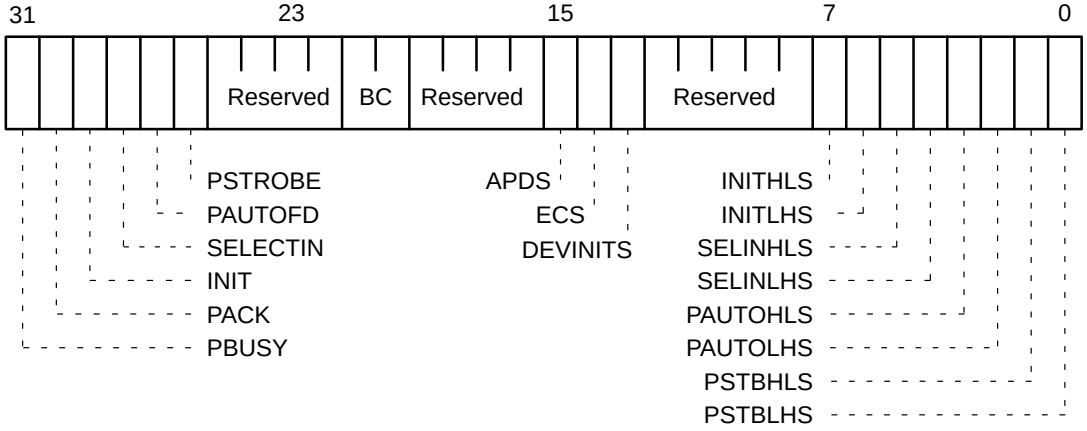


Figure 18. Advanced Parallel Status Register

This bit causes an ECI interrupt bit to be asserted when the ECM bit is set to 1. Writing a 1 to this bit will clear the ECS condition and allow the ECP Forward data handshake to proceed.

Bit 13: Device Initialization Status (DEVINITS)—This bit signals that the host has initiated an initialization cycle. The IEEE-1284 specification requires that the interface proceed immediately to Compatibility mode, accomplished by software upon the DEVINITI interrupt.

This bit causes a DEVINITI interrupt bit to be asserted when the DEVINITM bit is set to 1. (REVOE is also cleared immediately in hardware.) Writing a 1 to this bit will clear the DEVINITS condition, but is not needed typically, as the condition is cleared when the interface is returned to Compatibility mode.

Bits 12–8: Reserved

Bit 7: INIT High-to-Low Edge Detection Status (INITHLS)—INITHLS signals that the INIT signal has changed from High-to-Low (nInit has gone from Low-to-High). This bit causes an INITHLI interrupt bit to be asserted when the INITHLM bit is set to 1. Writing a 1 to this bit will clear the INITHLS condition.

Bit 6: INIT Low-to-High Edge Detection Status (INITLHS)—INITLHS signals that the INIT signal has changed from Low-to-High (nInit has gone from High-to-Low). This bit causes a INITLHI interrupt bit to be asserted when the INITLHM bit is set to 1. Writing a 1 to this bit will clear the INITLHS condition.

Bit 5: SELECTIN High-to-Low Edge Detection Status (SELINHLS)—SELINHLS signals that the SELECTIN signal has changed from High-to-Low (nSelectIn has gone from Low-to-High). This bit causes a SELINHLI interrupt bit to be asserted when the SELINHLM bit is set to 1. Writing a 1 to this bit will clear the SELINHLS condition.

Bit 4: SELECTIN Low-to-High Edge Detection Status (SELINLHS)—SELINLHS signals that the SELECTIN signal has changed from Low-to-High (nSelectIn has gone from High-to-Low). This bit causes a SELINLHI interrupt bit to be asserted when the SELINLHM bit is set to 1. Writing a 1 to this bit will clear the SELINLHS condition.

Bit 3: PAUTOFD High-to-Low Edge Detection Status (PAUTOHLS)—PAUTOHLS signals that the PAUTOFD signal has changed from High-to-Low (nAutofd has gone from Low-to-High). This bit causes a PAUTOHLI interrupt bit to be asserted when the PAUTOHLM bit is set to 1. Writing a 1 to this bit will clear the PAUTOHLS condition.

Bit 2: PAUTOFD Low-to-High Edge Detection Status (PAUTOLHS)—PAUTOLHS signals that the PAUTOFD signal has changed from Low-to-High (nAutofd signal has gone from High-to-Low). This bit causes a PAUTOLHI interrupt bit to be asserted when the PAUTOLHM bit is set to 1. Writing a 1 to this bit will clear the PAUTOLHS condition.

Bit 1: PSTROBE High-to-Low Edge Detection Status (PSTBHLS)—PSTBHLS signals that the PSTROBE signal has changed from High-to-Low (nStrobe signal has gone from Low-to-High). This bit causes a PSTBHLI interrupt bit to be asserted when the PSTBHLM bit is set to 1. Writing a 1 to this bit will clear the PSTBHLS condition.

Bit 0: PSTROBE Low-to-High Edge Detection Status (PSTBLHS)—PSTBLHS signals that the PSTROBE signal has changed from Low-to-High (nStrobe signal has gone from High-to-Low). This bit causes a PSTBLHI interrupt bit to be asserted when the PSTBLHM bit is set to 1. Writing a 1 to this bit will clear the PSTBLHS condition.

Advanced Parallel Interrupt Mask Register (APIM, Address 800000A8)

The Advanced Parallel Interrupt Mask Register (Figure 19) contains the interrupt mask bits for each condition status bit in the APST Register. When a mask bit is set to 1, it enables the corresponding condition status bit in the APST Register into the corresponding interrupt status bit of the APIS Register. When the mask bit is set to 0, the corresponding interrupt status bit is 0. The mask register is read/writeable.

All interrupt status bits in the APIS are OR'd together to form the aggregate Advanced Parallel Control Interrupt in the ICT Register.

Interrupts from the API are disabled when the APMODE is 0, whether their individual masks are set or not. Note that the APDM mask controls assertion of the APDI interrupt, while all other mask bits control the APCI interrupt.

Bits 31–16: Reserved

Bit 15: Advanced Port Data Transfer Interrupt Mask (APDM)—When APDM is set to 1, an assertion of the APDS bit directly causes an APDI interrupt in the ICT Register. When APDM is set to 0, there is no APDI interrupt.

Bit 14: ECP Command Interrupt Mask (ECM)—When ECM is set to 1, an assertion of the ECS bit sets the ECI interrupt bit, causing an APCI interrupt. When EMC is 0, the ECI interrupt bit is not set.

Bit 13: Device Initialization Interrupt Mask (DEVINITM)—When DEVINITM is set to 1, an assertion of the DEVINITS bit sets the DEVINITI interrupt bit, causing an APCI interrupt. When DEVINITM is 0, the DEVINITI interrupt bit is not set.

Bits 12–8: Reserved

Bit 7: INIT High-to-Low Interrupt Mask (INITHLM) When INITHLM is set to 1, an assertion of the INITHLS bit sets the INITHLI interrupt bit, causing an APCI interrupt. When INITHLM is 0, the INITHLI interrupt bit is not set.

Bit 6: INIT Low-to-High Interrupt Mask (INITLHM) When INITLHM is set to 1, an assertion of the INITLHS bit sets the INITLHI interrupt, causing an APCI interrupt. When INITLHM is set to 0, the INITLHI interrupt bit is not set.

Bit 5: SELECTIN High-to-Low Interrupt Mask (SELINHLM)—When SELINHLM is set to 1, an assertion of the SELINHLS bit sets the SELINHLI interrupt bit, causing an APCI interrupt. When SELINHLM is set to 0, the SELINHLI interrupt bit is not set.

Bit 4: SELECTIN Low-to-High Interrupt Mask (SELINLHM)—When SELINLHM is set to 1, an assertion of the SELINLHS bit sets the SELINLHI interrupt bit, causing an APCI interrupt. When SELINLHM is set to 0, the SELINLHI interrupt bit is not set.

Bit 3: PAUTOFD High-to-Low Interrupt Mask (PAUTOHLM)—When PAUTOHLM is set to 1, an assertion of the PAUTOHLS bit sets the PAUTOHLI interrupt bit, causing an APCI interrupt. When PAUTOHLM is set to 0, the PAUTOHLI interrupt bit is not set.

Bit 2: PAUTOFD Low-to-High Interrupt Mask (PAUTOLHM)—When PAUTOLHM is set to 1, an assertion of the PAUTOLHS bit sets the PAUTOLHI interrupt bit, causing an APCI interrupt. When PAUTOLHM is set to 0, the PAUTOLHI interrupt bit is not set.

Bit 1: PSTROBE High-to-Low Interrupt Mask (PSTBHLM)—When PSTBHLM is set to 1, an assertion of the PSTBHLS bit sets the PSTBHLI interrupt bit, causing an APCI interrupt. When PSTBHLM is set to 0, the PSTBHLI interrupt bit is not set.

Bit 0: PSTROBE Low-to-High Interrupt Mask (PSTBLHM)—When PSTBLHM is set to 1, an assertion of the PSTBLHS bit sets the PSTBLHI interrupt, causing an APCI interrupt. When PSTBLHM is set to 0, the PSTBLHI interrupt bit is not set.

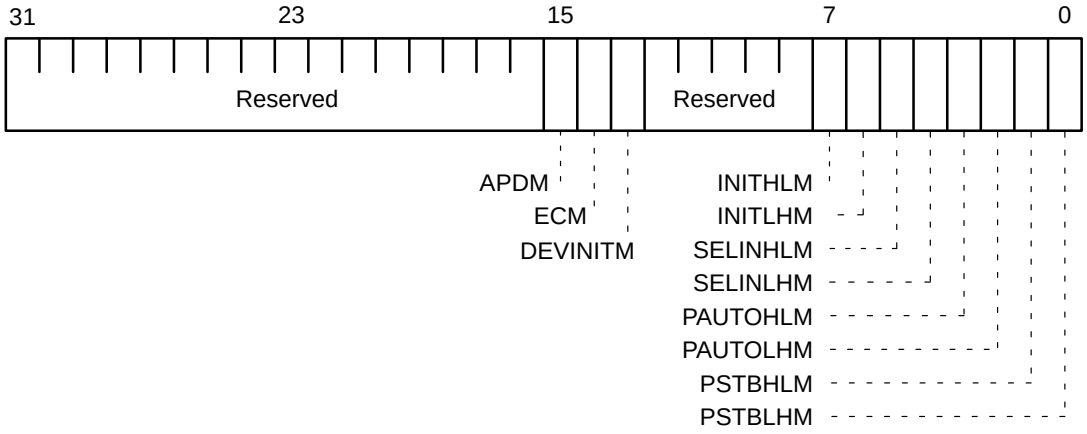


Figure 19. Advanced Parallel Interrupt Mask Register

Advanced Parallel Interrupt Status Register (APIS, Address 800000AC)

The Advanced Parallel Interrupt Status Register (Figure 20) contains the masked condition bits that make up the aggregate Advanced Parallel Control Interrupt in the ICT Register. The bits can be read to determine the source of the interrupt, and each bit can be written to a 1 to clear the corresponding condition and condition bit (writing to the condition bit in the APST Register performs the same function). The APCI is generated when the logical OR of all control interrupt status bits is 1.

Bits 31–16: Reserved

Bit 15: Reserved—There is no APDI interrupt in the APIS. Since there is only one data transfer interrupt, it asserts the APDI bit in the ICT Register directly. The APDS condition can be cleared in the APST Register.

Bit 14: ECP Command Interrupt (ECI)—This interrupt bit indicates that an ECP Forward command has been received. When this bit is a 1, the APCI interrupt occurs.

Writing a 1 to this bit clears the ECS and ECI conditions and releases the ECP Forward handshake, allowing more ECP data or command bytes to be received.

Bit 13: Device Initialization Interrupt (DEVINITI)—When this bit is a 1, the APCI interrupt occurs. When DEVINITI is written to a 1, the DEVINITS and DEVINITI bits are cleared.

Bits 12–8: Reserved

Bit 7: INIT High-to-Low Interrupt (INITHLI)—When this bit is a 1, the APCI interrupt occurs. When INITHLI is written to a 1, INITHLS and INITHLI are cleared.

Bit 6: INIT Low-to-High Interrupt (INITLHI)—When this bit is a 1, the APCI interrupt occurs. When INITLHI is written to a 1, INITLHS and INITLHI are cleared.

