AMCC

S5933 PCI Matchmaker Controller Data Book

The material in this document supersedes
all previous documentation issued for any
of the products included herein.

CONTENTS

AMCC®

1.0   ARCHITECTURAL OVERVIEW

INTRODUCTION TO THE PCI LOCAL BUS

The local bus concept was developed to break the PC data bottleneck. Traditional PC bus architectures are inadequate to meet the demands of today's graphics-oriented systems and large application sizes. A local bus moves peripherals off the I/O bus and places them closer to the system's processor bus, providing faster data transfer between the processor and peripherals.

The PCI Local Bus addresses the industry's need for a local bus standard that is not directly dependent on the speed and size of the processor bus, and that is both reliable and expandable. It represents the first time in the history of the PC industry that a common bus, independent of microprocessor design and manufacturer, has been adopted and used by rival PC computer architectures. PCI offers simple "plug and play" capability for the end user, and its performance is more than adequate for the most demanding applications, such as full-motion video.

PCI's major features include:

- Processor independence
- Support for autoconfiguration, using a comprehensive set of standard configuration functions
- Multiplexed, burst mode operation
- Synchronous up to 33 MHz
- Full multimaster capability, allowing any PCI master peer-to-peer access to any PCI slave
- Hidden central arbitration
- Address and data parity
- Low pin count for cost-effective packaging
- Three physical address spaces: memory, I/O, and configuration
- Capability of full concurrency with processor/memory subsystem

Figure 1-1. S5933 Block Diagram

x

y

DEVICE SPECIFICATION

S5933   **PCI CONTROLLER**
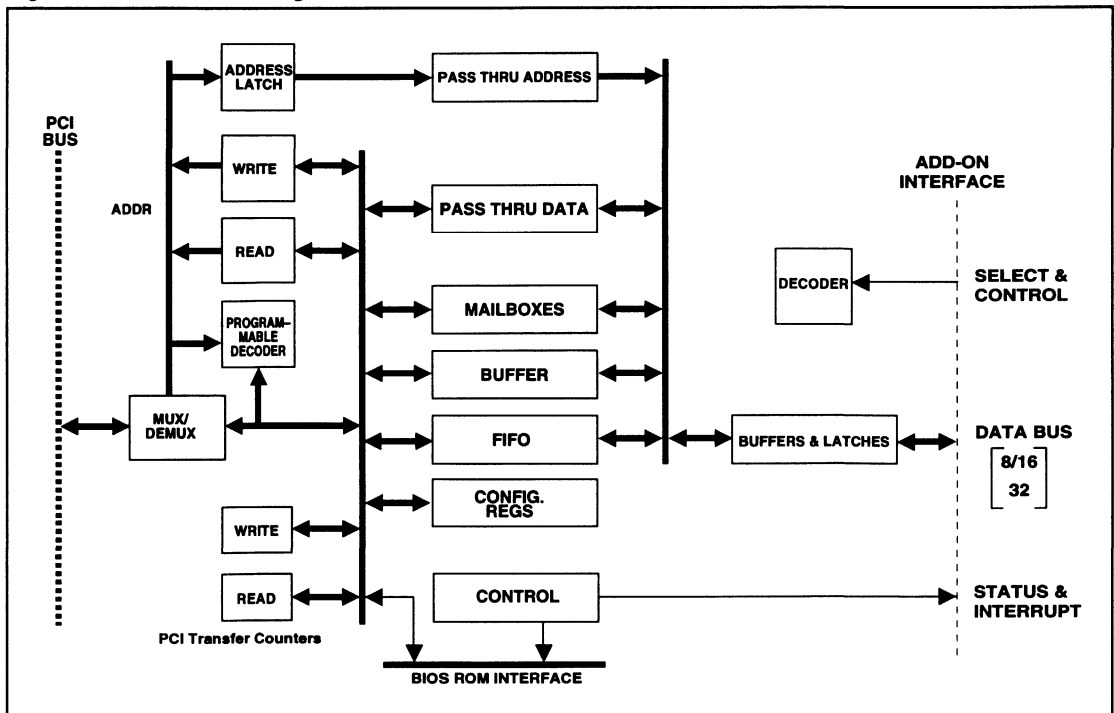
THE S5933 PCI CONTROLLER

AMCC's S5933 is a powerful and flexible PCI controller supporting several levels of interface sophistication. At the lowest level, it can serve simply as a PCI bus target with modest transfer requirements (since the PCI bus "plug-and-play" architecture is a benefit even when the transfer rate demand is low). For high-performance applications, the S5933 can become the bus master and can attain the peak transfer capabilities of 132 MByte/sec with a 32-bit PCI bus.

FUNCTIONAL ELEMENTS

The block diagram in Figure 1-1 shows the major functional elements within the S5933. The S5933 provides three physical bus interfaces: the PCI bus, the add-on bus, and an optional external non-volatile memory. Data movement can occur between the PCI bus and the add-on bus or the PCI bus and the non-volatile memory. Transfers between PCI and add-on buses can take place through the mailbox registers or the FIFOs, or can make use of the pass-thru data path. FIFO transfers on the PCI bus interface can be performed either through software control or through hardware (using the S5933 as bus master).

MAILBOXES

The mailbox registers (shown in Figure 1-2) of the S5933 provide a bidirectional data path that can be used to send data or software control information between the system platform and the add-on product. These mailboxes are intended to serve as customizable command, status, and parametric data registers. Use of the mailbox registers is defined by an application's own software; they are often used to initiate larger data transfers over either the FIFO or pass-thru data paths. Interrupts can be configured to occur on the PCI and add-on bus (if desired) based on a specific mailbox event(s).

Figure 1-2. Mailbox Register Concept

Page 1-4

Applied Micro Circuits Corporation
6195 Lusk Blvd., San Diego, CA 92121 • (619) 450-9333
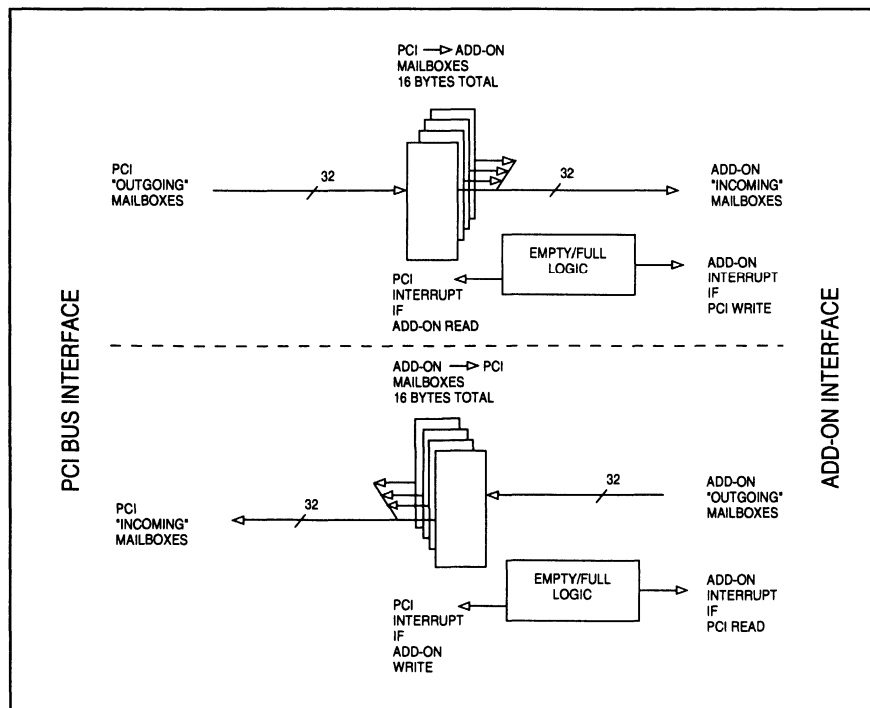
FIFOs

Two separate FIFO data paths exist within the S5933. One FIFO handles data movement from the PCI bus to the add-on bus (shown in Figure 1-3) and the other handles movement in the opposite direction (Figure 1-4). Both of the FIFOs are able to support PCI bus mastering; each has an address pointer and transfer count register associated with its PCI bus transaction. An important feature of the S5933 is the ability to optionally perform various endian translations when data is moved through the FIFO. This feature permits a single add-on product to maintain a fixed endian structure within the add-on while allowing the system platform to operate in its own native endian format.

Figure 1-3. PCI to Add-on FIFO Concept
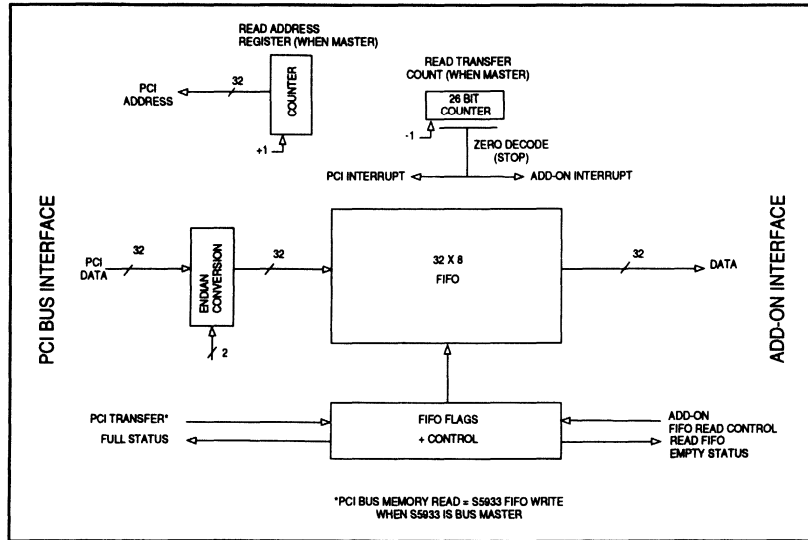
Figure 1-4. Add-on to PCI FIFO Concept
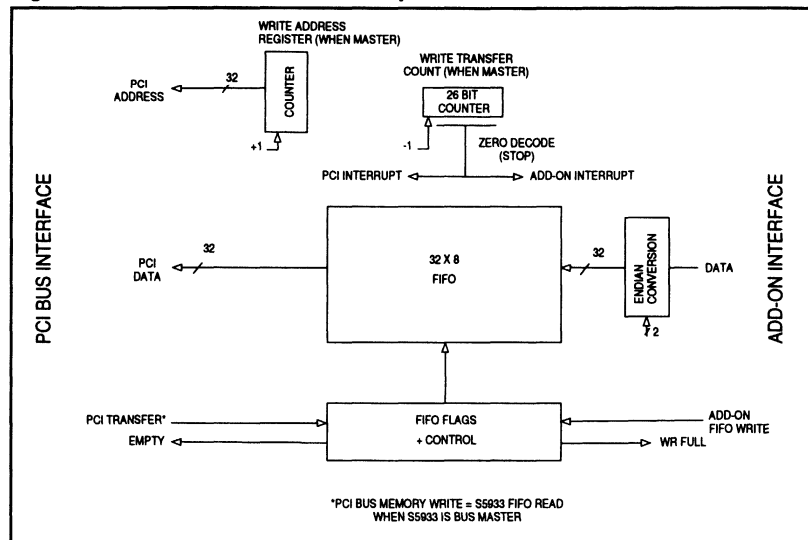
PASS-THRU

Figure 1-5 depicts the pass-thru concept that allows actual PCI bus transactions to be executed in real time with the add-on interface. The pass-thru feature can be used to achieve high-performance burst transfers between the PCI bus and add-on bus external peripherals or memory devices. In the pass-thru mode, the S5933 provides the benefits of the configuration registers and full PCI Bus Specification 2.1 compliance, while permitting customization and flexibility in the implementation of the add-on product.