Bit 5: SELECTIN High-to-Low Interrupt (SELINHLI) When this bit is a 1, the APCI interrupt occurs. When SELINHLI is written to a 1, SELINHLS and SELINHLI are cleared.

Bit 4: SELECTIN Low-to-High Interrupt (SELINLHI) When this bit is a 1, the APCI interrupt occurs. When SELINLHI is written to a 1, SELINLHS and SELINLHI are cleared.

Bit 3: PAUTOFD High-to-Low Interrupt (PAUTOHLI) When this bit is a 1, the APCI interrupt occurs. When PAUTOHLI is written to a 1, PAUTOHLS and PAUTOHLI are cleared.

Bit 2: PAUTOFD Low-to-High Interrupt (PAUTOLHI) When this bit is a 1, the APCI interrupt occurs. When PAUTOLHI is written to a 1, PAUTOLHS and PAUTOLHI are cleared.

Bit 1: PSTROBE High-to-Low Interrupt (PSTBHLI) When this bit is a 1, the APCI interrupt occurs. When PSTBHLI is written to a 1, PSTBHLS and PSTBHLI are cleared.

Bit 0: PSTROBE Low-to-High Interrupt (PSTBLHI) When this bit is a 1, the APCI interrupt occurs. When PSTBLHI is written to a 1, PSTBLHS and PSTBLHI are cleared.

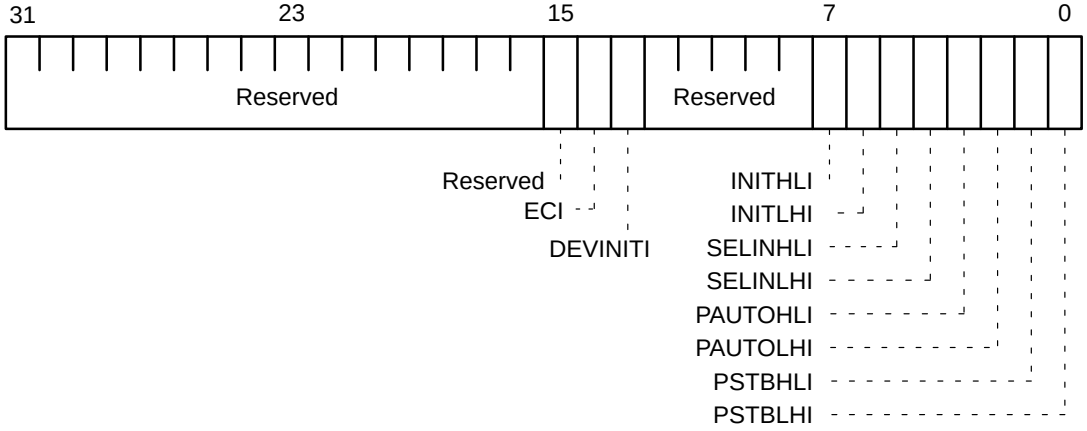


Figure 20. Advanced Parallel Interrupt Status Register

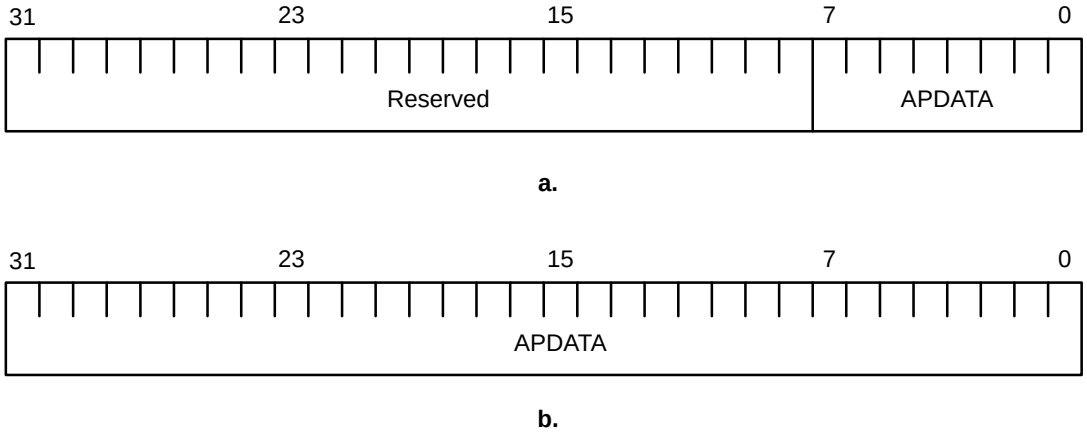


Figure 21. Advanced Parallel Data Register

Advanced Parallel Data Register (APDT, Address 800000B0)

The Advanced Parallel Data Register (Figure 21) is used to read data from and write data to the parallel port. This register is not implemented directly on the processor, but must be implemented by the user as an external bidirectional data latch.

Writing to the APDT address causes a decoded $\overline{\text{PWE}}$ output to write the current bus data byte or word to an external Data Register. Reading from the APDT address causes a decoded $\overline{\text{POE}}$ output to read data from the external data register to the data bus. The decoder operates even when the API is disabled.

Reading data from or writing data to the APDT automatically causes data transfer requests to be cleared and continues the appropriate handshake for the current mode, except in Nibble mode.

Bits 7–0: Advanced Port Parallel Data (APDATA) for 8-bit transfers (Figure 21a) or
Bits 31–0: Advanced Port Parallel Data (APDATA) for 32-bit transfers (Figure 21b)—APDATA contains packed-byte data being transferred to the processor and from the IEEE-1284 parallel bus. The register must exist external to the processor. The width of the field is dependent on the AFWT bit. AFWT is valid in Compatibility and ECP Forward modes only.

The instruction or DMA channel must be programmed for the proper width access to read the APDT correctly.

INITIALIZATION

During a processor reset, the APMODE field is set to 0, disabling parallel port interrupts and giving control of the shared PIO signals (PIO7/REVOE, PIO6/DATASTROBE, PIO5/SELECTIN, and PIO4/INIT) to the PIO port.

In the APCT Register, all fields are set to 0 except AFBUSY, APDHHB, and ACKLEN. AFBUSY and APDHHB are set to 1, forcing $\overline{\text{PBUSY}}$ (Busy) Low. The ACKLEN field is set to all 1s.

In the APST Register, the PSTBLHS, PSTBHLS, PAUTOLHS, PAUTOHLS, SELINLHS, SELINHLS, INITLHS, INITHLS, APDS, ECS, and DEVINITS bits are set to 0.

In the APIM Register, all interrupt masks are set to 0.

The POCT, PIN, POEN, and POUT registers must be configured before the parallel interface is enabled. Bits 6 and 7 of the POEN field must be set to 1; bits 4 and 5 of the POEN field must be set to 0. The parallel port interface can then be programmed incrementally, as required; there is no need to disable the interface before writing other registers.

CONTROLLING THE PARALLEL PORT INTERFACE

The API has been designed to allow easy access to input status information using a variety of software strategies, including polling, interrupt service, and DMA.

In its Advanced Parallel Status (APST) Register, the API reports a number of conditions that show either signal transitions or a data transfer request. These control and data transfer condition bits may be read and manipulated by software to control the operation of the parallel port interface (See Figures 22 and 23).

Eight of the condition bits are generated directly from external signal edges. The list below shows the signals whose edges are detected and that have corresponding edge-detection conditions in the APST Register. These input signals have one condition for high-to-low transitions, and one for low-to-high.

In general,

- **PSTROBE** signals a forward data strobe.
- **PAUTOFD** signals a reverse data strobe.
- **SELECTIN** signals 1284Active.
- **INIT** signals an ECP mode transition request or an initialization request.

The other conditions in the control group are the ECS and DEVINITS. The ECS condition signals a command byte during an ECP transfer. The DEVINITS is a signal generated when the host wants to asynchronously reinitialize the peripheral and set it back to Compatibility mode.

The APDS condition bit signals the status of a data transfer request. APDS indicates when a hardware handshake-supported data transfer mode receives (or is ready to transmit) a data byte.

Polling

The condition bit values in the APST Register can be used to request service for the conditions by polling from the support software. The specific condition can be cleared by writing a 1 to the corresponding condition bit in the APST Register.

Interrupts

If the particular condition requires faster response than polling can accomplish, an interrupt can be generated.

There are two types of API interrupts: control and data transfer. Each is supported with its own interrupt structure. The control (APCI) and data transfer (APDI) interrupts appear separately in the Interrupt Control (ICT) Register, allowing separate interrupt handlers for mode transition and data handling.

The Advanced Parallel Interrupt Mask Register (APIM) specifies the particular IEEE-1284 signal inputs that are combined to cause the next APCI interrupt. There also exists a mask (enable bit) for the single APDI data transfer interrupt. This allows easy programmer control as each IEEE-1284 transition occurs.

To generate an interrupt, the corresponding mask bit in the Advanced Parallel Interrupt Mask Register (APIM) for each condition must be set. When a condition bit is true and its mask is set, the corresponding interrupt flag is asserted in the Advanced Parallel Interrupt Status Register (APIS). This register is used by interrupt service routines to determine the condition that caused the latest APCI interrupt. All control condition interrupts and status bits must be reset manually, except for the DEVINIT status/interrupt bits, which are cleared automatically on entering Compatibility mode.

Data Transfers

Once a communications mode that supports full hardware handshaking is entered, polled APDS data transfer request bits, APDI interrupts, or DMA requests will cause a move of data from the Advanced Parallel Data (APDT) Register to memory in forward modes, and from memory to the APDT for reverse modes. The Advanced Parallel Data Register is a special register decode that addresses the external data latch. The register does not exist internal to the API; it causes the generation of the $\overline{\text{POE}}$ or $\overline{\text{PWE}}$ signals that read external data, or latch it, respectively.

In semi-automatically handshaked modes such as Nibble mode, data is handled by a combination of APDI and APCI interrupts.

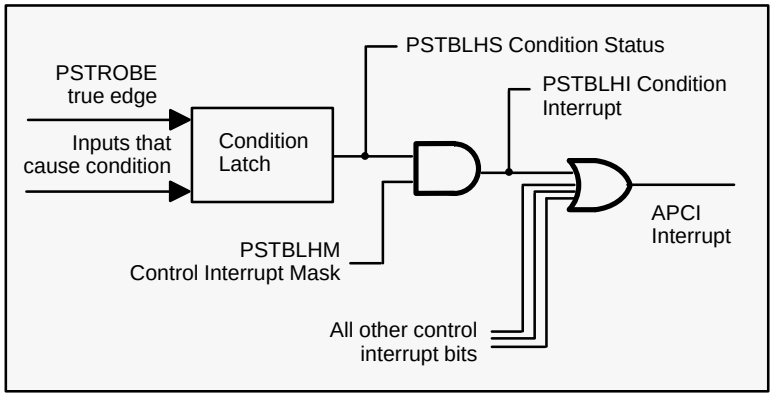
Data transfer requests, whether serviced by polling, interrupts, or DMA, are cleared automatically when the data is transferred (read or written); normally APDS need not be cleared by software directly.

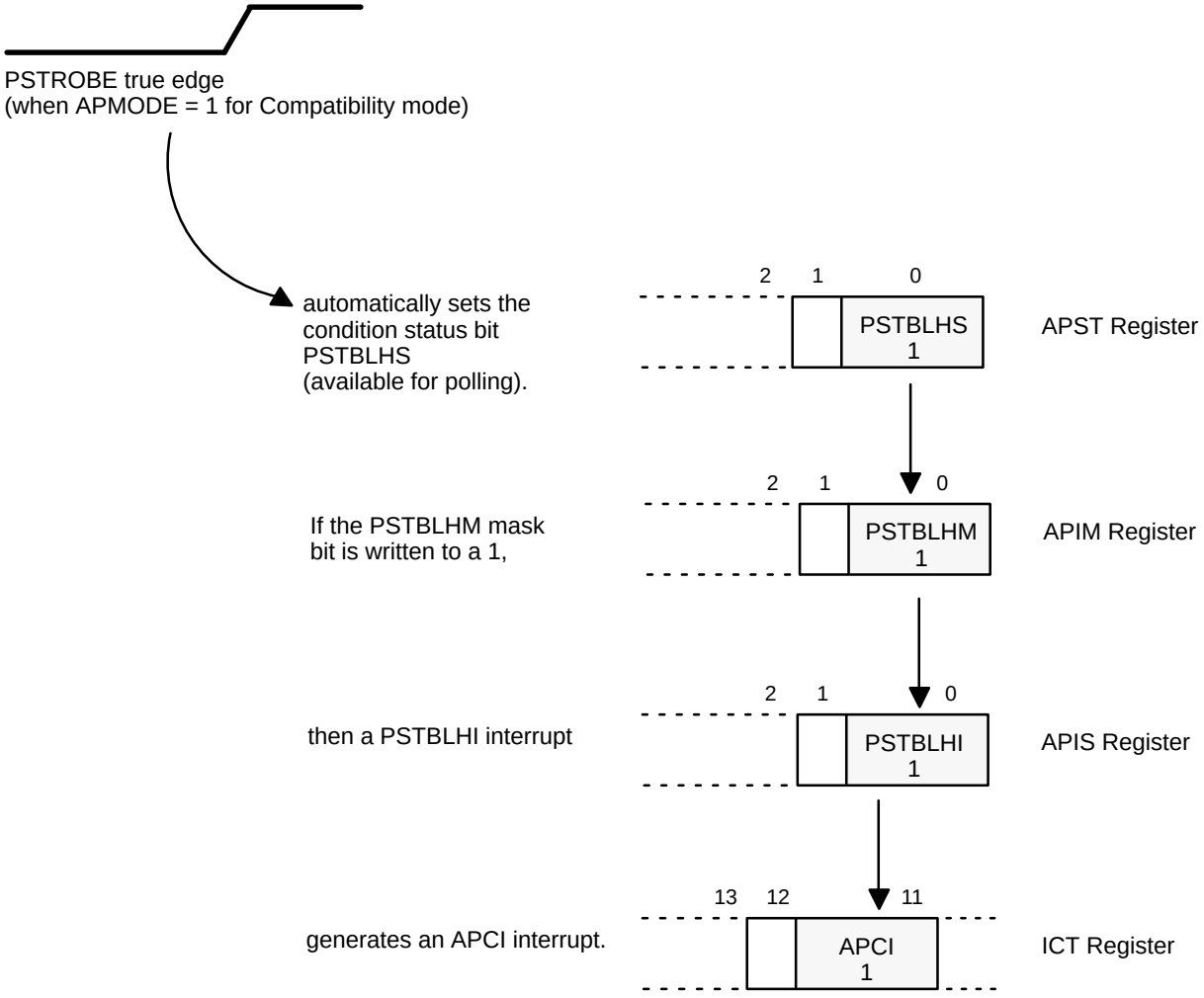
Figure 24 shows the timing of an external access. This external access is treated as either a DMA access or a processor PIA access for the purpose of prioritization with other accesses. Figure 25 shows the timing for a buffer write.

Data Interrupts

The DMAMODE bit controls whether the Data Transfer Request Status bit (APDS) causes an APDI data interrupt (if masked) or a DMA request.

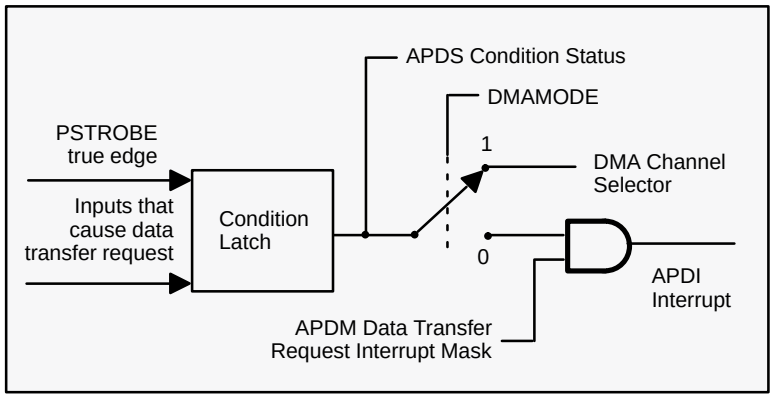
When the data transfer condition bit APDS is set to signal a data transfer, when DMAMODE is 0, and when the Data Transfer Interrupt Mask bit APDM is 1, then the APDI interrupt bit in the ICT Register is set, interrupting the processor for a data transfer.

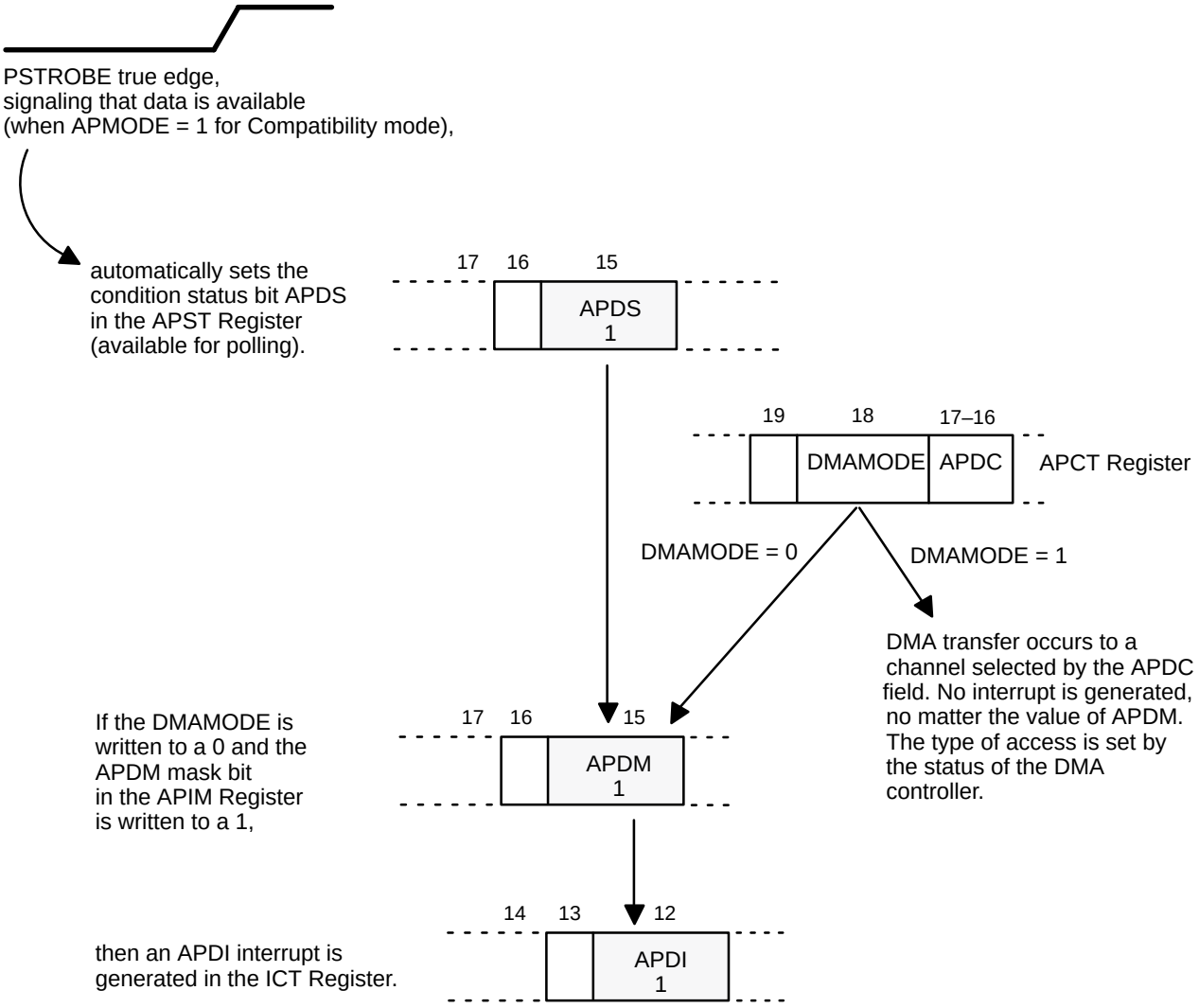




Note:
This example chain of events, based on PSTROBE true edge, is true for Compatibility mode when the value of APMODE is set to 1.

Figure 22. Example: Using A Control Status Condition to Generate an Interrupt in Compatibility Mode

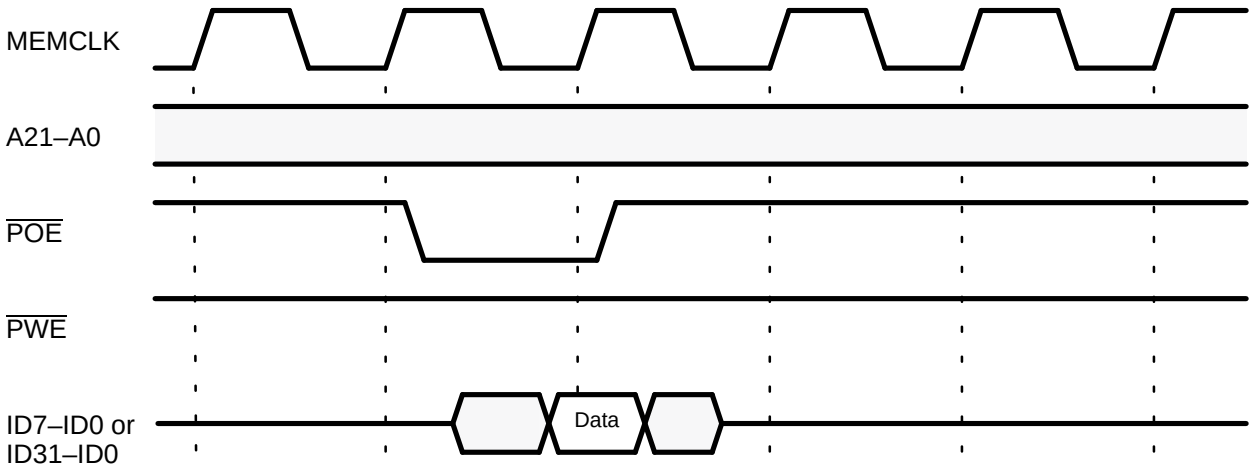




Note:

This example chain of events, based on PSTROBE true edge, is true for Compatibility mode when the value of APMODE is set to 1.

Figure 23. Example: Using the Data Status Condition in Compatibility Mode



Note:

Please refer to IEEE Std 1284-1994 for complete timing requirements, transition event descriptions, and timing diagrams specific to the standard.

Figure 24. Advanced Parallel Port Buffer Read Cycle for Forward Transfers

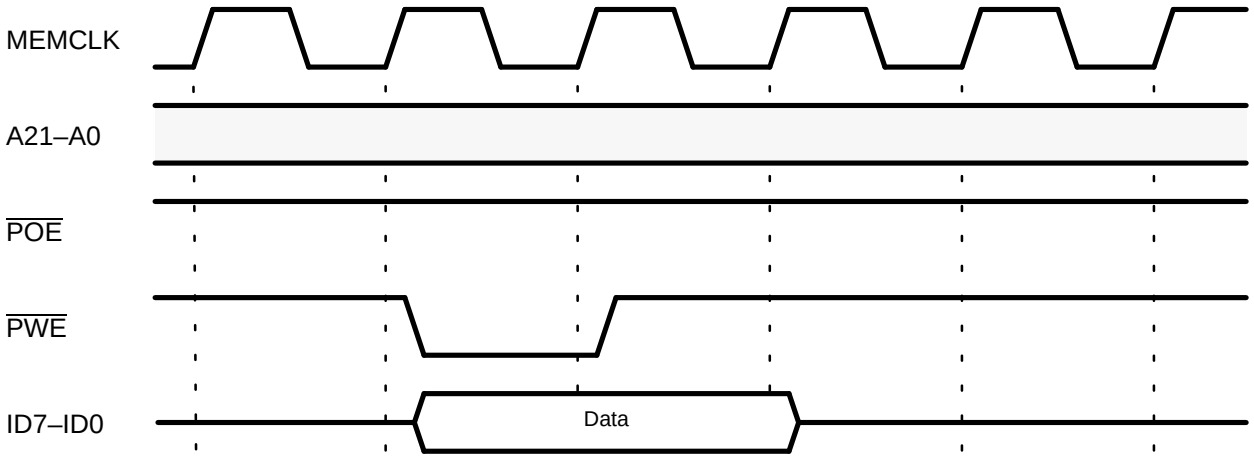


Figure 25. Advanced Parallel Port Buffer Write Cycle for Reverse Transfers

DMA

In all modes except Nibble, data handling can also be supported with DMA. When a data transfer condition (APDS) is set and DMAMODE is 1, the APDI is not asserted and a DMA transfer request is issued instead. The APDC field selects a particular DMA channel to request.