The pass-thru logic is comprised of one address register and two data registers (one for each direction). Control information necessary to define the current (pass-thru) PCI bus transaction is also provided to the add-on interface on dedicated S5933 package pins.

1.1   PCI AGENT

The S5933 was designed to serve as an "endpoint" within a given PCI system. This means that it is the final point in a data transfer (for example, a video screen, network, sound output, or storage device) and/or the beginning point of a data transfer (for example, video source, network, audio source, or storage device). Some PCI elements, termed "bridges," allow for data transactions to span to other bus standards and therefore are not "endpoints."

As a PCI bus agent, the S5933 is required to support the configuration registers in order to be compliant with the PCI Specification. These configuration registers provide for a standardized method for system platform software to integrate add-on functions in a machine. Another important aspect of the standard is that all PCI bus agents can be controlled (enabled/disabled, assigned physical address space, etc.) in a common manner, regardless of function, by the hosting system. These configuration registers defined by the PCI standard (and present within the S5933) are described in Chapter 3.

Figure 1-5. Pass-Thru Concept
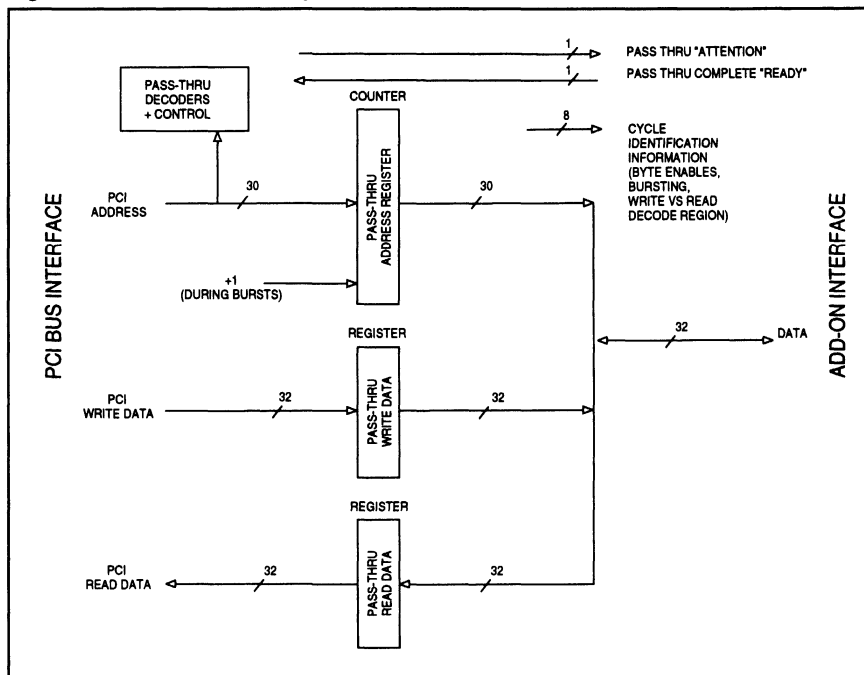
1.2 ADD-ON INTERFACE

The S5933 provides a simple, general-purpose interface to the add-on bus. The add-on data path is a 32-bit bus for the S5933. Data transfers to/from the S5933 internal registers are accomplished through a chip select decode in conjunction with either a read or write strobe. The S5933 provides dedicated pins which allow its FIFOs to be used in the implementation of custom DMA ports or additional external FIFOs (if necessary).

The output pins on the add-on interface include an interrupt source, a buffered clock, and a software-controllable reset. The interrupt output pin is provided to signal when a selected mailbox or self-test event occurs from the PCI interface. The buffered clock output is a possible add-on cost reduction feature, and in addition provides synchronization for pass-thru data transfers. The software-controllable reset from the S5933 provides add-on hardware with a means for proper handling of a system's "soft" reboot (e.g.CTRL-ALT-DEL).

1.3 NON-VOLATILE MEMORY INTERFACE

The non-volatile interface allows customization of the S5933 and provides for an optional BIOS ROM on the PCI bus. The non-volatile memory can be either a serial device or a byte-wide device. Serial devices may range from 128 bytes through 2048 bytes, and byte-wide devices from 128 bytes through 65,536 bytes.

As an example, the Vendor and Device ID numbers in the PCI configuration space (see Sections 3.1 and 3.2) can be initialized to reflect values contained in the external non-volatile memory. After initialization, a user's own Vendor and Device ID is then presented by the S5933 when requested by the PCI bus.

For some non-volatile devices, it is possible to write the non-volatile memory from the PCI interface. This feature lets the system designer establish a field upgrade strategy for the BIOS software or a method to support various system platforms with only one add-on board product.

Use of a byte-wide non-volatile memory device is shown in Figure 1-6; Figure 1-7 shows a serial device with the S5933.

Figure 1-6. Byte-Wide Non-Volatile Memory Interface with S5933
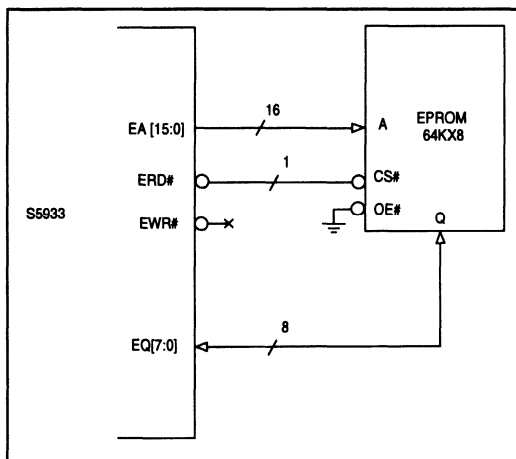
Figure 1-7. Serial Non-Volatile Memory Interface with S5933
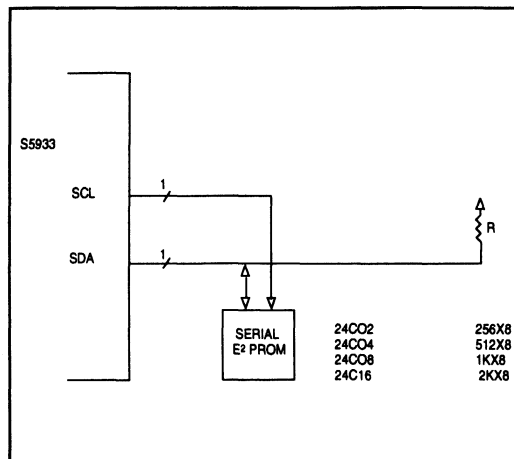
2.0   SIGNAL DESCRIPTIONS

Signal Type Definition

The following signal type definitions [in, out, t/s, s/t/s and o/d] are taken from Revision 2.1 of the PCI local bus specification.

in   Input is a standard input-only signal.

out   Totem Pole Output is a standard active driver.

t/s   Tri-State® is a bidirectional, tristate input/output pin.

s/t/s   Sustained Tri-State is an active low tristate signal owned and driven by one and only one agent at a time. The agent that drives an s/t/s pin low must drive it high for at least one clock before letting it float. A new agent cannot start driving a s/t/s signal any sooner than one clock after the previous owner tri-states it. A pullup is required to sustain the inactive state until another agent drives it, and must be provided by the central source.

o/d   Open Drain allows multiple devices to share as a wire-OR.

Note that a # symbol at the end of a signal name denotes that the active state occurs when the signal is at a low voltage. When no # symbol is present, the signal is active high.

Figure 2-1. S5933 Signal Pins
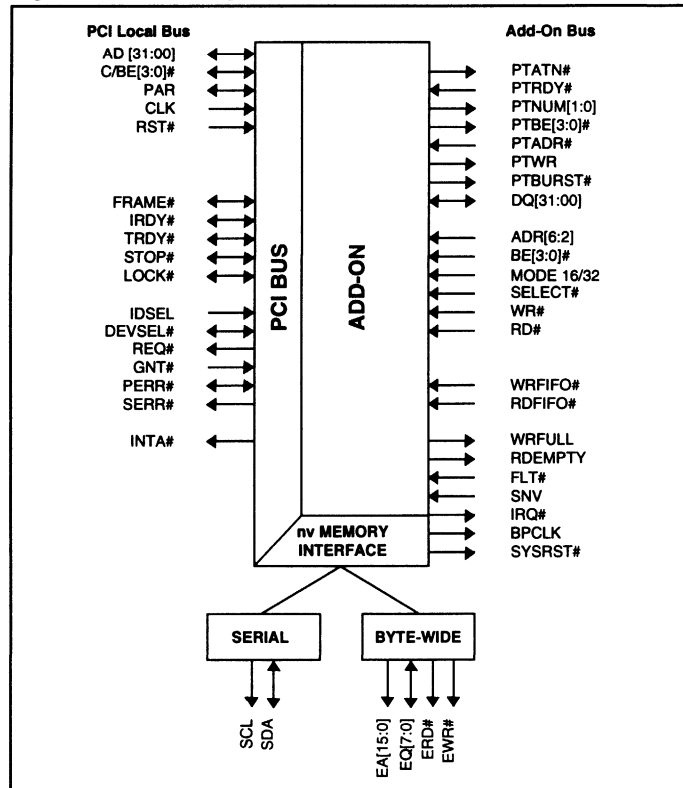
2.1   PCI BUS INTERFACE SIGNALS

2.1.1 Address and Data Pins — PCI Local Bus

Signal	Type	Description
AD[31:00]	t/s	Local Bus Address/Data lines. Address and data are multiplexed on the same pins. Each bus operation consists of an address phase followed by one or more data phases. Address phases are identified when the control signal, FRAME#, is asserted. Data transfers occur during those clock cycles in which control signals IRDY# and TRDY# are both asserted.
C/BE[3:0]#	t/s	Bus Command and Byte Enables. These are multiplexed on the same pins. During the address phase of a bus operation, these pins identify the bus command, as shown in the table below. During the data phase of a bus operation, these pins are used as Byte Enables, with C/BE[0]# enabling byte 0 (least significant byte) and C/BE[3]# enabling byte 3 (most significant byte). C/BE[3:0]# Description (during address phase) 0 0 0 0 Interrupt Acknowledge 0 0 0 1 Special Cycle 0 0 1 0 I/O READ 0 0 1 1 I/O WRITE 0 1 0 0 Reserved 0 1 0 1 Reserved 0 1 1 0 Memory Read 0 1 1 1 Memory Write 1 0 0 0 Reserved 1 0 0 1 Reserved 1 0 1 0 Configuration Read 1 0 1 1 Configuration Write 1 1 0 0 MEMORY READ - Multiple 1 1 0 1 Dual Address Cycle 1 1 1 0 Memory Read Line 1 1 1 1 Memory Write and Invalidate
PAR	t/s	Parity. This signal is even parity across the entire AD[31:00] field along with the C/BE[3:0]# field. The parity is stable in the clock following the address phase and is sourced by the master. During the data phase for write operations, the bus master sources this signal on the clock following IRDY# active; during the data phase for read operations, this signal is sourced by the target and is valid on the clock following TRDY# active. The PAR signal therefore has the same timing as AD[31:00}, delayed by one clock.