A special command interrupt allows separate handling of ECP-Forward-mode commands in the DMA data stream.

Full-Word Transfer

A faster mode of data transfer in the forward direction is the full-word transfer. Full-word transfers are valid only in Compatibility and ECP Forward modes.

This feature allows the designer to latch input data into four external latches and to read the full word from the APDT at one time, reducing the demand placed on the processor and reducing bus bandwidth requirements.

External hardware is used in full-word transfer systems to latch and concatenate the separately strobed data bytes into a 32-bit word. Then, (in full-word transfer mode) when a 32-bit word has been assembled, the API automatically issues a single data request for the entire

word. It does not issue data transfer requests for the intervening bytes; in fact, it acknowledges them without delay, speeding up the transfer greatly.

It is inadvisable to read the external APDT Register before receiving a data byte, whether in AFWT mode or not. In either case, a data byte may be lost. Resetting the external byte counter for AFWT operation whenever the APDT is read will guarantee synchronization through ECP command intervention.

ECP Commands

ECP mode supports several advanced features to improve the effectiveness of the protocol for applications such as raster image devices. These include support for multiple channels of 8-bit bidirectional transfers, as well as support for compression using run-length encoding.

To distinguish between commands and data, the context of the ECP Forward data stream may be modified by the status of the PAUTOFD (nAutoFd) signal at transfer time. This bit is known as the command bit. The IEEE-1284 standard calls for optional changes in peripheral handling of the data stream when the status of the command bit changes. These changes are completely application-specific and are optional.

The API on the Am29202 microcontroller provides three different features to facilitate automatic handling of command conditions in ECP Forward mode. These include:

- Automatic hold-off of the data transfer mechanism by not asserting APDS on the affected command byte.
- Generation of a specialized command status (that can be masked to cause a command interrupt) called ECS (ECP Command Status).
- A way to set the expected polarity of the command bit that should cause the ECS condition. This bit is called CPE (Command Polarity Expected).

When the command bit (status of PAUTOFD) is the same as CPE, the command status is set (ECS is asserted; if masked, ECI occurs). At this point, the normal data transfer request is disabled and does not occur. The data that is modified by the command bit must be handled by a separate command handler. Even when the command data is read, the normal data handshake does not occur. This allows the peripheral to read the byte value and accomplish any actions that the command implied, before allowing the data stream to restart. The data stream will restart and continue automatically when the interrupt handler clears the ECS condition (by writing a 1 to the ECS or ECI bits).

Using Full-Word Transfer with ECP Commands

When a command occurs during an ECP full-word transfer stream, the pending data count is ignored and the ECP Command Status (and if masked, an ECP Command Interrupt) bit is asserted. The software must utilize the Byte Count (BC) field to determine the actual location of the command within the full word being captured and the amount of external data that is valid. Then, a read of the APDT captures the partial word in the embedded command.

The current (command) byte will not be handshaked until the ECS/ECI status is cleared. However, this procedure should not be accomplished until the command has been interpreted, the remaining data left in the full word distributed to the appropriate buffer, and the BC field cleared. The BC field is cleared by clearing the AFWT bit.

The remaining full-word data requires that the AFWT bit be set again and the DMA controller addresses be reset for the new buffer locations affected by the intervening command. Finally, the ECS bit is cleared to allow the interface to continue with the next AFWT data byte.

Mode Selection

Changing modes successfully involves performing two different kinds of operations in software:

- Selecting an IEEE-1284 mode and communicating that choice back to the host.
- Setting up the Am29202 microcontroller's internal hardware to support the negotiated mode.

Communicating a Mode Choice to the Host

Although the IEEE-1284-compliant host initiates mode change requests, it is the software on the peripheral side that selects which mode the peripheral will support.

When the peripheral receives a request (along with an IEEE-1284 extensibility byte) from the host to enter a specific mode, the software evaluates the request and signals the host when the requested mode is one that the peripheral will support. This procedure is well documented in IEEE Std 1284-1994 with phase transition diagrams, descriptions of signal transition events, and timing diagrams.

Configuring the API to Support a Negotiated Mode

Setting up the Am29202 microcontroller hardware to support the negotiated mode is accomplished by writing a value to the APMODE field in the APCT Register. The value of APMODE (see Table 10) tells the microcontroller to interpret incoming and outgoing signals according to handshaking protocols particular to each mode. It also sets other functions, such as the allocation of PIOs and DMA direction.

Note that setting APMODE is completely independent of the mode negotiation process. The host and peripheral negotiate for a mutually acceptable mode, which may or may not match the current APMODE setting. The programmer must reset APMODE at various times during negotiation into and out of the modes:

- Immediately after the interrupt for negotiating to another mode is received, APMODE should be set for Compatibility mode.
- Immediately before negotiation is ended, APMODE should be set to the desired value for the new mode.
- When data-direction-change interrupts occur (from ECP Forward to Reverse, or ECP Reverse to Forward), APMODE should be set for the appropriate submode.

As a general rule, the desired APMODE setting should not be enabled before the programmer is completely ready for the Am29202 microcontroller hardware to begin interpreting inputs according to the automatic handshake protocols for that particular mode.

Software Control of Handshaking

If desired, the programmer can turn off all automatic handshaking and take direct control of all aspects of the API interface. Register fields provide complete access to all the required controls to operate the parallel interface using software alone.

- When set to 1, the APDHHA and APDHHB bits in the APCT Register turn over direct control of the PACK and $\overline{\text{PBUSY}}$ outputs to the AFACK and AFBUSY bits.
- The APDS field contains all real-time input values.
- When the APMODE field in the APCT Register is set to 0, no preset operating mode is defined. Although the LH and HL status conditions for PSTROBE, PAUTOFD, SELECTIN, and INIT are not available in the APST Register when APMODE is 0, the real-time status of these signals, as well as those for $\overline{\text{PBUSY}}$ and PACK, is available.

USING SOFTWARE IN IEEE-1284 MODES

The API hardware provides a rich set of controls that gives the programmer maximum flexibility. If desired, most of the API operation (except for negotiation) can be automatic. The programmer need only set up a few controls for the Am29202 microcontroller hardware to handle most handshakes automatically.

The key to this control is the APCI interrupt, generated on the edges of the four protocol signals and on the ECP Command and Device Initialization conditions. This interrupt is used to manage mode changes, negotiation, and termination. Application software must modify the Advanced Parallel Interrupt Mask Register at each stage of an IEEE-1284 transition, selecting the edges required for the next possible interrupts, as well as doing the work required at that phase transition.

This section presents some minimal programming suggestions, not necessarily complete or in sequence, to differentiate between what happens automatically in the Am29202 microcontroller hardware and what should be programmed in software. The programmer should use the IEEE Std 1284-1994 document as the authoritative reference source. Table 11, "Using Control Status Conditions in IEEE-1284 Modes," is presented to facilitate reference back and forth between the two documents.

Compatibility Mode

This mode is the first, and most basic, of the IEEE-1284 modes. It is similar to the classic (Centronics) port in timing.

The API interface should always be initialized to Compatibility mode by software. This is the default IEEE-1284 communications mode for all hosts and peripherals. This mode is maintained until the host has successfully verified that it is connected to an IEEE-1284-compliant device.

From Compatibility mode, the host can either negotiate with the peripheral for another mutually supported mode or transmit data to the peripheral using Compatibility mode. A peripheral-to-host transfer is requested by the host, negotiating with the peripheral for a mutually supported communication mode. At the direction of the host, the API interface can be returned to Compatibility mode at any time.

In Compatibility mode (APMODE set to 1), the API interface provides automatically-handshaked data transfers with polling, interrupt, or DMA support for forward byte (or full-word) transfers.

Automatic Handshakes

The API generates automatic handshakes in Compatibility mode as follows: On PSTROBE (nStrobe) true edge, the $\overline{\text{PBUSY}}$ (Busy) signal goes active and the data transfer request bit (APDS) is driven true. The DATASTROBE line is pulsed automatically when PSTROBE is asserted. This causes the external low-to-high-triggered data latch to capture the data on the active edge of PSTROBE. PACK (nAck) is driven true when data is read from APDT Register, for a pulse length of T_{ACKLEN}. $\overline{\text{PBUSY}}$ is set false when PACK is set false (the pulse is completed). PAUTOFD (nAutoFd) status is available in the APST Register.

The Acknowledge Length (ACKLEN) field in the APCT Register sets the length of the PACK pulse generated by the automatic handshakes. In software:

- Set the ACKLEN field to an appropriate length of time in MEMCLK cycles. The IEEE standard calls out a minimum PACK (nAck) pulse width of 500 ns, but a longer one may be desired. The ACKLEN field can be programmed from 1 to 255 cycles in length.

Data Transfers

Data transfer is requested on PSTROBE (nStrobe) going active. This sets the APDS bit, for polling.

To program interrupt-driven data transfers:

- Set the APDM mask bit and clear DMAMODE, causing an APDI interrupt in the ICT Register whenever the APDS is set. No DMA request will be issued.

To enable DMA transfers:

- Set the DMAMODE bit to 1, causing a DMA request to the channel set in the APDC field. No APDI is generated, irrespective of the status of APDM.

Preventing Deadlocks During Data Transfer

Deadlocks can occur when the forward channel has stalled because it is full and the host is requesting status information on the reverse channel. When using forward-channel data transfers in Compatibility mode, the programmer should set up certain controls to ensure that clogging in the forward-channel does not preclude negotiation into a reverse mode.

In order to prevent deadlocks, the programmer should be aware of what is happening with the internal buffer at all times. Setting up an internal busy length that can be detected by the application allows software to determine when the internal buffer is nearing full. There should be enough space left over in the internal buffer so that, even though the internal process is shown as busy, an APDS interrupt will still be accepted for the last byte. The last byte can then be rescued out of the external register and stuck on the end of the buffer, even though it is internally thought of as full.

Note that this requires the programmer to make a distinction between an *external* busy, where the API is technically busy ("buffer-full" busy), and an *internal*

Table 11. Using Control Status Conditions in IEEE-1284 Modes

Status Bit Name in Am29202 Microcontroller's APST Register	APST Bit Mnemonic	Signal State As Shown on Timing Diagrams in IEEE Std 1284-1994		
		Compatibility Mode	Byte and Nibble Modes	ECP Mode
PSTROBE Low-to-High (True) Edge Detection Status	PSTBLHS	nStrobe True edge	HostClk False edge	HostClk False edge
PSTROBE High-to-Low (False) Edge Detection Status	PSTBHLS	nStrobe False edge	HostClk True edge	HostClk True edge
PAUTOFD Low-to-High (True) Edge Detection Status	PAUTOLHS	nAutoFd True edge	HostBusy False edge	HostAck False edge
PAUTOFD High-to-Low (False) Edge Detection Status	PAUTOHLS	nAutoFd False edge	HostBusy True edge	HostAck True edge
SELECTIN Low-to-High (True) Edge Detection Status	SELINLHS	nSelectIn True edge	1284Active False edge	1284Active False edge
SELECTIN High-to-Low (False) Edge Detection Status	SELINHLS	nSelectIn False edge	1284Active True edge	1284Active True edge
INIT Low-to-High (True) Edge Detection Status	INITLHS	nInit True edge	nInit True edge	nReverseRequest True edge
INIT High-to-Low (False) Edge Detection Status	INITHLS	nInit False edge	nInit False edge	nReverseRequest False edge

Note:

The LH and HL designations refer to the signal at the Am29202 microcontroller, inverted from the IEEE-1284 bus.

application-supported busy, where the application decides that no more data is going to be accepted for a period of time.

In this situation, the Advanced Forced Busy (AFBUSY) bit in the APCT Register can be used to control the $\overline{\text{PBUSY}}$ pin for the external busy condition. The Asynchronous Busy Control (ABC) bit is used for the internal busy condition, to asynchronously force $\overline{\text{PBUSY}}$ asserted when the application needs to appear busy. When set, ABC will throttle the host during those periods when the internal buffer is nearing full and the application wants the peripheral to appear busy.

A simple way of preventing deadlocks is to:

- Set ABC asserted at any time to throttle host data. Interrupts received for negotiation may continue to be accepted.

Enabling Negotiation to Another Mode

To enable IEEE-1284 negotiation, in software:

- Set SELECTIN false-edge interrupt mask (SELINHLM) and PAUTOFD true-edge interrupt mask (PAUTOLHM) to cause interrupts for transition to IEEE-1284 negotiation mode. (Such a transition normally happens only in the Forward Idle state.) When either interrupt occurs, check for real-time status of the other status value to signal the negotiation request.