2

2.1.2 System Pins — PCI Local Bus

Signal	Type	Description
CLK	in	Clock. The rising edge of this signal is the reference upon which all other signals are based, with the exception of RST# and the interrupt (IRQA#-). The maximum frequency for this signal is 33 MHz and the minimum is DC (0 Hz).
RST#	in	Reset. This signal is used to bring all other signals within this device to a known, consistent state. All PCI bus interface output signals are not driven (tri-stated), and open drain signals such as SERR# are floated.

2.1.3 Interface Control Pins — PCI Bus Signal

Signal	Type	Description
FRAME#	s/t/s	Frame. This signal is driven by the current bus master and identifies both the beginning and duration of a bus operation. When FRAME# is first asserted, it indicates that a bus transaction is beginning and that valid addresses and a corresponding bus command are present on the AD[31:00] and C/BE[3:0] lines. FRAME# remains asserted during the data transfer portion of a bus operation and is deasserted to signify the final data phase.
IRDY#	s/t/s	Initiator Ready. This signal is sourced by the bus master and indicates that the bus master is able to complete the current data phase of a bus transaction. For write operations, it indicates that valid data is on the AD[31:00] pins. Wait states occur until both TRDY# and IRDY# are asserted together.
TRDY#	s/t/s	Target Ready. This signal is sourced by the selected target and indicates that the target is able to complete the current data phase of a bus transaction. For read operations, it indicates that the target is providing valid data on the AD[31:00] pins. Wait states occur until both TRDY# and IRDY# are asserted together.
STOP#	s/t/s	Stop. The Stop signal is sourced by the selected target and conveys a request to the bus master to stop the current transaction.
LOCK#	in	Lock. The lock signal provides for the exclusive use of a resource. The S5933 may be locked as a target by one master at a time. The S5933 cannot lock a target when it is a master. The operation of the lock function is discussed in Section 7.2.2.
IDSEL	in	Initialization Device Select. This pin is used as a chip select during configuration read or write operations.
DEVSEL#	s/t/s	Device Select. This signal is sourced by an active target upon decoding that its address and bus commands are valid. For bus masters, it indicates whether any device has decoded the current bus cycle.

2.1.4 Arbitration Pins (Bus Masters Only) — PCI Local Bus

Signal	Type	Description
REQ#	out	Request. This signal is sourced by an agent wishing to become the bus master. It is a point-to-point signal and each master has its own REQ#.
GNT#	in	Grant. The GNT# signal is a dedicated, point-to-point signal provided to each potential bus master and signifies that access to the bus has been granted.

2.1.5 Error Reporting Pins — PCI Local Bus

Signal	Type	Description
PERR#	s/t/s	Parity Error. This pin is used for reporting parity errors during the data portion of a bus transaction for all cycles except a Special Cycle. It is sourced by the agent receiving data and driven active two clocks following the detection of the error. This signal is driven inactive (high) for one clock cycle prior to returning to the tri-state condition.
SERR#	o/d	System Error. This pin is used for reporting address parity errors, data parity errors on Special Cycle commands, or any error condition having a catastrophic system impact.

2.1.6 Interrupt Pin — PCI Local Bus

Signal	Type	Description
INTA#	o/d	Interrupt A. This pin is a level sensitive, low active interrupt to the host. The INTA# interrupt must be used for any single function device requiring an interrupt capability.

2.2   NON-VOLATILE MEMORY INTERFACE SIGNALS

This signal grouping provides for connection to external non-volatile memories. Either a serial or byte-wide device may be used.

The serial interface shares the read and write control pins used for interfacing with byte-wide memory devices. Since it is intended that only one (serial or byte wide) configuration be used in any given implementation, separate descriptions are provided for each (Section 2.2.1 and 2.2.2 below). The S5933 provides the pins necessary to interface to a byte wide non-volatile memory. When they are connected to a properly configured serial memory, these byte wide interface pins assume alternate functions. These alternate functions include added external FIFO status flags, FIFO reset control, add-on control for bus mastering and a hardware interface mailbox port.

2.2.1   Serial nv Devices

Signal	Type	Description
SCL	out	Serial Clock. This output is intended to drive a two-wire Serial Interface and functions as the bus's master. It is intended that this signal be directly connected to one or more inexpensive serial non-volatile RAMs or EEPROMs. This pin is shared with the byte wide interface signal, ERD#.
SDA	t/s	Serial Data/Address. This bidirectional pin is used to transfer addresses and data to or from a serial nvRAM or EEPROM. This pin is shared with the byte wide interface signal, EWR#. This pin has an internal pull-up resistor.
SNV	in	Serial Non-Volatile Device. This input, when high, indicates a serial boot device or no boot device is present. When this pin is low, a byte-wide boot device is present. This pin has an internal pull-up resistor.

2.2.2   Byte-Wide nv Devices

Signal	Type	Description
EA[15:00]	t/s	External nv memory address. These signals connect directly to the external BIOS (or EEPROM) or EPROM address pins EA0 through EA15. The PCI interface controller assembles 32-bit-wide accesses through multiple read cycles of the 8-bit device. The address space from 0040h through 007Fh is used to preload and initialize the PCI configuration registers. Should an external nv memory be used, the minimum size required is 128 bytes and the maximum is 64K bytes. When a serial memory is connected to the S5933, the pins EA[7:0] are reconfigured to become a hardware add-on to PCI mailbox register with the EA8 pin as the mailbox load clock. Also, the EA15 signal pin will provide an indication that the PCI to add-on FIFO is full (FRF), and the EA14 signal pin will indicate whether the add-on to PCI FIFO is empty (FWE). EA[9:0] have internal pull-up resistors.
ERD#	out	External nv memory read control. This pin is asserted during read operations involving the external non-volatile memory. Data is transferred into the S5933 during the low to high transition of ERD#. This pin is shared with the serial external memory interface signal, SCL.
EWR#	t/s	External nv memory write control. This pin is asserted during write operations involving the external non-volatile memory. Data is presented on pins EQ[7:0] along with its address on pins EA[15:0] throughout the entire assertion of EWR#. This pin is shared with the serial external memory interface signal, SDA.
EQ[7:0]	t/s	External memory data bus. These pins are used to directly connect with the data pins of an external non-volatile memory. When a serial memory is connected to the S5933, the pins EQ4, EQ5, EQ6 and EQ7 become reconfigured to provide signal pins for bus mastering control from the add-on interface (see Section 10.2.3.3). EQ[7:6] have internal pull-down resistors. EQ[5:0] have internal pull-up resistors.

2.3 ADD-ON BUS INTERFACE SIGNALS

The following sets of signals represent the interface pins available for the add-on function. There are four groups: Register access, FIFO access, Pass-thru mode pins, and general system pins.

2.3.1 Register Access Pins

Signal	Type	Description
DQ[31:00]	t/s	Datapath DQ0–DQ31. These pins represent the datapath for the add-on peripheral's data bus. They provide the interface to the controller's FIFO and other registers. When MODE=V_{CC}, only DQ[15:00] are used. DQ[31:0] have internal pull-up resistors.
ADR[6:2]	in	Add-on Addresses. These signals are the address lines to select which of the 16 DWORD registers within the controller is desired for a given read or write cycle, as shown in the table below. (Refer to Chapter 5 for detailed descriptions of the registers.) ADR6 has an internal pull-down resistor. **ADR[6:2]** — **Register Name** 0 0 0 0 0 — Add-on Incoming Mailbox Reg. 1 0 0 0 0 1 — Add-on Incoming Mailbox Reg. 2 0 0 0 1 0 — Add-on Incoming Mailbox Reg. 3 0 0 0 1 1 — Add-on Incoming Mailbox Reg. 4 0 0 1 0 0 — Add-on Outgoing Mailbox Reg. 1 0 0 1 0 1 — Add-on Outgoing Mailbox Reg. 2 0 0 1 1 0 — Add-on Outgoing Mailbox Reg. 3 0 0 1 1 1 — Add-on Outgoing Mailbox Reg. 4 0 1 0 0 0 — Add-on FIFO Port 0 1 0 0 1 — Bus Master Write Address Register 0 1 0 1 0 — Add-on Pass-thru Address 0 1 0 1 1 — Add-on Pass-thru Data 0 1 1 0 0 — Bus Master Read Address Register 0 1 1 0 1 — Add-on Mailbox Empty/Full Status 0 1 1 1 0 — Add-on Interrupt Control 0 1 1 1 1 — Add-on General Control/Status Register 1 0 1 1 0 — Bus Master Write Transfer Count 1 0 1 1 1 — Bus Master Read Transfer Count
BE3# or ADR1	in	Byte Enable 3 (32-bit mode) or ADR1 (16 bit mode). This pin is used in conjunction with the read or write strobes (RD# or WR#) and the add-on select signal, SELECT#. As a Byte Enable, it is necessary to have this pin asserted to perform write operations to the register identified by ADR[6:2] bit locations d24 through d31; for read operations it controls the DQ[31:24] output drive. BE3# has an internal pull-up resistor.
BE[2:0]#	in	Byte Enable 2 through 0. These pins provide for individual byte control during register read or write operations. BE2# controls activity over DQ[23:DQ16], BE1# controls DQ[15:8], and BE0# controls DQ[7:0]. During read operations they control the output drive for each of their respective byte lanes; for write operations they serve as a required enable to perform the modification of each byte lane. BE[2:0]# have internal pull-up resistors.
SELECT#	in	Select for the add-on interface. This signal must be driven low for any write or read access to the add-on interface registers. This signal must be stable during the assertion of command signals WR# or RD#.
WR#	in	Write strobe. This pin, when asserted in conjunction with the SELECT# pin, causes the writing of one of the internal registers. The specific register and operand size are identified through address pins ADR[6:2] and the byte enables, BE[3:0]#.
RD#	in	Read strobe. This pin, when asserted in conjunction with the SELECT# pin, causes the reading of one of the internal registers. The specific register and operand size are identified through address pins ADR[6:2] and the byte enables BE[3:0]#.
MODE	in	This pin control whether the S5933 data accesses on the DQ bus are to be 32-bits wide (MODE = low) or 16-bits wide (MODE = high). When in the 16 bit mode, the signal BE3# is reassigned as the address signal ADR1. MODE has an internal pull-down resistor.

2.3.2 FIFO Access Pins

Signal	Type	Description
WRFIFO#	in	Write FIFO. This signal provides a method to directly write the FIFO without having to generate the SELECT# signal or the ADR[6:2] value of [01000b] to access the FIFO. Access width is either 32 bits or 16 bits depending on the data bus size available. This signal is intended for implementing PCI DMA transfers with the add-on system. This pin has an internal pull-up resistor.
RDFIFO#	in	Read FIFO. This signal provides a method to directly read the FIFO without having to generate the SELECT# signal or the ADR[6:2] value of [01000b] to access the FIFO. Access width is either 32 bits or 16 bits, depending on the data bus size defined by the MODE pin. This signal is intended for implementing PCI DMA transfers with the add-on system. This pin has an internal pull-up resistor.
WRFULL	out	Write FIFO full. This pin indicates whether the add-on-to-PCI bus FIFO is able to accept more data. This pin is intended to be used to implement DMA hardware on the add-on system bus. A logic low output from this pin can be used to represent a DMA write (Add-on to-PCI FIFO) request.
RDEMPTY	out	Read FIFO Empty. This pin indicates whether the read FIFO (PCI-to-add-on FIFO) contains data. This pin is intended to be used by the add-on system to control DMA transfers from the PCI bus to the add-on system bus. A logic low from this pin can be used to represent a DMA (PCI-to-add-on FIFO) request.

2.3.3 Pass-thru Interface Pins

Signal	Type	Description
PTATN#	out	Pass-thru Attention. This signal identifies that an active PCI bus cycle has been decoded and data must be read from or written to the Pass-Thru Data Register.
PTBURST#	out	Pass-thru Burst. This signal identifies PCI bus operations involving the current pass-thru cycle as requesting burst access.
PTRDY#	in	Pass-thru Ready. This input indicates when add-on logic has completed a pass-thru cycle and another may be initiated.
PTNUM[1:0]	out	Pass-thru Number. These signals identify which of the four base address registers decoded a pass-thru bus activity. These bits are only meaningful when signal PTATN# is active. A value of 00 corresponds to Base Address Register 1, a value of 01 for Base Address Register 2, and so on.
PTBE[3:0]#	out	Pass-thru Byte Enables. These signals indicate which bytes are requested for a given pass-thru operation. They are valid during the presence of signal PTATN# active.
PTADR#	in	Pass-thru Address. This signal causes the actual pass-thru requested address to be presented as outputs on the DQ pins DQ[31:0] for add-ons with 32-bit buses, or the low-order 16 bits for add-ons with 16-bit buses. It is necessary that all other bus control signals be in their inactive state during the assertion of PTADR#. The purpose of this signal is to provide the direct addressing of external add-on peripherals through use of the PTNUM[1:0] and the low-order address bits presented on the DQ bus with this pin active.
PTWR	out	Pass-thru Write. This signal identifies whether a pass-thru operation is a read or write cycle. This signal is valid only when PTATN# is active.

2.3.4 System Pins

Signal	Type	Description
SYSRST#	out	System Reset. This low active output is a buffered form of the PCI bus reset, RST#. It is not synchronized to any clock within the PCI interface controller. Additionally, this signal can be invoked through software from the PCI host interface.
BPCLK	out	Buffered PCI Clock. This output is a buffered form of the PCI bus clock and, as such, has all of the behavioral characteristics of the PCI clock (i.e., DC-to-33 MHz capability).
IRQ#	out	Interrupt. This pin is used to signal the add-on system that a significant event has occurred as a result of activity within the PCI controller.
FLT#	in	Float. When asserted, all S5933 outputs are floated. This pin has an internal pull-up resistor.

PCI Configuration Space Header

31 24	23 16	15 8	7 00	
DEVICE ID		VENDOR ID		00
STATUS		COMMAND		04
CLASS CODE			REV ID	08
BIST	HEADER TYPE = 0	LATENCY TIMER	CACHE LINE SIZE	0C
	BASE ADDRESS REGISTER #0			10
	BASE ADDRESS REGISTER #1			14
	BASE ADDRESS REGISTER #2			18
	BASE ADDRESS REGISTER #3			1C
	BASE ADDRESS REGISTER #4			20
	BASE ADDRESS REGISTER #5			24
RESERVED = 0's				28
RESERVED = 0's				2C
EXPANSION ROM BASE ADDRESS				30
RESERVED = 0's				34
RESERVED = 0's				38
MAX_LAT	MIN_GNT	INTERRUPT PIN	INTERRUPT LINE	3C

LEGEND

- EPROM IS DATA SOURCE (READ ONLY)

- CONTROL FUNCTION

- EPROM INITIALIZED RAM
  (CAN BE ALTERED FROM PCI PORT)

- EPROM INITIALIZED RAM
  (CAN BE ALTERED FROM ADD-ON PORT)

- HARD-WIRED TO ZEROES

Note: Some registers are a combination of the above.
        See individual sections for full description.

3.0   PCI CONFIGURATION REGISTERS

Each PCI bus device contains a unique 256-byte region called its configuration header space. Portions of this configuration header are mandatory in order for a PCI agent to be in full compliance with the PCI specification. This section describes each of the configuration space fields—its address, default values, initialization options, and bit definitions—and also provides an explanation of its intended usage.

Table 3-1. Configuration Registers

Configuration Address Offset	Abbreviation	Register Name	Section #
00h–01h	VID	Vendor Identification	3.1
02h–03h	DID	Device Identification	3.2
04h–05h	PCICMD	PCI Command Register	3.3
06h–07h	PCISTS	PCI Status Register	3.4
08h	RID	Revision Identification Register	3.5
09h–0Bh	CLCD	Class Code Register	3.6
0Ch	CALN	Cache Line Size Register	3.7
0Dh	LAT	Master Latency Timer	3.8
0Eh	HDR	Header Type	3.9
0Fh	BIST	Built-in Self-test	3.10
10h–27h	BADR0-BADR5	Base Address Registers (0-5)	3.11
28h–2Fh	—	Reserved	
30h	EXROM	Expansion ROM Base Address	3.12
34h–3Bh	—	Reserved	
3Ch	INTLN	Interrupt Line	3.13
3Dh	INTPIN	Interrupt Pin	3.14
3Eh	MINGNT	Minimum Grant	3.15
3Fh	MAXLAT	Maximum Latency	3.16
40h–FFh	—	Not used	

3.1   VENDOR IDENTIFICATION REGISTER (VID)

Register Name: Vendor Identification

Address Offset: 00h-01h

Power-up value: 10E8h (AMCC, Applied Micro Circuits Corp.)

Boot-load: External nvRAM offset 040h-41h

Attribute: Read Only (RO)

Size: 16 bits

The VID register contains the vendor identification number. This number is assigned by the PCI Special Interest Group and is intended to uniquely identify any PCI device. Write operations from the PCI interface have no effect on this register. After reset is removed, this field can be boot-loaded from the external non-volatile device (if present and valid) so that other legitimate PCI SIG members can substitute their vendor identification number for this field. Sections 6.2 and 6.3 detail the optional boot-load operation.

Figure 3-1. Vendor Identification Register

```
 15                                                                          0
   ┌──────────────────────────────────────────────────────────────────────┐
   │                              10E8h                                     │
   └────────────────────────────────┬───────────────────────────────────────┘
                                     └────────── Vendor Identification Register (RO)
```
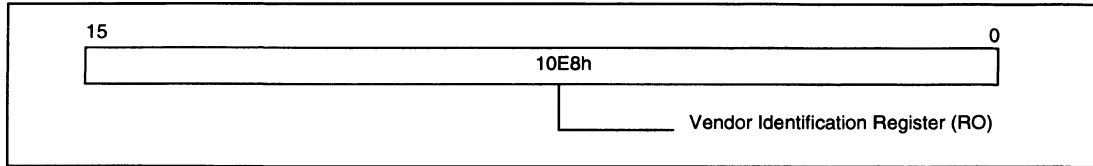
Table 3-2. Vendor Identification Register

Bit	Description
15:0	Vendor Identification Number: This is a 16 bit-value assigned to AMCC.

3.2 DEVICE IDENTIFICATION REGISTER (DID)

Register Name:	Device Identification
Address Offset:	02h-03h
Power-up value:	4750h (ASCII hex for 'GP', General Purpose)
Boot-load:	External nvRAM offset 042h-43h
Attribute:	Read Only
Size:	16 bits

The DID register contains the vendor-assigned device identification number. This number is generated by AMCC in compliance with the conditions of the PCI specification. Write operations from the PCI interface have no effect on this register. After reset is removed, this field can be boot-loaded from the external non-volatile device (if present and valid) so that other legitimate PCI SIG members can substitute their own device identification number for this field. Sections 6.2 and 6.3 detail the optional boot-load operation.

Figure 3-2. Device Identification Register
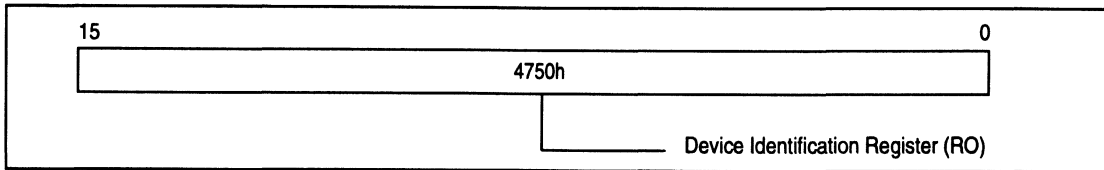
Table 3-3. Device Identification Register

Bit	Description
15:0	Device Identification Number: This is a 16-bit value initially assigned by AMCC for applications using the AMCC Vendor ID.

3.3 PCI COMMAND REGISTER (PCICMD)

Register Name:	PCI Command
Address Offset:	04h-05h
Power-up value:	0000h
Boot-load:	not used
Attribute:	Read/Write (R/W on 6 bits, Read Only for all others)
Size:	16 bits

This 16-bit register contains the PCI Command. The function of this register is defined by the PCI specification and its implementation is required of all PCI devices. Only six of the ten fields are used by this device; those which are not used are hardwired to 0. The definitions for all fields are provided here for completeness.

Figure 3-3. PCI Command Register
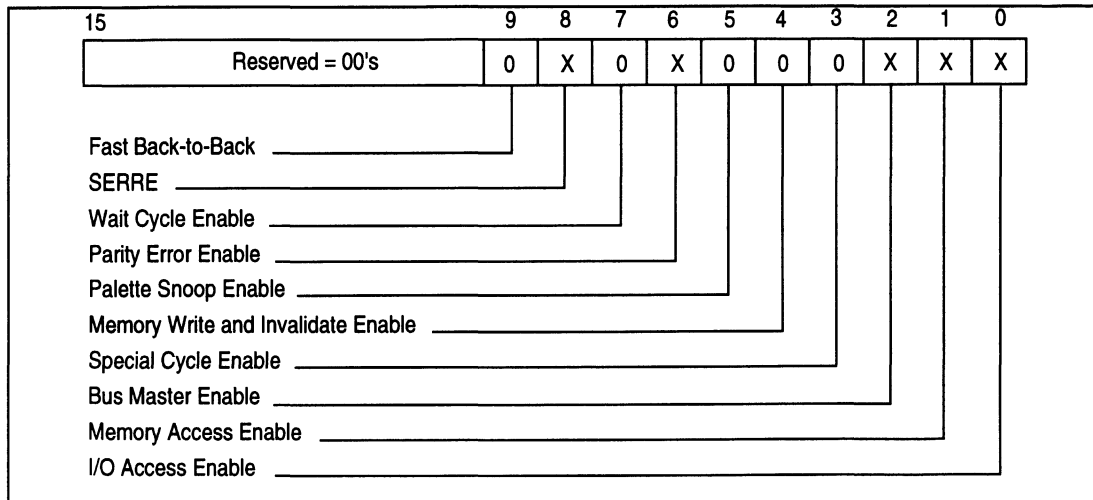
Table 3-4. PCI Command Register

Bit	Description
15:10	Reserved. Equals all 0's.
9	Fast Back-to-Back Enable. The S5933 does not support this function. This bit must be set to zero. This bit is cleared to a 0 upon RESET#.
8	System Error Enable. When this bit is set to 1, it permits the S5933 controller to drive the open drain output pin, SERR#. This bit is cleared to 0 upon RESET#. The SERR# pin driven active normally signifies a parity error on the address/control bus.
7	Wait Cycle Enable. This bit controls whether this device does address/data stepping. Since the S5933 controller never uses stepping, it is hardwired to 0.
6	Parity Error Enable. This bit, when set to a one, allows this controller to check for parity errors. When a parity error is detected, the PCI bus signal PERR# is asserted. This bit is cleared (parity testing disabled) upon the assertion of RESET#.
5	Palette Snoop Enable. This bit is not supported by the S5933 controller and is hardwired to 0. This feature is used solely for PCI-based VGA devices.
4	Memory Write and Invalidate Enable. This bit allows certain Bus Master devices to use the Memory Write and Invalidate PCI bus command when set to 1. When set to 0, masters must use the Memory Write command instead. The S5933 controller does not support this command when operated as a master and therefore it is hardwired to 0.
3	Special Cycle Enable. Devices which are capable of monitoring special cycles can do so when this bit is set to 1. The S5933 controller does not monitor (or generate) special cycles and this bit is hardwired to 0.
2	Bus Master Enable. This bit, when set to a one, allows the S5933 controller to function as a bus master. This bit is initialized to 0 upon the assertion of signal pin RESET#.
1	Memory Space Enable. This bit allows the S5933 controller to decode and respond as a target for memory regions that may be defined in one of the five base address registers. (See Section 3.11.) This bit is initialized to 0 upon the assertion of signal pin RESET#.
0	I/O Space Enable. This bit allows the S5933 controller to decode and respond as a target to I/O cycles which are to regions defined by any one of the five base address registers. (See Section 3.11). This bit is intialized to 0 upon the assertion of signal pin RESET#.