Negotiation Phase

In AMD's implementation of the IEEE-1284 standard on the Am29202 microcontroller, the process of negotiation between host and peripheral for a mutually acceptable mode is handled completely by software. The basic steps of the negotiation process are always the same, no matter what mode is the final target or how many times the same negotiation has already occurred.

The complete negotiation process is thoroughly described in IEEE Std 1284-1994. This section presents some minimal software recommendations that apply specifically to AMD's implementation of the standard.

Negotiation starts from the SELECTIN (nSelectIn) false-edge interrupt where PAUTOFD is true, or from the PAUTOFD true-edge interrupt where SELECTIN is false.

During negotiation, the software should take direct control of the PACK (nAck) and $\overline{\text{PBUSY}}$ (Busy) status lines. On assertion of APDHHA, PACK internal status will be cleared, and PACK will not be automatically generated in hardware. In software:

- Set APDHHA and APDHHB true. Set the proper status for signaling IEEE-1284 compliancy: $\overline{\text{PERROR}}$ (PError) true, PACK (nAck) true, FAULT (nFault) false, and $\overline{\text{SELECT}}$ (Select) true.
- To ensure that the forthcoming extensibility byte is not interpreted as data, disable data transfers by clearing the APDM and DMAMODE bits.
- Clear the SELINHLM and PAUTOLHM bits, set the PSTBLHM bit, and return from interrupt.

Once in negotiation mode, the PSTROBE high-to-low interrupt signals an extensibility byte on the data latch. In software:

- Set the PSTROBE false-edge interrupt mask (PSTBHLM). At that interrupt, read the extensibility byte in the APDT Register and determine if the mode can be supported (or, if the peripheral chooses to support it). APDS is cleared automatically.
- Set the status lines for the mode selected. Set internal $\overline{\text{PBUSY}}$ status. Set $\overline{\text{PERROR}}$ false and set FAULT (nFault) true if peripheral-to-host data is available. Set the $\overline{\text{SELECT}}$ (XFlag) line to its appropriate value (corresponding to the extensibility feature requested), indicating approval for that mode.
- Set APMODE to the correct value for the new mode.
- Before ending negotiation, enable a table of actions in the negotiation section of the driver. The applicable new mode will require a particular set of interrupt masks, data transfer modes, and status lines to be set.
- Set the PACK (nAck) line false, ending the negotiation.

If negotiation fails, the $\overline{\text{SELECT}}$ (Select) line is set false, host-to-peripheral busy status is placed on $\overline{\text{PBUSY}}$ (Busy), peripheral-to-host data available is set on FAULT (nFault), and PACK (nAck) is set false.

Terminating a Mode

The SELINLHM interrupt edge mask should always be set when in any IEEE-1284 mode, allowing driver support for mode termination. To enable application-driven termination back to Compatibility mode:

- Set the SELECTIN true-edge interrupt mask (SELINLHM). At that interrupt, manually complete the valid-state-termination handshake described in the IEEE standard and return from interrupt. A new interrupt on SELECTIN false edge can start another negotiation.

Device ID

If the negotiated mode is a device ID mode, then that mode is entered with data pending to be sent to the host. That data is the device ID string, and it is inserted into the data stream ahead of anything else already pending. That mode is ended when the ID string has been sent and must terminate for renegotiation.

Idle Mode

If the new mode is a reverse channel mode and there is no data pending (reverse idle mode), then after the status is latched on PACK (nAck), the host will either wait at busy, terminate, or force the peripheral into an idle mode.

Idle mode can be reached by the host assertion of PAUTOFD (nAutoFd) while the host thinks there is no data available, but the interface need not switch to idle phase. Internal data available status will change asynchronously through the application, but the host's knowledge of that status occurs only after it is signaled on FAULT (nFault) and only after being strobed in with PACK (nAck).

To signal the presence of new reverse data:

- Assert FAULT and pulse PACK.

Once in idle mode, the interface can either stay in idle, or terminate normally.

Nibble Mode

Nibble mode provides for slow software-driven reverse-channel communications only. Nibble mode is the only one of the supported IEEE-1284 modes that requires the programmer to handle data transfers completely in software. In order to set up the lines with status information in addition to data, the first and second nibbles are handled differently.

Data is carried on four status lines: FAULT (nFault), $\overline{\text{SELECT}}$ (Select), $\overline{\text{PERROR}}$ (PError), and $\overline{\text{PBUSY}}$ (Busy). No data is transferred on the signal lines used for forward-channel data. The forward channel continues to be driven by the host only, allowing unidirectional hosts to have access to a reverse channel.

There is semi-automatic hardware handshake support.

No DMA transfers are available in this mode.

SELECTIN (nSelectIn) true-edge interrupts must be enabled for application-driven termination back to Compatibility mode.

The DATASTROBE line is not activated, once in this mode.

Data Transfers

All data transfers are signaled via APDS status (and if masked, APDI interrupts) and are handled from software control. Semi-automatic handshakes are generated in this mode. To signal acknowledgment of data, PACK (nAck) is automatically deasserted on the deassertion of PAUTOFD (nAutoFd). This partial handshake support (on PAUTOFD false edge) is termed "semi-automatic."

In software:

- Set APDHHB true and handle $\overline{\text{PBUSY}}$ manually.
- To utilize the delayed PACK mechanism, set the APDHHA bit to 0. If fully manual control is desired, set APDHHA to 1.
- Load a delay value into ACKDELAY consistent with the IEEE-1284 standard, or longer.

First Nibble

When PAUTOFD (nAutoFd) is asserted showing host not busy, the API hardware automatically generates an APDS data transfer request.

In software:

- Distribute the bits of the low nibble of the first byte into the nibble consisting of: FAULT, $\overline{\text{SELECT}}$, $\overline{\text{PERROR}}$, and $\overline{\text{PBUSY}}$ for Data1–Data4.
- Immediately assert AFAS to force PACK (nAck) asserted after a delay of length $T_{ACKDELAY}$.

The built-in delay means that the processor need not be interrupted again until the next PAUTOFD (nAutoFd) assertion (showing ready for more data). PACK (nAck) is semi-automatically deasserted on the deassertion of PAUTOFD (nAutoFd) (signaling acknowledgment of data). This completes the first nibble of the byte.

Second Nibble

For the second nibble, an extra step must be inserted at the very end of the transfer. Before deasserting PACK (nAck), the peripheral must place status information on the lines previously used for data. Then, after a data set-up time, PACK (nAck) can be deasserted, thus ending the transfer of the byte.

For this to occur, the peripheral must do two things: block the automatic deassertion of PACK (nAck) after data is sent, and be interrupted when PAUTOFD (nAutoFd) goes inactive.

The series of steps to transfer the second nibble is shown below, in order:

When PAUTOFD (nAutoFd) is again asserted showing host not busy, the API hardware generates another APDS data transfer request.

In software:

- Place the second nibble of the byte of data onto the nibble data lines as for the first nibble. Also set PAUTOHLM and Background Status Defer (BSD) to 1.

PAUTOHLM will alert the service routine when to take the transferred data off the nibble data lines and when to put the return status information on them.

BSD controls the semi-automatic handshake that deasserts PACK (nAck), once asserted. When BSD is 0, PACK (nAck) deasserts on PAUTOFD (nAutoFd) deassertion (handshake completes). When BSD is 1, PACK (nAck) is held asserted until BSD is again set to 0 (handshake deferred). When BSD is cleared, PACK (nAck) is deasserted after a delay of length $T_{ACKDELAY}$ (deferred handshake completes).

- Once the mask and BSD are set to the correct levels, send the second nibble by again asserting AFAS.

After $T_{ACKDELAY}$, PACK (nAck) is asserted, and the host again deasserts PAUTOFD (nAutoFd). This time the PACK (nAck) is not deasserted automatically. Instead PAUTOFD (nAutoFd) deassertion causes a control interrupt on APCI. This BSD-based selection of the status-output phase increases interrupt efficiency over being interrupted every time PAUTOFD deasserts and does not require constant rewrites to the APCT Register.

In software:

- Update $\overline{\text{PBUSY}}$ (Busy) to the peripheral host-to-peripheral forward-channel-busy status (for the Compatibility mode channel), set reverse-data-available status on FAULT, set $\overline{\text{PERROR}}$ to track FAULT, and clear BSD, allowing the status phase to handshake.

After the delay, PACK (nAck) deasserts and the interface is ready to send another byte or to change modes.

Changing Modes

To enable application-driven termination back to Compatibility mode:

- Set the SELECTIN true-edge interrupt mask (SELINLHM). At that interrupt, set APMODE to 1 for Compatibility mode. Manually complete the valid-state-termination handshake described in the IEEE standard and return from interrupt. A new interrupt on SELECTIN (nSelectIn) false edge can again start another negotiation.

Nibble Idle Phase

In Nibble Idle phase, the peripheral must signal the presence of new reverse data. In software:

- Assert FAULT and pulse PACK.

Nibble ID

Nibble ID mode is identical to Nibble mode, except that Nibble ID mode is entered with data always pending, and that data is always the IEEE-1284 ID data message. Even if there is other data available in the stream, the ID message is sent before any other pending data. When the ID message is sent, the host terminates the mode and renegotiates for any further data. This mode is distinguished from Nibble mode by software only.

Byte Mode

Byte mode supports byte-wide reverse data transfers on the eight data lines used for forward channel data in Compatibility mode. API support for Byte mode is similar to that for Nibble mode, but offers automatic handshakes, faster transmission, and less complicated programming.

This mode, like ECP Reverse, requires a special external signal to reverse the data direction, driving latched output data onto the IEEE-1284 data bus. This line is called REVOE and drives the external dual-direction bus driver/latch (LS652 or ACT652).

The IEEE-1284 handshake protocol allows for the time required to disable the host data-driver and enable the peripheral data-driver, as well as to enter the reverse mode. An opposite sequence is used for termination.

Data transfers are requested on PAUTOFD (nAutoFd) assertion in this mode.

DMA transfers for data are enabled via the DMAMODE bit.

SELECTIN (nSelectIn) true-edge interrupts must be enabled for application-driven termination back to Compatibility mode.

The DATASTROBE line is not activated, once in this mode.

Automatic Handshakes

APDS occurs on PAUTOFD (nAutoFd) true edge and signals host readiness for reverse data. One data-setup time (500 ns) after the APDT data latch has been written, PACK (nAck) is automatically asserted (this is delayed through the ACKDELAY mechanism). The host will remove PAUTOFD (nAutoFd), acknowledging PACK (nAck). Finally, PACK (nAck) deassertion occurs, completing the handshake.

The host will then send a pulse on the PSTROBE (nStrobe) line signaling that the byte was accepted and processed. This PSTROBE input will not cause a forward data latching on DATASTROBE. It signals the acknowledgment of a byte only. It can be ignored, or used as a flow control indicator.

Another assertion of the PAUTOFD signal indicates that another byte should be sent; the host will only request if data is available.

The Acknowledge Delay (ACKDELAY) field in the APCT Register is the minimum data setup time from when the data is output to the time the PACK signal is generated (signaling data transfer) automatically in hardware. In software:

- Load a delay value into ACKDELAY consistent with the IEEE-1284 standard, or longer.

Data Transfers

Data transfer is requested on PAUTOFD (nAutoFd) going active. This sets the APDS bit.

To program interrupt-driven data transfers:

- Set the APDM mask bit and clear DMAMODE, causing an APDI interrupt in the ICT Register whenever the APDS is set. No DMA request will be issued.

To enable DMA transfers:

- Set the DMAMODE bit to 1, causing a DMA request to the channel pointed to by the APDC field. No APDI is generated, irrespective of the status of APDM.

Using DMA in Byte Mode

Note that when using DMA in Byte mode, the first byte cannot be transferred automatically. This is because the direction of the data driver must be reversed after the host has guaranteed disabling of its drivers. This occurs after the first byte has been requested. In software:

- Set INTREVOE immediately after PAUTOFD (nAutoFd) assertion to set up reverse transfers. (The first PAUTOFD assertion occurs after the host disables its data drivers.)
- Set an interrupt for PAUTOFD (nAutoFd) false edge. Program the DMA controller, enable it, and return from interrupt.

Setting Status Information

Status lines are read by the host at the end of each byte transfer.