3.4   PCI STATUS REGISTER (PCISTS)

Register Name: PCI Status

Address Offset: 06h-07h

Power-up value: 0080h

Boot-load: not used

Attribute: Read Only (RO), Read/Write
                      Clear (R/WC)

Size: 16 bits

This 16-bit register contains the PCI status information. The function of this register is defined by the PCI specification and its implementation is required of all PCI devices. Only some of the bits are used by this device; those which are not used are hardwired to 0. Most status bits within this register are designated as "write clear," meaning that in order to clear a given bit, the bit must be written as a 1. All bits written with a 0 are left unchanged. These bits are identified in Figure 3-4 as (R/WC). Those which are Read Only are shown as (RO) in Figure 3-4.

Figure 3-4. PCI Status Register
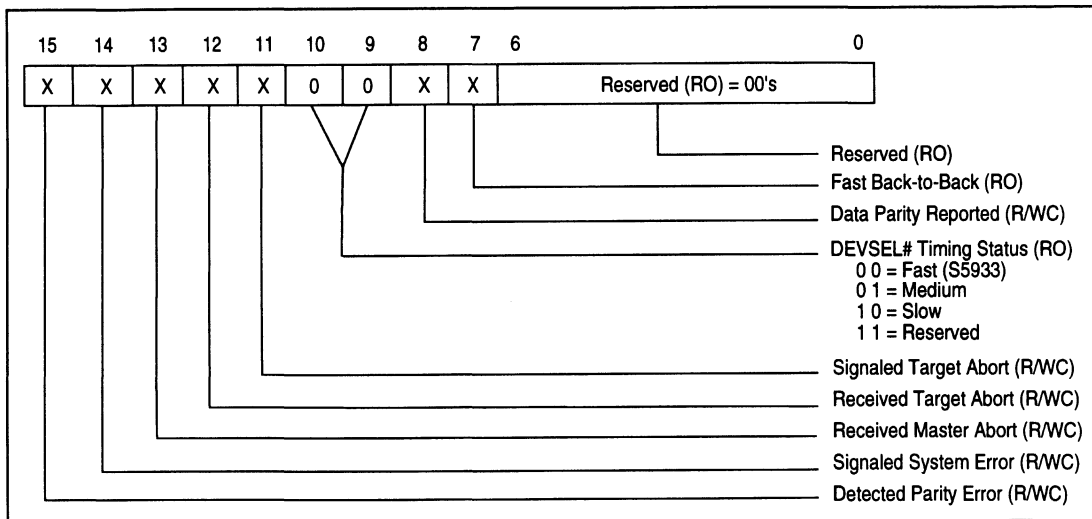
Table 3-5. PCI Status Register

Bit	Description
15	Detected Parity Error. This bit is set whenever a parity error is detected. It functions independently from the state of Command Register Bit 6. This bit may be cleared by writing a 1 to this location.
14	Signaled System Error. This bit is set whenever the device asserts the signal SERR#. This bit can be reset by writing a 1 to this location.
13	Received Master Abort. This bit is set whenever a bus master abort occurs. See Section 7.1.4.3 for further definition of a master abort. This bit can be reset by writing a 1 to this location.
12	Received Target Abort. This bit is set whenever this device has one of its own initiated cycles terminated by the currently addressed target. This bit can be reset by writing a 1 to this location.
11	Signaled Target Abort. This bit is set whenever this device aborts a cycle when addressed as a target. This bit can be reset by writing a 1 to this location.
10:9	Device Select Timing. These bits are read-only and define the signal behavior of DEVSEL# from this device when accessed as a target.
8	Data Parity Reported. This bit is set upon the detection of a data parity error for a transfer involving the S5933 device as the master. The Parity Error Enable bit (D6 of the Command Register) must be set in order for this bit to be set. Once set, it can only be cleared by either writing a 1 to this location or by the assertion of the signal RESET#.
7	Fast Back-to-back Capable. When equal to 1, this indicates that the device can accept fast back-to-back cycles as a target.
6:0	Reserved. Equal all 0's.

3

3.5 REVISION IDENTIFICATION REGISTER (RID)

Register Name: Revision Identification

Address Offset: 08h

Power-up value: 00h

Boot-load: External nvRAM/EPROM offset 048h

Attribute: Read Only

Size: 8 bits

The RID register contains the revision identification number. This field is initially cleared. Write operations from the PCI interface have no effect on this register. After reset is removed, this field can be boot-loaded from the external non-volatile device (if present and valid) so that another value may be used. Sections 6.2 and 6.3 detail the optional boot-load operation.

Figure 3-5. Revision Identification Register
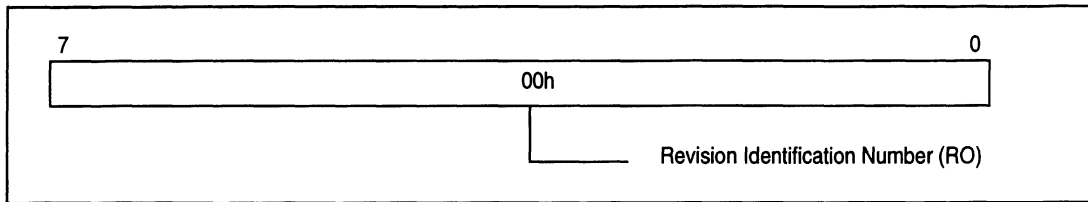
Table 3-6. Revision Identification Register

Bit	Description
7:0	Revision Identification Number. Initialized to zeros, this register may be loaded to the value in non-volatile memory at offset 048h.

3.6   CLASS CODE REGISTER (CLCD)

Register Name: Class Code
Address Offset: 09h-0Bh
Power-up value: FF0000h
Boot-load: External nvRAM offset
                    049h-4Bh
Attribute: Read Only
Size: 24 bits

This 24-bit, read-only register is divided into three one-byte fields: the base class resides at location 0Bh, the sub-class at 0Ah, and the programming interface at 09h. The default setting for the base class is all ones (FFh), which indicates that the device does not fit into the thirteen base classes defined in the PCI Local Bus Specification. It is possible, however, through use of the external non-volatile memory, to implement one of the defined class codes described in Table 3-7 below.

For devices that fall within the seven defined class codes, sub-classes are also assigned. Tables 3-8 through 3-14 describe each of the sub-class codes for base codes 00h through 0Ch, respectively.

Figure 3-6. Class Code Register

@0Bh		@0Ah		@09h	(Offset)
7 0	7	0	7	0	(Bit)
Base Class		Sub-Class		Prog I/F	
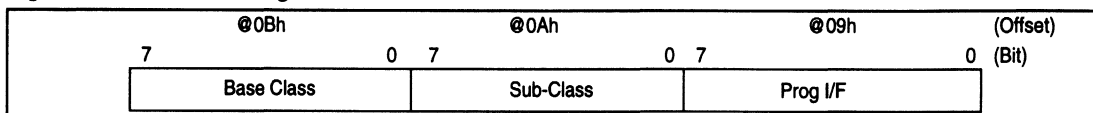
Table 3-7. Defined Base Class Codes

Base-Class	Description
00h	Early, pre-2.0 PCI specification devices
01h	Mass storage controller
02h	Network controller
03h	Display controller
04h	Multimedia device
05h	Memory controller
06h	Bridge device
07h	Simple communication controller
08h	Base system peripherals
09h	Input devices
0Ah	Docking stations
0Bh	Processors
0Ch	Serial bus controllers
0D-FEh	Reserved
FFh	Device does not fit defined class codes (default)

Table 3-8. Base Class Code 00h: Early, Pre-2.0 Specification Devices

Sub-Class	Prog I/F	Description
00h	00h	All devices other than VGA
01h	00h	VGA-compatible device

Table 3-9. Base Class Code 01h: Mass Storage Controllers

Sub-Class	Prog I/F	Description
00h	00h	SCSI controller
01h	xxh	IDE controller
02h	00h	Floppy disk controller
03h	00h	IPI controller
04h	00h	RAID controller
80h	00h	Other mass storage controller

Table 3-10. Base Class Code 02h: Network Controllers

Sub-Class	Prog I/F	Description
00h	00h	Ethernet controller
01h	00h	Token ring controller
02h	00h	FDDI controller
03h	00h	ATM controller
80h	00h	Other network controller

Table 3-11. Base Class Code 03h: Display Controllers

Sub-Class	Prog I/F	Description
00h	00h	VGA-compatible controller
00h	01h	8514 compatible controller
01h	00h	XGA controller
80h	00h	Other display controller

Table 3-12. Base Class Code 04h: Multimedia Devices

Sub-Class	Prog I/F	Description
00h	00h	Video device
01h	00h	Audio device
80h	00h	Other multimedia device

Table 3-13. Base Class Code 05h: Memory Controllers

Sub-Class	Prog I/F	Description
00h	00h	RAM memory controller
01h	00h	Flash memory controller
80h	00h	Other memory controller

Table 3-14. Base Class Code 06h: Bridge Devices

Sub-Class	Prog I/F	Description
00h	00h	Host/PCI bridge
01h	00h	PCI/ISA bridge
02h	00h	PCI/EISA bridge
03h	00h	PCI/Micro Channel bridge
04h	00h	PCI/PCI bridge
05h	00h	PCI/PCMCIA bridge
06h	00h	NuBus bridge
07h	00h	CardBus bridge
80h	00h	Other bridge type

Table 3-15. Base Class Code 07h: Simple Communications Controllers

Sub-Class	Prog I/F	Description
00h	00h	Generic XT compatible serial controller
	01h	16450 compatible serial controller
	02h	16550 compatible serial controller
01h	00h	Parallel port
	01h	Bidirectional parallel port
	02h	ECP 1.X compliant parallel port
80h	00h	Other communications device

Table 3-16. Base Class Code 08h: Base System Peripherals

Sub-Class	Prog I/F	Description
00h	00h	Generic 8259 PIC
	01h	ISA PIC
	02h	EISA PIC
01h	00h	Generic 8237 DMA controller
	01h	ISA DMA controller
	02h	EISA DMA controller
02h	00h	Generic 8254 system timer
	01h	ISA system timer
	02h	EISA system timers (2 timers)
03h	00h	Generic RTC controller
	01h	ISA RTC controller
80h	00h	Other system peripheral

Table 3-17. Base Class Code 09h: Input Devices

Sub-Class	Prog I/F	Description
00h	00h	Keyboard controller
01h	00h	Digitizer (Pen)
02h	00h	Mouse controller
80h	00h	Other input controller

Table 3-18. Base Class Code 0Ah: Docking Stations

Sub-Class	Prog I/F	Description
00h	00h	Generic docking station
80h	00h	Other type of docking station

Table 3-19. Base Class Code 0Bh: Processors

Sub-Class	Prog I/F	Description
00h	00h	386
01h	00h	486
02h	00h	Pentium™
10h	00h	Alpha™
40h	00h	Co-processor

Table 3-20. Base Class Code 0Ch: Serial Bus Controllers

Sub-Class	Prog I/F	Description
00	00h	Firewire (IEEE 1394)
01h	00h	ACCESS.bus
02h	00h	SSA

3.7 CACHE LINE SIZE REGISTER (CALN)

Register Name: Cache Line Size
Address Offset: 0Ch
Power-up value: 00h, hardwired
Boot-load: not used
Attribute: Read Only
Size: 8 bits

This register is hardwired to 0. The cache line configuration register is used by the system to define the cache line size in doubleword (64-bit) increments. This controller does not use the "Memory Write and Invalidate" PCI bus cycle commands when operating in the bus master mode, and therefore does not internally require this register. When operating in the target mode, this controller does not have the connections necessary to "snoop" the PCI bus and accordingly cannot employ this register in the detection of burst transfers that cross a line boundary.

Figure 3-7. Cache Line Size Register
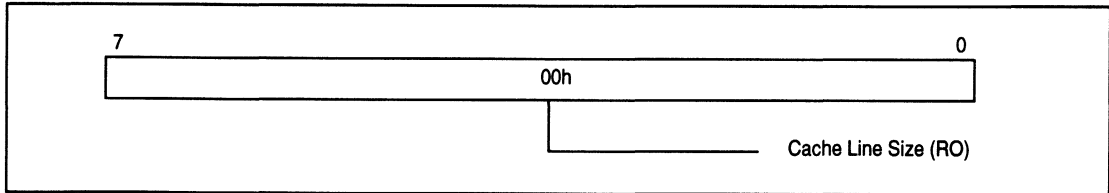
3.8   LATENCY TIMER REGISTER (LAT)

Register Name: Latency Timer

Address Offset: 0Dh

Power-up value: 00h

Boot-load: External nvRAM offset
                   04Dh

Attribute: Read/Write, bits 7:3;
                   Read Only bits 2:0

Size: 8 bits

The latency timer register has meaning only when this controller is used as a bus master and pertains to the number of PCI bus clocks that this master will be guaranteed. The nonzero value for this register is internally decremented after this device has been granted the bus and has begun to assert FRAME#. Prior to this latency timer count reaching zero, this device can ignore the removal of the bus grant and may continue the use of the bus for data transfers.

Figure 3-8. Latency Timer Register

7	6	5	4	3	2	1	0	Bit
X	X	X	X	X	0	0	0	Value

Latency Timer value (R/W)
# of clocks x 8

3.9 HEADER TYPE REGISTER (HDR)

Register Name:	Header Type
Address Offset	0Eh
Power-up value:	00h
Boot-load:	External nvRAM offset 04Eh
Attribute:	Read Only
Size:	8 bits

This register consists of two fields: Bits 6:0 define the format for bytes 10h through 3Fh of the device configuration header, and bit 7 establishes whether this device represents a single function (bit 7 = 0) or a multifunction (bit 7 = 1) PCI bus agent. The S5933 is a single function PCI device.

3

Figure 3-9. Header Type Register
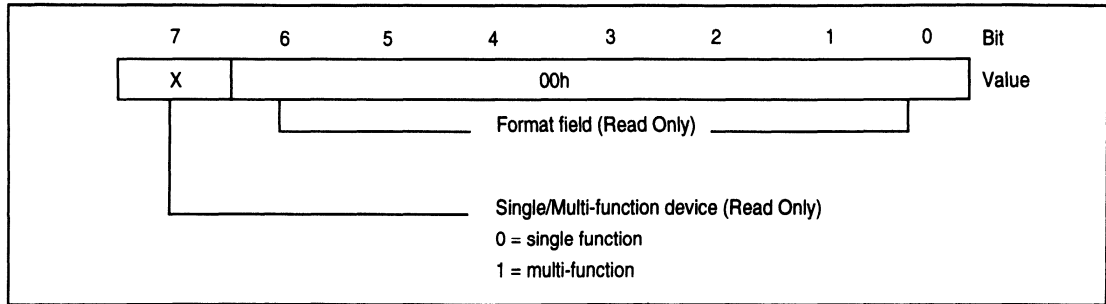
3.10 BUILT-IN SELF-TEST REGISTER (BIST)

Register Name: Built-in Self-Test
Address Offset   0Fh
Power-up value: 00h
Boot-load: External nvRAM/EPROM offset 04Fh
Attribute: D7, D5-0 Read Only, D6 as PCI bus write only
Size: 8 bits

The Built-In Self-Test (BIST) register permits the implementation of custom, user-specific diagnostics. This register has four fields as depicted in Figure 3-10. Bit 7, when set signifies that this device supports a built-in self test. When bit 7 is set, writing a 1 to bit 6 will commence the self test. In actuality, writing a 1 to bit 6 produces an interrupt to the add-on interface. Bit 6 will remain set until cleared by a write operation to this register from the add-on bus interface. When bit 6 is reset it is interpreted as completion of the self-test and an error is indicated by a non-zero value for the completion code (bits 3:0).

Figure 3-10. Built-In Self Test Register
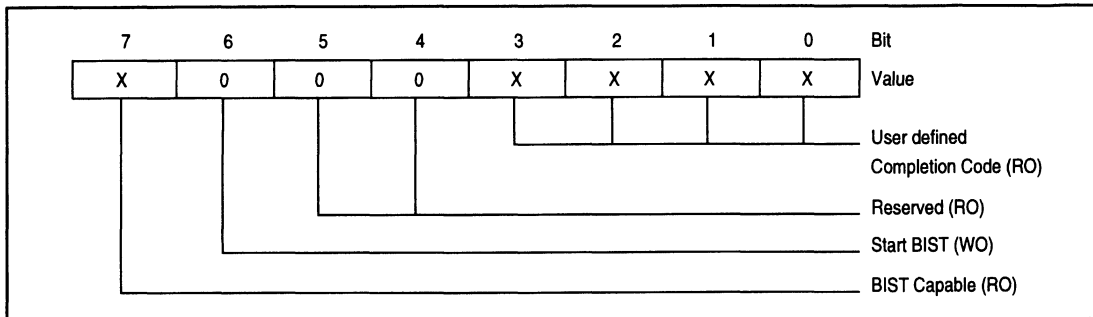
Table 3-21. Built-In Self-Test Register

Bit	Description
7	BIST Capable. This bit indicates that the add-on device supports a built-in self-test when a one is returned. A zero should be returned if this self test feature is not desired. This field is read only from the PCI interface.
6	Start BIST. Writing a 1 to this bit indicates that the self-test should commence. This bit can only be written when bit 7 is a 1. When bit 6 becomes set, an interrupt is issued to the add-on interface. Other than through the reset pin, Bit 6 can only be cleared by a write to this element from the add-on bus interface as outlined in Section 5.5. The PCI bus specification requires that this bit be cleared within 2 seconds after being set, or the device will be failed.
5:4	Reserved. These bits are reserved. This field will always return zeros.
3:0	Completion Code. This field provides a method for detailing a device-specific error. It is considered valid when the Start BIST field (bit 6) changes from 1 to 0. An all-zero value for the completion code indicates successful completion. Refer to Section 5.10.

3

3.11 BASE ADDRESS REGISTERS (BADR)

Register Name: Base Address
Address Offset: 10h, 14h, 18h, 1Ch, 20h, 24h
Power-up value: FFFFFFC1h for offset 10h;
                    00000000h for all others
Boot-load: External nvRAM offset
                    050h, 54h, 58h, 5Ch, 60h
                    (BADR0-4)
Attribute: high bits Read/Write; low bits
                    Read Only
Size: 32 bits

The base address registers provide a mechanism for assigning memory or I/O space for the add-on function. The actual location(s) the add-on function is to respond to is determined by first interrogating these registers to ascertain the size or space desired, and then writing the high-order field of each register to place it physically in the system's address space. Bit zero of each field is used to select whether the space required is to be decoded as memory (bit 0 = 0) or I/O (bit 0 = 1). Since this PCI controller has 16 DWORDs of internal operating registers (Chapter 4), the Base Address Register at offset 10h is assigned to them. The remaining five base address registers can only be used by boot-loading them from the external nvRAM Interface. BADR5 register is not implemented and will return all 0's.

Determining Base Address Size

The address space defined by a given base address register is determined by writing all 1s to a given base address register from the PCI bus and then reading that register back. The number of 0s returned starting from D4 for memory space and D2 for I/O space toward the high-order bits reveals the amount of address space desired. Tables 3-23 and 3-24 list the possible returned values and their corresponding size for both memory and I/O, respectively. Included in the table are the nvRAM/EPROM boot values which correspond to a given assigned size. A register returning all zeros is disabled.

Assigning the Base Address

After a base address has been sized as described in the preceding paragraph, the region associated with that base address register (the high order one bits) can physically locate it in memory (or I/O) space. For example, the first base address register returns FFFFFFC1h indicating an I/O space (D0=1) and is then written with the value 00000300h. This means that the controller's internal registers can be selected for I/O addresses between 00000300h through 0000033Fh, in this example. The base address value must be on a natural binary boundary for the required size (example 300h, 340h, 380h etc.; 338h would not be allowable).

Figure 3-11a. Base Address Register — Memory
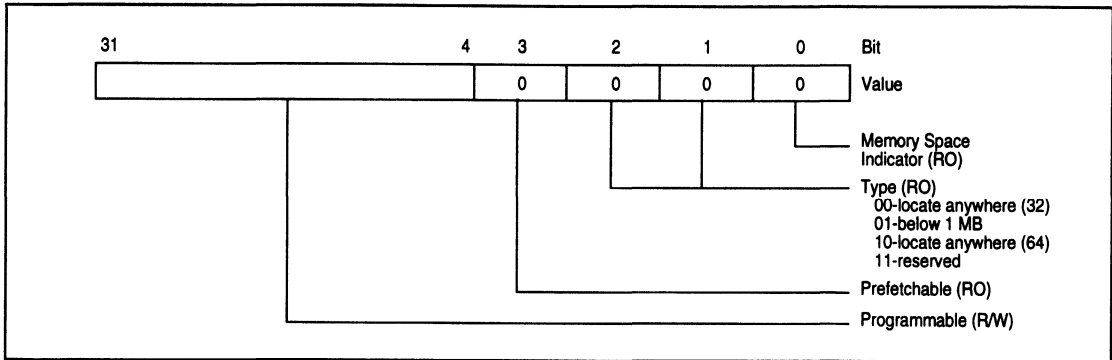
Figure 3-11b. Base Address Register — I/O
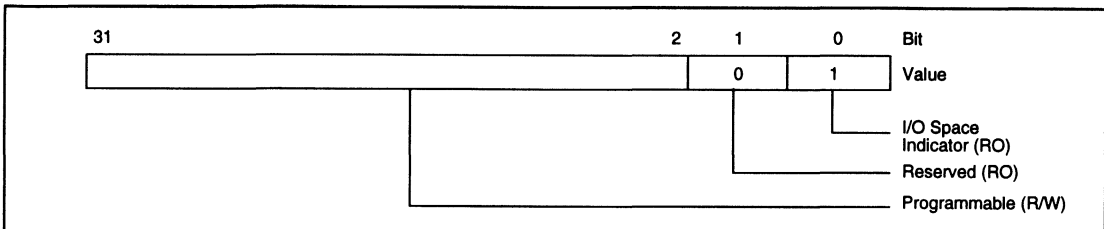
Table 3-22a. Base Address Register — Memory (Bit 0 = 0)