Peripheral-to-host data available on FAULT (nFault) and forward host-to-peripheral busy status on $\overline{\text{PBUSY}}$ (Busy) must be setup at least 500 ns before the automatic deassertion of PACK (nAck).

The nDataAvail flow control structure defined in the IEEE-1284 standard requires a way to update the status on the FAULT (nFault) signal line. A simple procedure in software is to:

- Set the DMA channel length to n-1.
- Set the Count Terminate Enable (CTE) bit in the DMA Control Register.
- In the DMA count-terminate interrupt handler, clear the status of data available by setting the FAULT status to false before the next byte is requested.
- Load the last byte into the APDT Register, causing that byte to transmit automatically.

If the peripheral design requires a particular status function to be explicitly forced with a guaranteed data setup time, then the processor may set an interrupt for PAUTOFD deassertion and set Background Status Defer (BSD). BSD is used the same way as in Nibble mode.

The hold-off procedure described below is available but not required. Status lines can be updated by software at their activity times and will be interpreted by the processor at the next byte completion (PACK deassertion).

In software:

- Set BSD to hold off the PACK (nAck) deassertion that completes the handshake.
- Have the interrupt service routine for the PAUTOHLI update the status values, then clear BSD and PAUTOHLM (if it was a one-time update cycle), and return from interrupt.

The clearing of BSD starts an ACKDELAY cycle and deasserts PACK (nAck) at the completion of that period automatically, thus guaranteeing data setup time.

Changing Modes

To enable application-driven termination back to Compatibility mode:

- Set the SELECTIN true-edge interrupt mask (SELINLHM). At that interrupt, set APMODE to 1 for Compatibility mode. Determine and write the proper condition of INTREVOE. Manually complete the valid-state-termination handshake described in the IEEE standard and return from interrupt. A new interrupt on SELECTIN (nSelectIn) false edge can start another negotiation.

Byte Idle Phase

In Byte Idle phase, the peripheral must signal the presence of new reverse data. In software:

- Assert FAULT and pulse PACK.

Byte ID

Byte ID mode is identical to Byte mode, except that Byte ID mode is entered with data always pending, and that data is always the IEEE-1284 ID data message. Even if there is other data available in the stream, the ID message is sent before any other pending data. When the ID message is sent the host terminates the mode and renegotiates for any further data. This mode is distinguished from Byte mode by software only.

ECP MODE

A distinction of the ECP Forward and Reverse modes is that they can be transferred to and from each another without a renegotiation back to Compatibility mode. They differ from Nibble and Byte modes in that respect.

In order to set up the Am29202 microcontroller hardware for the correct automatic handshaking protocols, the programmer must set the APMODE field to the correct value when entering and leaving ECP Forward and ECP Reverse modes.

ECP Forward

ECP Forward mode operates similarly to the Compatibility mode, except that $\overline{\text{PBUSY}}$ (Busy) is used as reverse IEEE-1284 data strobe and PACK is not used at all.

The host signals that data is available by asserting PSTROBE (nStrobe). The peripheral regulates data flow by delaying the acknowledgment of the PSTROBE assertion when the channel is busy and the buffer is still full. This acknowledgment hold-off stops the host from continuing until the buffer is emptied. The APDS bit is then set by PSTROBE assertion. When the peripheral responds to the data transfer request (via polling, interrupt service, or DMA), the API releases the hold and asserts $\overline{\text{PBUSY}}$ (Busy). Because the host data retrieval and access time is overlapped with the peripheral data transfer time, the total bus speed is high.

The software must enable the control interrupt for INIT (nInit) assertion, to correctly transfer to the ECP Reverse mode.

The $\overline{\text{PERROR}}$ (PError) line is used to signal the setup phases after negotiation and before the idle phases (the beginning of automatically handshaked data transfers). It also functions as the acknowledgment of the INIT (nInit) signal, and tells the host when it can send data.

DMA transfers may be enabled for forward data using the DMAMODE bit.

FAULT (nFault) may be driven asynchronously to signal data available for transfer in ECP Reverse mode.

SELECTIN (nSelectIn) true-edge interrupts must be enabled for application-driven termination back to Compatibility mode.

Automatic Handshakes

The host signals that data is available by asserting PSTROBE (nStrobe). Once PSTROBE asserts, hardware automatically asserts $\overline{\text{PBUSY}}$ (Busy), signaling acknowledgment and readiness to receive.

If data has not yet been extracted from the data latch from the last data byte, the handshake mechanism will hold off $\overline{\text{PBUSY}}$ (Busy) assertion until the data is read from the APDT Register (irrespective of the read mechanism).

The host then responds by deasserting PSTROBE (nStrobe). A data transfer request is generated by the API hardware, along with the DATASTROBE, at PSTROBE (nStrobe) deassertion. The DATASTROBE line is pulsed automatically on PSTROBE (nStrobe) deassertion.

Finally, the peripheral automatically signals acceptance with $\overline{\text{PBUSY}}$ (Busy) deassertion.

Distinguishing Commands From Data

ECP Forward mode uses hardware handshaking to transfer eight data bits, but the context of the data is modified by the PAUTOFD (nAutoFd) signal at transfer time. The data is interpreted as a user-defined command when PAUTOFD is asserted and interpreted as target native data when PAUTOFD is deasserted.

The instance of a command byte automatically causes an ECP Command Status (ECS) condition instead of an APDS condition. In software:

- Set the ECM and the CPE bits to cause an APCI interrupt.

The hardware handshake is automatically disabled by the internal logic, and no data transfer requests occur for that byte (until the ECS/ECI condition is cleared).

In software:

- Read the command, interpret it completely, and then re-enable hardware-handshaking by clearing the ECS or ECI bits (in either the APST or APIS registers).

Using CPE

The decision to interrupt the data stream (either DMA- or interrupt-supported) for an exception byte is controlled by the Command Polarity Expected bit.

The use of the CPE bit in ECP Forward situations with only single-byte commands in long data streams is very simple. In software:

- Set CPE to 1, allowing interrupts when PAUTOFD is High (nAutoFd or HostAck is Low). Service the ECI interrupt, read the command byte from APDT to determine its meaning, set proper application-specific context, clear interrupts (to set data stream going again), and return from interrupt.

Some systems may use the command identifiers as a means to transfer a second data stream (whether thought of as an extended command stream or a second data stream). In those conditions, where a series of command bytes (command bit set) will be transferred continuously between the non-command byte stream,

the use of CPE can support easy DMA handling of both streams.

- Set CPE to 1 to detect the first command condition. In the interrupt service routine for ECI, read the command byte directly from the APDT (the handshake does not complete in this case), and set the context or condition required.
- Then, reverse the polarity of CPE and load a new address and count into the DMA controller to handle the second (command) data stream. Move the first byte of the command stream to the buffer front and release the channel by clearing ECS/ECI, and return from interrupt.

The DMA will transfer the remaining bytes in the command stream, and when the command bit changes again, the ECS/ECI will once again occur.

- Handle the transitions between conditions in the same way, alternating between DMA pointers.

Handling Deadlocks

Note that if the peripheral deadlocks during forward data transmission, the host will signal INIT (nInit), and then, expect a $\overline{\text{PERROR}}$ (PError) to allow a reverse mode transfer.

To ensure proper handling of deadlocks:

- Always accept INITs in the middle of a handshake (ECP Busy condition). They will only come if the peripheral deadlocks (stays busy in the middle of a handshake for more than 35 ms).
- Always clear the ECP Forward channel busy status whenever INIT occurs.

Changing Modes

To transfer into ECP Reverse mode:

- Set INIT true-edge interrupt mask (INITLHM). At that interrupt, set APMODE to 5 for ECP Reverse mode. Set INTREVOE, set up any DMA control variables, reverse the polarity of the INIT interrupt, and assert $\overline{\text{PERROR}}$ (PError) to begin reverse data transfer.

To enable application-driven termination back to Compatibility mode:

- Set the SELECTIN true-edge interrupt mask (SELINLHM). At that interrupt, set APMODE to 1 for Compatibility mode. Determine and write the proper condition of INTREVOE. Manually complete the valid-state-termination handshake described in the IEEE standard and return from interrupt. A new interrupt on SELECTIN (nSelectIn) false edge can start another negotiation.

ECP Reverse

ECP Reverse mode, like Byte mode, requires a special output line to reverse the data direction, driving latched output data onto the IEEE-1284 data bus. This line is called REVOE and drives the external bidirectional bus driver/latch (LS652 or ACT652). The IEEE-1284 handshake protocol allows for the timing of host data-driver disabling and of peripheral data-driver enabling for entering reverse modes, and the opposite sequence for termination.

The control interrupt for INIT (nInit) deassertion must be enabled to correctly transfer to the ECP Forward mode.

The $\overline{\text{PERROR}}$ (PError) line functions as the acknowledgment of the INIT (nInit) signal and tells the host when it can send data, in the reverse-to-forward phase.

DMA transfers for data are enabled via the DMAMODE bit.

The host signals readiness for another byte of data by asserting PAUTOFD (nAutoFd).

SELECTIN (nSelectIn) true-edge interrupts must be enabled for application-driven termination back to Compatibility mode.

The DATASTROBE line is not activated, once in this mode.

Automatic Handshakes

The host signals readiness for data by requesting ECP Reverse mode. The API automatically generates APDS upon entry and automatically asserts PACK (nAck) after the ACKDELAY period. PACK (nAck) assertion from the peripheral is answered by PAUTOFD (nAutoFd) deassertion signaling acknowledgment.

The API logic responds to PAUTOFD (nAutoFd) deassertion by causing automatic PACK (nAck) deassertion handshake, and PACK (nAck) deassertion by the peripheral is answered by a PAUTOFD (nAutoFd) assertion from the host, signaling an end to the handshake for the byte. This final phase causes another data transfer request.

In software:

- Set the ACKLEN field to an appropriate data setup time in MEMCLK cycles.

Data Transfers

A data transfer is automatically requested in two different situations.

When the interface is first changed from ECP Forward to ECP Reverse, the data transfer request is set, allowing the programmer to setup the DMA control variables and then transfer to ECP Reverse without being required to “prime the pump” (which entails writing the first byte into

the APDT Register from the program and loading the DMA controller for n-1 bytes).

Once ECP Reverse mode is established, PAUTOFD (nAutoFd) assertion at the end of a transfer automatically causes the next data transfer request.

To enable reverse data transfers:

- Set INTREVOE immediately after the PAUTOFD (nAutoFd) assertion.

To program interrupt-driven data transfers:

- Set the APDM mask bit and clear DMAMODE, causing a APDI interrupt in the ICT Register whenever the APDS is set. No DMA request will be issued.

To enable DMA transfers:

- Set the DMAMODE bit to 1, causing a DMA request to the channel pointed to by the APDC field. No APDI is generated, irrespective of the status of APDM.

Distinguishing Commands From Data

ECP Reverse mode uses hardware handshaking to transfer eight data bits to the host, but the context of the data may be modified by the $\overline{\text{PBUSY}}$ (Busy) signal.

- To transfer data using DMA, force AFBUSY to the condition required for the data stream, sending all bytes as data.
- To send commands using DMA, set the DMA byte count for the stretch of non-command data, allowing automatic handshaking and DMA data support.
- Set the CTE bit in the DMA Control Register.

Then, on the DMA count-terminate interrupt, the processor may assert the command bit and send a single command byte by writing AFBUSY and writing the command byte directly to the APDT.

- Program the next stretch of non-command data into the DMA byte length for the next automatic handshaking period.

If data transfer is handled by APDI interrupts only, then each individual request for data may be used to set both reverse data into the APDT, as well as the command status into AFBUSY.

Changing Modes

To transfer directly into ECP Forward mode:

- Set INIT false-edge interrupt mask (INITHLM). At that interrupt, set APMODE to 4 for ECP Forward mode. Set INTREVOE to 0 and assert PSTROBE (nStrobe) to begin forward data transfer.

If the host transitions back to ECP Forward mode with the INIT (nInit) deassertion, pending data transfer requests must be cleared before returning to ECP Forward mode. In software:

- Write a 1 to the APDS (after disabling the DMA controller if used).

To enable application-driven termination back to Compatibility mode:

- Set the SELECTIN true-edge interrupt mask (SELINLHM). At that interrupt, set APMODE to 1 for Compatibility mode. Determine and write the proper condition of INTREVOE. Manually complete the valid-state-termination handshake described in the IEEE standard and return from interrupt. A new interrupt on SELECTIN (nSelectIn) false edge can start another negotiation.