Bit	Description
31:4	Base Address Location. These bits are used to position the decoded region in memory space. Only bits which return a 1 after being written as 1 are usable for this purpose. Except for Base Address Register 0, these bits are individually enabled by the contents sourced from the external boot memory.
3	Prefetchable. When set as a 1, this bit signifies that this region of memory can be cached. Cachable regions can only be located within the region altered through PCI bus memory writes. This bit, when set, also implies that all read operations will return the data associated for all bytes regardless of the Byte Enables. Memory space which cannot support this behavior should leave this bit in the zero state. For Base Addresses 1 through 4, this bit is set by the Reset pin and later initialized by the external boot memory (if present). Base Address Register 0 always has this bit set to 0. This bit is read only from the PCI interface.
2:1	Memory Type. These two bits identify whether the memory space is 32 or 64 bits wide and if the space location is restricted to be within the first megabyte of memory space. The table below describes the encoding:
	Bits 2 1 **Description**
	0 0 Region is 32 bits wide and can be located anywhere in 32 bit memory space.
	0 1 Region is 32 bits wide and must be mapped below the first MByte of memory space.
	1 0 Region is 64 bits wide and can be mapped anywhere within 64 bit memory space. (Not supported by this controller.)
	1 1 Reserved. (Not supported by this controller.)
1	The 64-bit memory space is not supported by this controller, so bit 2 should not be set. The only meaningful option is whether it is desired to position memory space anywhere within 32-bit memory space or restrain it to the first megabyte. For Base Addresses 1 through 5, this bit is set by the reset pin and later initialized by the external boot memory (if present).
0	Space Indicator = 0. When set to 0, this bit identifies a base address region as a memory space and the remaining bits in the base address register are defined as shown in Table 3-22a.

Table 3-22b. Base Address Register — I/O (Bit 0 = 1)

Bit	Description
31:2	Base Address Location. These bits are used to position the decoded region in I/O space. Only bits which return a "1" after being written as "1" are usable for this purpose. Except for Base Address 0, these bits are individually enabled by the contents sourced from the external boot memory (EPROM or nvRAM).
1	Reserved. This bit should be zero. (Note: disabled Base Address Registers will return all zeros for the entire register location, bits 31 through 0).
0	Space Indicator = 1. When one this bit identifies a base address region as an I/O space and the remaining bits in the base address register have the definition as shown in Table 3-11b.

Table 3-23. Read Response (Memory Assigned) to an All-Ones Write Operation to a Base Address Register

Response	Size in bytes	[EPROM boot value] (note 1)
00000000h	none - disabled	00000000h or BIOS missing (note 2,3)
FFFFFFF0h	16 bytes (4 DWORDs)	FFFFFFF0h
FFFFFFE0h	32 bytes (8 DWORDs)	FFFFFFE0h
FFFFFFC0h	64 bytes (16 DWORDs)	FFFFFFC0h
FFFFFF80h	128 bytes (32 DWORDs)	FFFFFF80h
FFFFFF00h	256 bytes (64 DWORDs)	FFFFFF00h
FFFFFE00h	512 bytes (128 DWORDs)	FFFFFE00h
FFFFFC00h	1K bytes (256 DWORDs)	FFFFFC00h
FFFFF800h	2K bytes (512 DWORDs)	FFFFF800h
FFFFF000h	4K bytes (1K DWORDs)	FFFFF000h
FFFFE000h	8K bytes (2K DWORDs)	FFFFE000h
FFFFC000h	16K bytes (4K DWORDs)	FFFFC000h
FFFF8000h	32K bytes (8K DWORDs)	FFFF8000h
FFFF0000h	64K bytes (16K DWORDs)	FFFF0000h
FFFE0000h	128K bytes (32K DWORDs)	FFFE0000h
FFFC0000h	256K bytes (64K DWORDs)	FFFC0000h
FFF80000h	512K bytes (128K DWORDs)	FFF80000h
FFF00000h	1M bytes (256K DWORDs)	FFF00000h
FFE00000h	2M bytes (512K DWORDs)	FFE00000h
FFC00000h	4M bytes (1M DWORDs)	FFC00000h
FF800000h	8M bytes (2M DWORDs)	FF800000h
FF000000h	16M bytes (4M DWORDs)	FF000000h
FE000000h	32M bytes (8M DWORDs)	FE000000h
FC000000h	64M bytes (16M DWORDs)	FC000000h
F8000000h	128M bytes (32M DWORDs)	F8000000h
F0000000h	256M bytes (64M DWORDs)	F0000000h
E0000000h	512M bytes (128M DWORDs)	E0000000h

Note 1. The two most significant bits define bus width for BADR1:4 in Pass-Thru operation (see Section 11.3.1).

Note 2. Bits D3, D2 and D1 may be set to indicate other attributes for the memory space. See text for details.

Note 3. BADR5 register is not implemented and will return all 0's.

Table 3-24. Read Response (I/O Assigned) to an All-Ones write Operation to a Base Address Register

Response	Size in bytes	[EPROM boot value]
00000000h	none - disabled	00000000h or BIOS missing (note 3)
FFFFFFFDh	4 bytes (1 DWORDs)	FFFFFFFDh
FFFFFFF9h	8 bytes (2 DWORDs)	FFFFFFF9h
FFFFFFF1h	16 bytes (4 DWORDs)	FFFFFFF1h
FFFFFFE1h	32 bytes (8 DWORDs)	FFFFFFE1h
FFFFFFC1h	64 bytes (16 DWORDs)	FFFFFFC1h (note 4)
FFFFFF81h	128 bytes (32 DWORDs)	FFFFFF81h
FFFFFF01h	256 bytes (64 DWORDs)	FFFFFF01h
FFFFFE01h	512 bytes (128 DWORDs)	FFFFFE01h
FFFFFC01h	1K bytes (256 DWORDs)	FFFFFC01h
FFFFF801h	2K bytes (512 DWORDs)	FFFFF801h
FFFFF001h	4K bytes (1K DWORDs)	FFFFF001h
FFFFE001h	8K bytes (2K DWORDs)	FFFFE001h
FFFFC001h	16K bytes (4K DWORDs)	FFFFC001h
FFFF8001h	32K bytes (8K DWORDs)	FFFF8001h
FFFF0001h	64K bytes (16K DWORDs)	FFFF0001h
FFFE0001h	128K bytes (32K DWORDs)	FFFE0001h
FFFC0001h	256K bytes (64K DWORDs)	FFFC0001h
FFF80001h	512K bytes (128K DWORDs)	FFF80001h
FFF00001h	1M bytes (256K DWORDs)	FFF00001h
FFE00001h	2M bytes (512K DWORDs)	FFE00001h
FFC00001h	4M bytes (1M DWORDs)	FFC00001h
FF800001h	8M bytes (2M DWORDs)	FF800001h
FF000001h	16M bytes (4M DWORDs)	FF000001h
FE000001h	32M bytes (8M DWORDs)	FE000001h
FC000001h	64M bytes (16M DWORDs)	FC000001h
F8000001h	128M bytes (32M DWORDs)	F8000001h
F0000001h	256M bytes (64M DWORDs)	F0000001h
E0000001h	512M bytes (128M DWORDs)	E0000001h

Note 4. Base Address Register 0 (at offset) 10h powers up as FFFFFFC1h. This default assignment allows usage without an external boot memory. Should an EPROM or nvRAM be used, the base address can be boot loaded to become a memory space (FFFFFFC0h or FFFFFFC2h).

3.12 EXPANSION ROM BASE ADDRESS REGISTER (XROM)

Register Name: Expansion ROM Base Address
Address Offset: 30h
Power-up value: 00000000h
Boot-load: External nvRAM offset
                   70h
Attribute: bits 31:11, bit 0 Read/Write; bits
                   10:1 Read Only
Size: 32 bits

The expansion base address ROM register provides a mechanism for assigning a space within physical memory for an expansion ROM. Access from the PCI bus to the memory space defined by this register will cause one or more accesses to the S5933 controllers' external BIOS ROM (or nvRAM) interface. Since PCI bus accesses to the ROM may be 32 bits wide, repeated operations to the ROM are generated by the S5933 and the wider data is assembled internal to the S5933 controller and then transferred to the PCI bus by the S5933.

Figure 3-12. Expansion ROM Base Address Register
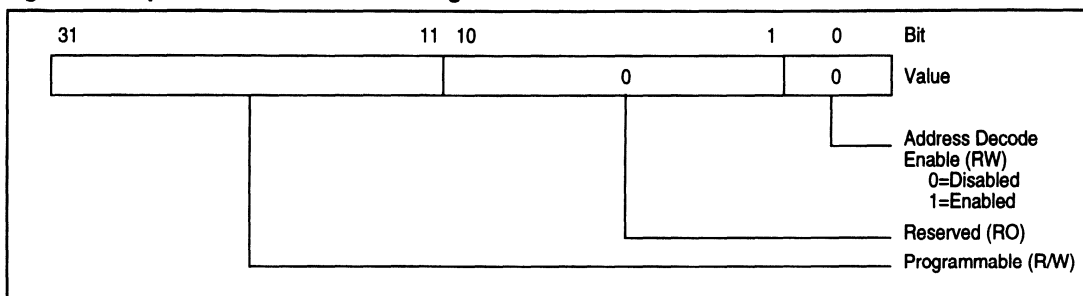
Table 3-25. Expansion ROM Base Address Register

Bit	Description
31:11	Expansion ROM Base Address Location. These bits are used to position the decoded region in memory space. Only bits which return a 1 after being written as 1 are usable for this purpose. These bits are individually enabled by the contents sourced from the external boot memory (EPROM or nvRAM). The desired size for the ROM memory is determined by writing all ones to this register and then reading back the contents. The number of bits returned as zeros, in order from least significant to most significant bit, indicates the size of the expansion ROM. This controller limits the expansion ROM area to 64K bytes. The allowable returned values after all ones are written to this register are shown in Table 3-26.
10:1	Reserved. All zeros.
0	Address Decode Enable. The Expansion ROM address decoder is enabled or disabled with this bit. When this bit is set, the decoder is enabled; when this bit is zero, the decoder is disabled. It is required that the command register (Section 3.3) also have the memory decode enabled for this bit to have an effect.

Table 3-26. Read Response to Expansion ROM Base Address Register (after all-ones written)

Response	Size in bytes	[EPROM boot value]
00000000h	none - disabled	00000000h or BIOS missing
FFFFF801h	2K bytes (512 DWORDs)	FFFFF801h
FFFFF001h	4K bytes (1K DWORDs)	FFFFF001h
FFFFE001h	8K bytes (2K DWORDs)	FFFFE001h
FFFFC001h	16K bytes (4K DWORDs)	FFFFC001h
FFFF8001h	32K bytes (8K DWORDs)	FFFF8001h
FFFF0001h	64K bytes (16K DWORDs)	FFFF0001h

3.13 INTERRUPT LINE REGISTER (INTLN)

Register Name:	Interrupt Line
Address Offset:	3Ch
Power-up value:	FFh
Boot-load:	External nvRAM offset 7Ch
Attribute:	Read/Write
Size:	8 bit

This register indicates the interrupt routing for the S5933 controller. The ultimate value for this register is system-architecture specific. For x86 based PCs, the values in this register correspond with the established interrupt numbers associated with the dual 8259 controllers used in those machines. In x86-based PC systems, the values of 0 to 15 correspond with the IRQ numbers 0 through 15, and the values from 16 to 254 are reserved. The value of 255 (the controller's default power-up value) signifies either "unknown" or "no connection" for the system interrupt. This register is boot-loaded from the external boot memory, if present, and may be written by the PCI interface.