ECP Reverse ID

ECP Reverse ID mode is identical to ECP Reverse mode, except that ECP Reverse ID mode is entered with data always pending, and that data is always the IEEE-1284 ID data message. Even if there is other data available in the stream, the ID message is sent before any other pending data. When the ID message is sent, the host terminates the mode and renegotiates for any further data. This mode is distinguished from ECP Reverse mode by software only.

ABSOLUTE MAXIMUM RATINGS

Storage Temperature . . . . . –65°C to +125°C
Voltage on any Pin
with Respect to GND . . . . . –0.5 to V_{CC} +0.5 V
Maximum V_{CC} . . . . . 6.0 V DC

Stresses outside the stated ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device functionality.

OPERATING RANGES

Commercial (C) Devices

Case Temperature (T_C) . . . . . 0°C to +85°C (C)
Supply Voltage (V_{CC}) . . . . . +4.75 to +5.25 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over COMMERCIAL Operating Range

Symbol	Parameter Description	Test Conditions	Notes	Advance Information		Unit
				Min	Max	
V_{IL}	Input Low Voltage		1	–0.5	0.8	V
V_{IH}	Input High Voltage		1	2.0	V_{CC} +0.5	V
$V_{ILINCLK}$	INCLK Input Low Voltage			–0.5	0.8	V
$V_{IHINCLK}$	INCLK Input High Voltage			2.4	V_{CC} +0.5	V
V_{OL}	Output Low Voltage for All Outputs except MEMCLK	I_{OL} = 3.2 mA			0.45	V
V_{OH}	Output High Voltage for All Outputs except MEMCLK	I_{OH} = –400 µA		2.4		V
I_{LI}	Input Leakage Current	0.45 V ≤ V_{IN} ≤ V_{CC} –0.45 V	2		±10 or +10/–200	µA
I_{LO}	Output Leakage Current	0.45 V ≤ V_{OUT} ≤ V_{CC} –0.45 V			±10	µA
I_{CCOP}	Operating Power Supply Current	V_{CC} = 5.25 V, Outputs Floating; Holding $\overline{\text{RESET}}$ active	3 4 5		175 234 280	mA mA mA
V_{OLC}	MEMCLK Output Low Voltage	I_{OLC} = 20 mA			0.6	V
V_{OHC}	MEMCLK Output High Voltage	I_{OHC} = –20 mA		V_{CC} –0.6		V

Notes:

1. *All inputs except* INCLK.
2. *The Low input leakage current is –200 µA for the following inputs:* TCK, TDI, TMS, $\overline{\text{TRST}}$, DREQ1, $\overline{\text{WAIT}}$, $\overline{\text{INTR}}$2, and $\overline{\text{INTR}}$0. *These pins have weak internal pull-up transistors.*
3. *I_{CC} measured at 12.5 MHz, V_{cc}=5.25 V, Reset Condition.*
4. *I_{CC} measured at 16.7 MHz, V_{cc}=5.25 V, Reset Condition.*
5. *I_{CC} measured at 20.0 MHz, V_{cc}=5.25 V, Reset Condition.*

CAPACITANCE

Symbol	Parameter Description	Test Conditions	Advance Information		Unit
			Min	Max	
C_{IN}	Input Capacitance	fC = 10 MHz		15	pF
C_{INCLK}	INCLK Input Capacitance			15	pF
C_{MEMCLK}	MEMCLK Capacitance			20	pF
C_{OUT}	Output Capacitance			20	pF
$C_{I/O}$	I/O Pin Capacitance			20	pF

Note:

Limits guaranteed by characterization.

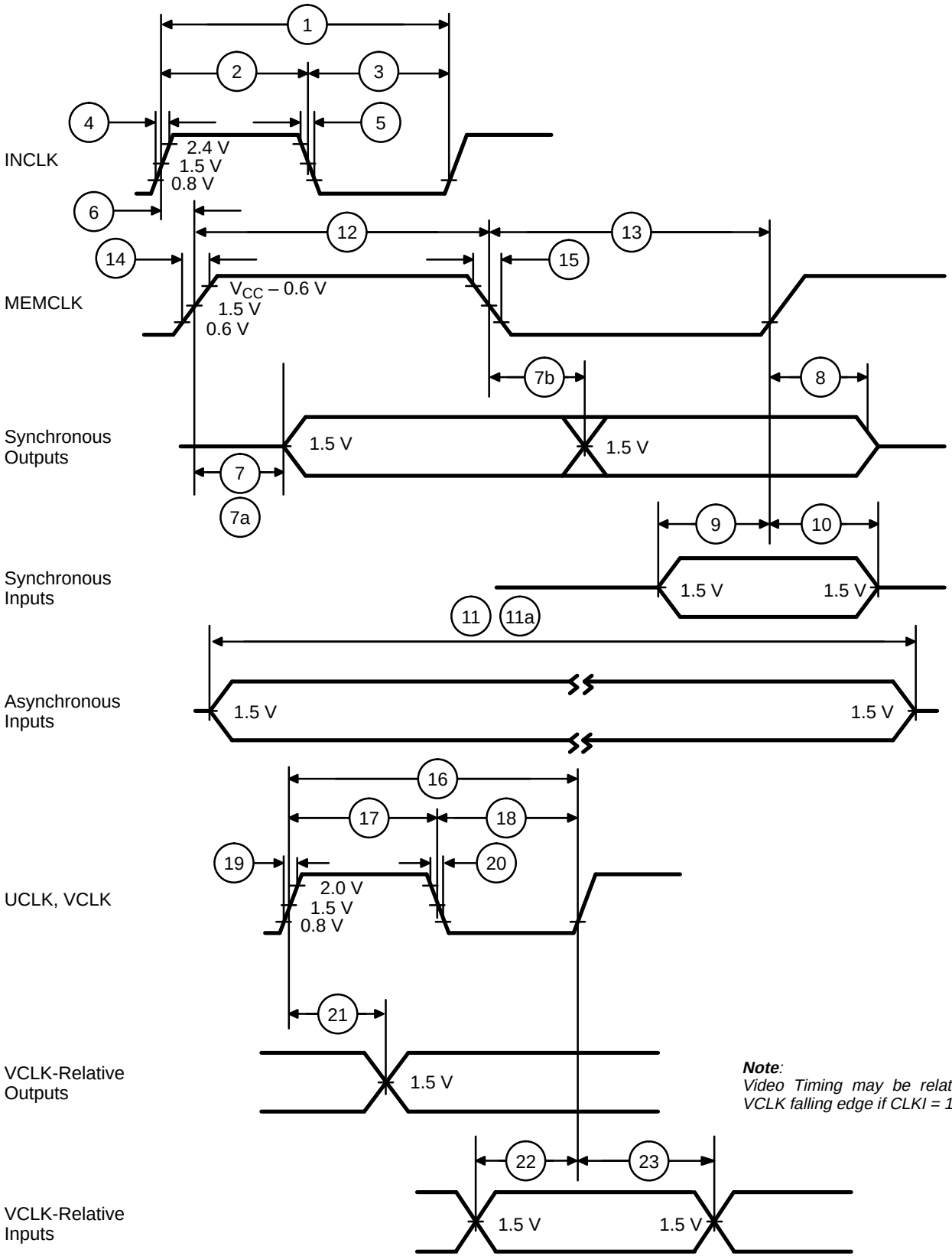
SWITCHING CHARACTERISTICS over COMMERCIAL Operating Range

No.	Parameter Description	Test Conditions (Note 1)	Advance Information						Unit
			20 MHz		16 MHz		12 MHz		
			Min	Max	Min	Max	Min	Max	
1	INCLK Period (=0.5T)	Note 2, 9	25	62.5	30	62.5	40	62.5	ns
2	INCLK High Time	Note 2	9	53.5	9	53.5	12	53.5	ns
3	INCLK Low Time	Note 2	9	53.5	9	53.5	12	53.5	ns
4	INCLK Rise Time	Note 2		4		4		4	ns
5	INCLK Fall Time	Note 2		4		4		4	ns
6	MEMCLK Delay from INCLK		0	10	0	10	0	10	ns
7	Synchronous Output Valid Delay from MEMCLK Rising Edge	Note 3a	1	11	1	11	1	15	ns
7a	Synchronous Output Valid Delay from MEMCLK Rising Edge	Note 3a	1	12	1	12	1	15	ns
7b	Synchronous Output Valid Delay from MEMCLK Falling Edge	Note 3b	1	10	1	10	1	15	ns
8	Synchronous Output Disable Delay from MEMCLK Rising Edge	Note 8	1	10	1	10	1	15	ns
9	Synchronous Input Setup Time		10		10		12		ns
10	Synchronous Input Hold Time		0		0		0		ns
11	Asynchronous Pulse Width	Note 4a, 9	4T		4T		4T		ns
11a	Asynchronous Pulse Width	Note 4b							ns
12	MEMCLK High Time	Note 5	0.5T–3	0.5T+3	0.5T–3	0.5T+3	0.5T–3	0.5T+3	ns
13	MEMCLK Low Time	Note 5	0.5T–3	0.5T+3	0.5T–3	0.5T+3	0.5T–3	0.5T+3	ns
14	MEMCLK Rise Time	Note 5	0	4	0	4	0	5	ns
15	MEMCLK Fall Time	Note 5	0	4	0	4	0	5	ns
16	UCLK, VCLK Period	Note 2	25		30		40		ns
17	UCLK, VCLK High Time	Note 2, 8	9		9		12		ns
18	UCLK, VCLK Low Time	Note 2, 8	9		9		12		ns
19	UCLK, VCLK Rise Time	Note 2		4		4		4	ns
20	UCLK, VCLK Fall Time	Note 2		4		4		4	ns
21	Synchronous Output Valid Delay from VCLK Edge	Note 6	1	15	1	15	1	20	ns
22	Input Setup Time to VCLK Edge	Note 6, 7	10		10		15		ns
23	Input Hold Time to VCLK Edge	Note 6, 7	0		0		0		ns
24	TCK Frequency			2		2		2	MHz

Notes:

1. *All outputs driving 80 pF, measured at V_{OL}=1.5 V and V_{OH}=1.5 V. For higher capacitance, add 1 ns output delay per 20 pF loading, up to 300 pF total capacitance.*
2. *INCLK, VCLK, and UCLK can be driven with TTL inputs. If not used, UCLK must be tied High.*
3. *a. Parameter 7a applies only to the outputs PIO15–PIO4 and DACK1. Parameter 7 applies to the remaining outputs.*
   b. Parameter 7b applies only to the outputs RASx, CASx, RSWE, and ROMOE. Some of these signals can also be asserted during the rising edge of MEMCLK, depending on the type of access being performed.
4. *a. Parameter 11 applies to all asynchronous inputs except LSYNC and PSYNC.*
   b. The LSYNC and PSYNC minimum width time is two bit-times. One bit-time corresponds to one internal video clock period. The internal video clock period is a function of the VCLK period and the programmed VCLK divisor.
5. *MEMCLK can drive an external load of 100 pF.*
6. *Active VCLK edge depends on the CLKI bit in the Video Control Register.*
7. *LSYNC and PSYNC may be treated as synchronous signals by meeting setup and hold times. The synchronization delay still applies.*
8. *Not production tested but guaranteed by design or characterization.*
9. *T=1 MEMCLK period, as defined by the actual frequency on the MEMCLK pin.*

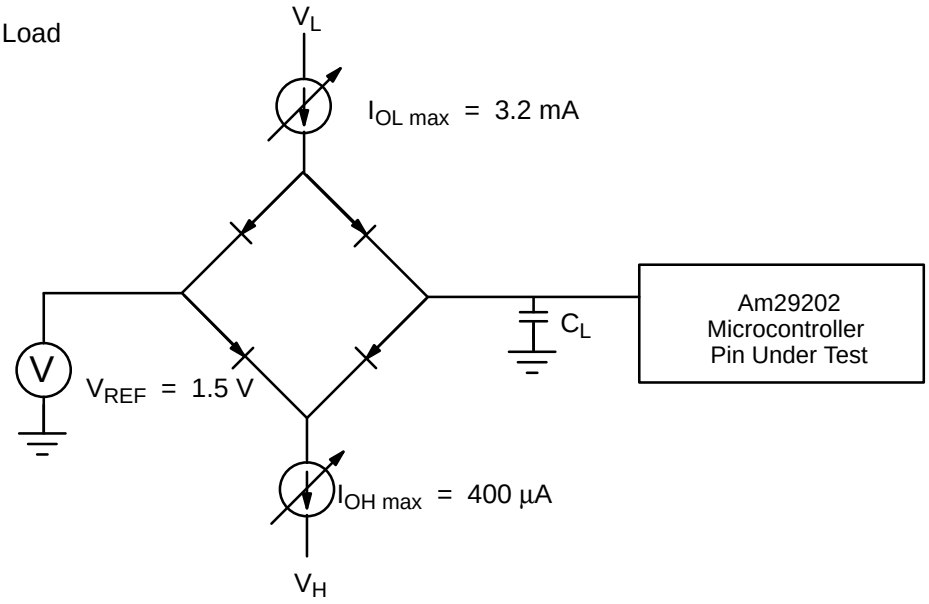
SWITCHING WAVEFORMS



***Note**:*
Video Timing may be relative to VCLK falling edge if CLKI = 1.

SWITCHING TEST CIRCUIT

Model of Dynamic Test Load



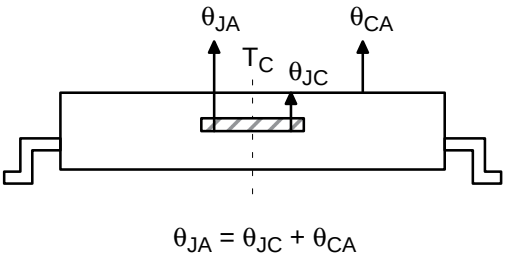
Note:

C_L is guaranteed to be a minimum 80-pF parasitic load. It represents the distributed load parasitic attributed to the test hardware and instrumentation present during production testing.

THERMAL CHARACTERISTICS

PQFP Package

The Am29202 microcontroller is specified for operation with case temperature ranges for a commercial temperature device. Case temperature is measured at the top center of the package as shown in the figure below.



$\theta_{JA} = \theta_{JC} + \theta_{CA}$

Thermal Resistance (°C/Watt)

The various temperatures and thermal resistances can be determined using the following equations along with information given in Table 12. (The variable *P* is power in watts.)

$\theta_{JA} = \theta_{JC} + \theta_{CA}$
$P = I_{CCOP} \cdot V_{CC}$

$T_J = T_C + P \cdot \theta_{JC}$
$T_J = T_A + P \cdot \theta_{JA}$

$T_C = T_J - P \cdot \theta_{JC}$
$T_C = T_A + P \cdot \theta_{CA}$

$T_A = T_J - P \cdot \theta_{JA}$
$T_A = T_C - P \cdot \theta_{CA}$

Allowable ambient temperature curves for various airflows are given in Figures 26 and 27. These graphs assume a maximum V_{CC} and a maximum power supply current equal to I_{CCOP}. All calculations made using the above information should guarantee that the operating case temperature does not exceed the maximum case temperature. Since P is a function of operating frequency, calculations can also be made to determine the ambient temperature at various operating speeds.

Table 12. PQFP Thermal Characteristics (°C/Watt) Surface Mounted

Am29202 Microcontroller		Airflow—ft./min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
θ_{JA}	Junction-to-Ambient	36	32	29	27
θ_{JC}	Junction-to-Case	8	8	8	8
θ_{CA}	Case-to-Ambient	28	24	21	19

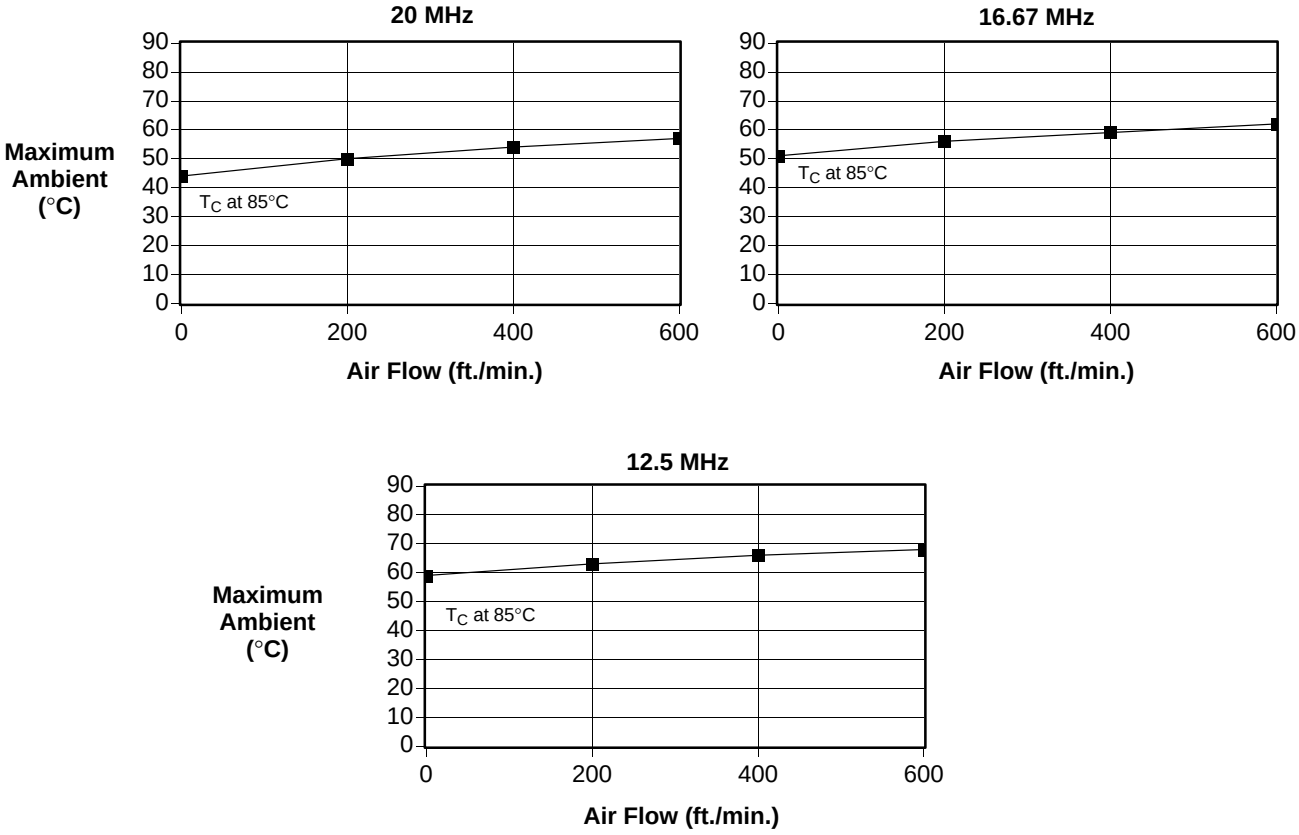


Figure 26. Maximum Allowable Ambient Temperature
(Data Sheet Limit, $I_{CCOPmax}$, V_{CC}=+5.25 V, Average Thermal Impedance)

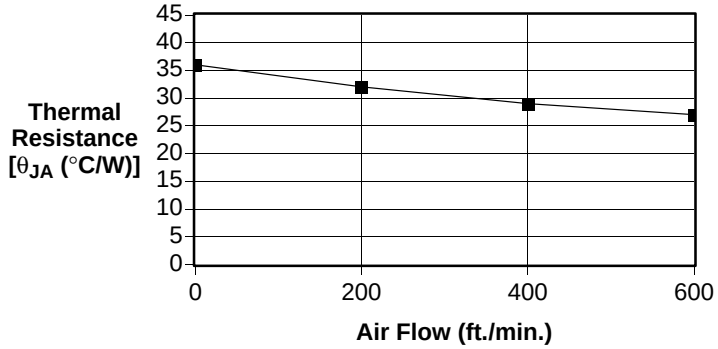
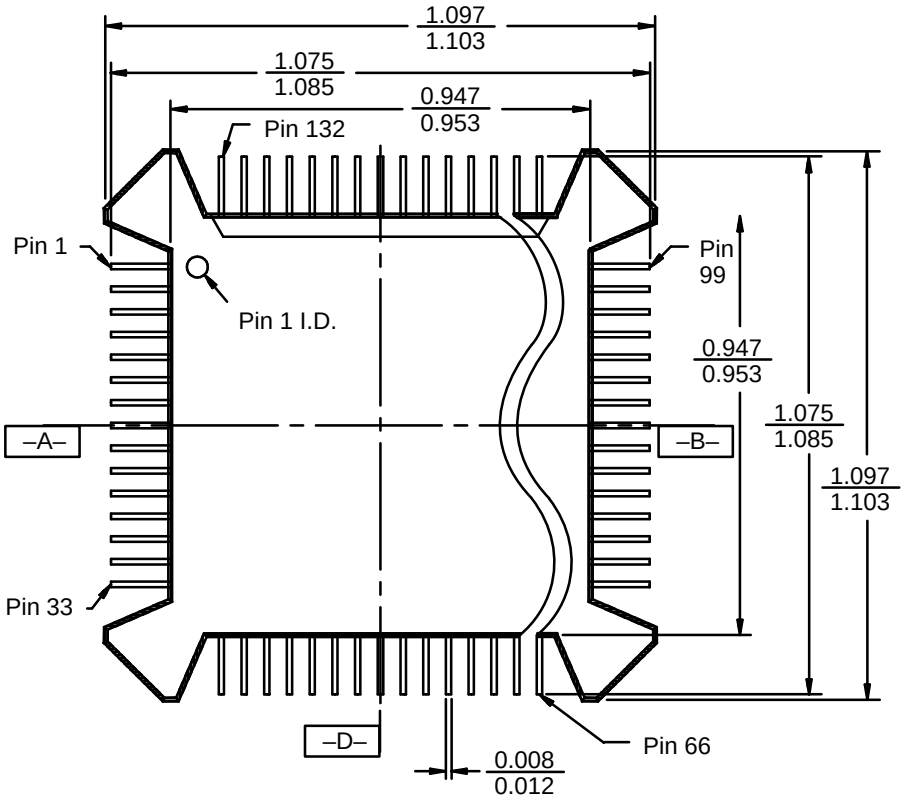
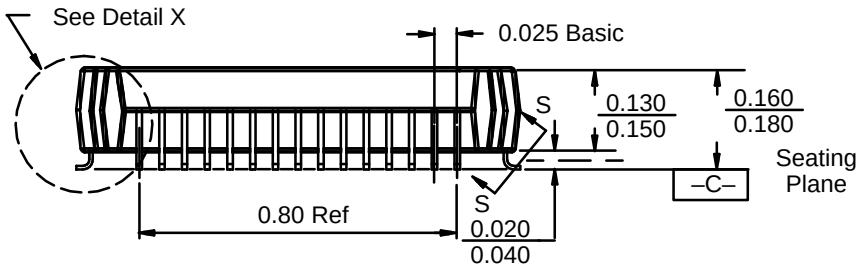


Figure 27. Thermal Impedance

PHYSICAL DIMENSIONS
PQB 132, Trimmed and Formed
Plastic Quad Flat Pack (measured in inches)



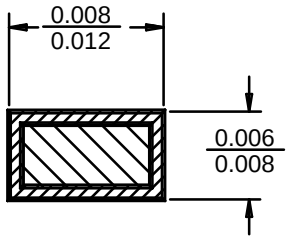
Top View



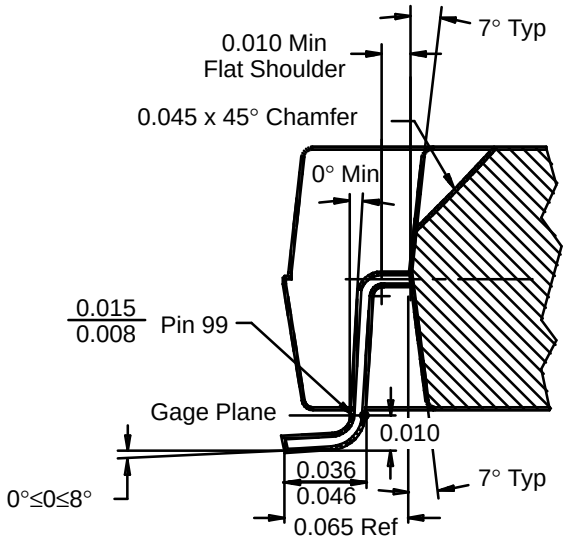
Side View

Note:
Not to scale. For reference only.

PQB 132 (continued)



Section S–S



Detail X

Note:
Not to scale. For reference only.