Figure 3-13. Interrupt Line Register

7	6	5	4	3	2	1	0	Bit
			FFh					Value
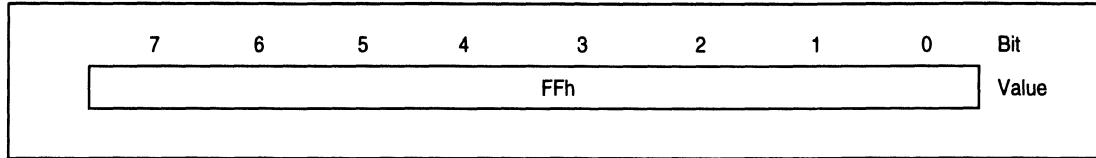
3.14 INTERRUPT PIN REGISTER (INTPIN)

Register Name: Interrupt Pin

Address Offset: 3Dh

Power-up value: 01h

Boot-load: External nvRAM offset
                   7Dh

Attribute: Read Only

Size: 8 bits

This register identifies which PCI interrupt, if any, is connected to the controller's PCI interrupt pins. The allowable values are 0 (no interrupts), 1 (INTA#), 2 (INTB#), 3 (INTC#), and 4 (INTD#). The default power-up value assumes INTA#.

Figure 3-14. Interrupt Pin Register

3
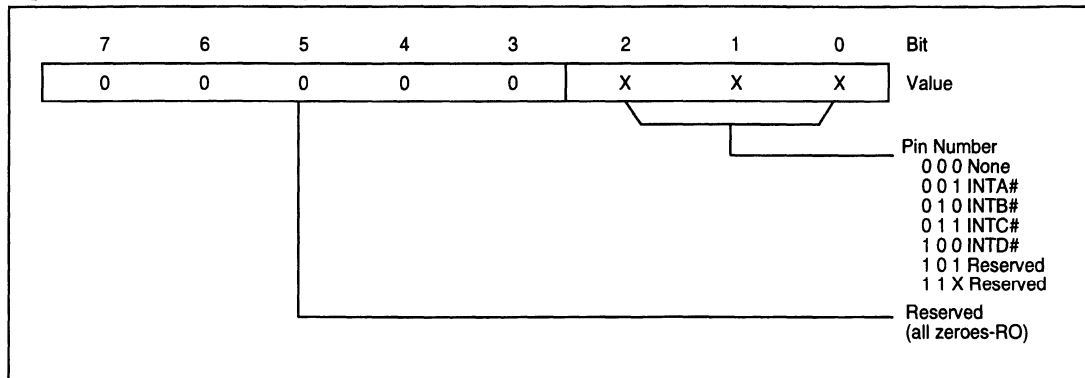
3.15 MINIMUM GRANT REGISTER (MINGNT)

Register Name:	Minimum Grant
Address Offset:	3Eh
Power-up value:	00h
Boot-load:	External nvRAM offset 7Eh
Attribute:	Read Only
Size:	8 bits

This register may be optionally used by bus masters to specify how long a burst period the device needs. A value of zero indicates that the bus master has no stringent requirement. The units defined by the least significant bit are in 250-ns increments. This register is treated as "information only" and has no further implementation within this device.

Values other than zero are possible when an external boot memory is used.

Figure 3-15. Minimum Grant Register

7	6	5	4	3	2	1	0	bit
0	0	0	0	0	0	0	0	value

Value x 250ns (RO)
00-no requirement
01-FFh

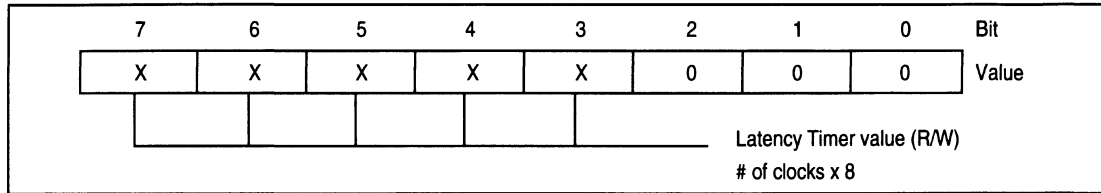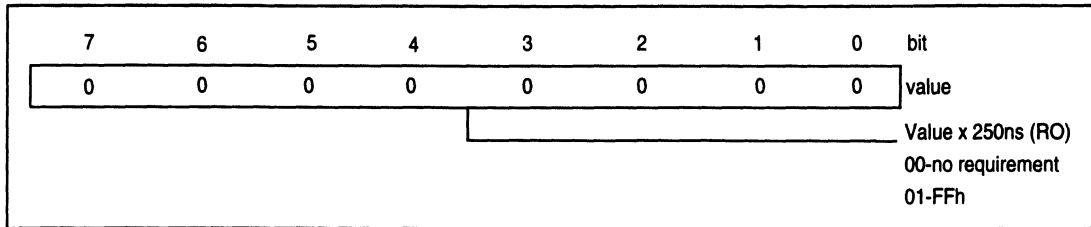## 3.16 MAXIMUM LATENCY REGISTER (MAXLAT)

Register Name: Maximum Latency

Address Offset: 3Fh

Power-up value: 00h

Boot-load: External nvRAM offset 7Fh

Attribute: Read Only

Size: 8 bits

This register may be optionally used by bus masters to specify how often this device needs PCI bus access. A value of zero indicates that the bus master has no stringent requirement. The units defined by the least significant bit are in 250-ns increments. This register is treated as "information only" and has no further implementation within this device.

Values other than zero are possible when an external boot memory is used.

Figure 3-16. Maximum Latency Register

7	6	5	4	3	2	1	0	bit
0	0	0	0	0	0	0	0	value

Value x 250ns (RO)
00-no requirement
01-FFh
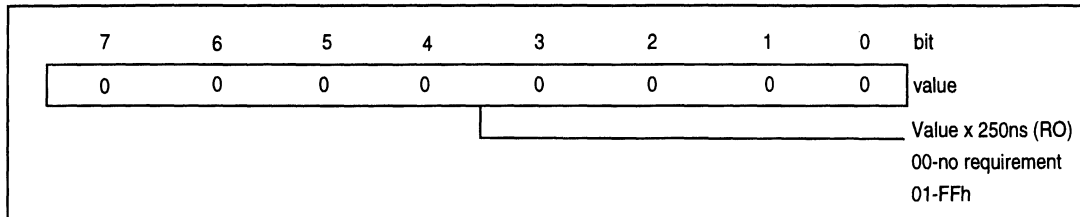
4.0   PCI BUS OPERATION REGISTERS

The PCI bus operation registers are mapped as 16 consecutive DWORD registers located at the address space (I/O or memory) specified by the Base Address Register 0 (Section 3.11). These locations are the primary method of communication between the PCI and add-on buses. Data, software-defined commands and command parameters can be either exchanged through the mailboxes, transferred through the FIFO in blocks under program control, or transferred using the FIFOs under Bus Master control. Table 4-1 lists the PCI Bus Operation Registers.

Table 4-1. Operation Registers — PCI Bus

Address Offset	Abbreviation	Register Name
00h	OMB1	Outgoing Mailbox Register 1
04h	OMB2	Outgoing Mailbox Register 2
08h	OMB3	Outgoing Mailbox Register 3
0Ch	OMB4	Outgoing Mailbox Register 4
10h	IMB1	Incoming Mailbox Register 1
14h	IMB2	Incoming Mailbox Register 2
18h	IMB3	Incoming Mailbox Register 3
1Ch	IMB4	Incoming Mailbox Register 4
20h	FIFO	FIFO Register port (bidirectional)
24h	MWAR	Master Write Address Register
28h	MWTC	Master Write Transfer Count Register
2Ch	MRAR	Master Read Address Register
30h	MRTC	Master Read Transfer Count Register
34h	MBEF	Mailbox Empty/Full Status
38h	INTCSR	Interrupt Control/Status Register
3Ch	MCSR	Bus Master Control/Status Register

4.1 OUTGOING MAILBOX REGISTERS (OMB)

Register Names: Outgoing Mailboxes 1-4

PCI Address Offset: 00h, 04h, 08h, 0Ch

Power-up value: XXXXXXXXh

Attribute: Read/Write

Size: 32 bits

These four DWORD registers provide a method for sending command or parameter data to the add-on system. PCI bus operations to these registers may be in any width (byte, word, or DWORD). Writing to these registers can be a source for add-on bus interrupts (if desired) by enabling their interrupt generation through the use of the add-on's interrupt control/status register (Section 5.9).

4.2 INCOMING MAILBOX REGISTERS (IMB)

Register Names: Incoming Mailboxes 1-4

PCI Address Offset: 10h, 14h, 18h, 1Ch

Power-up value: XXXXXXXXh

Attribute: Read Only

Size: 32 bits

These four DWORD registers provide a method for receiving user defined data from the add-on system. PCI bus read operations to these registers may be in any width (byte, word, or DWORD). Only read operations are supported. Reading from these registers can optionally cause an add-on bus interrupt (if desired) by enabling their interrupt generation through the use of the add-on's interrupt control/status register (described in Section 5.9).

Mailbox 4, byte 3 only exists as device pins on the S5933 devices when used with a serial nonvolatile memory.

4.3 FIFO REGISTER PORT (FIFO)

Register Name: FIFO Port

PCI Address Offset: 20h

Power-up value: XXXXXXXXh

Attribute: Read/Write

Size: 32 bits

This location provides access to the bidirectional FIFO. Separate registers are used when reading from or writing to the FIFO. Accordingly, it is not possible to read what was written to this location. The FIFO registers are implicitly involved in all bus master operations and, as such, should not be accessed during active bus master transfers. When operating upon the FIFOs with software program transfers involving word or byte operations, the *endian* sequence of the FIFO should be established as outlined in Section 10.1.1.2 in order to preserve the internal FIFO data ordering and flag management. The FIFO's fullness may be observed by reading the master control-status register, MCSR, described in Section 4.10.

4.4 PCI CONTROLLED BUS MASTER WRITE ADDRESS REGISTER (MWAR)

Register Name: Master Write Address
PCI Address Offset: 24h
Power-up value: 00000000h
Attribute: Read/Write
Size: 32 bits

This register is used to establish the PCI address for data moving from the add-on bus to the PCI bus during PCI bus memory write operations. It consists of a 30-bit counter with the low-order two bits hardwired as zeros. Transfers may be any non-zero byte length as defined by the transfer count register, MWTC (Section 4.5), and must begin on a DWORD boundary. This DWORD boundary starting constraint is placed upon this controller's PCI bus master transfers so that byte lane alignment can be maintained between the S5933 controller's internal FIFO data path, the add-on interface, and the PCI bus.

Note: Applications which require a non-DWORD starting boundary will need to move the first few bytes under software program control (and without using the FIFO) to establish a DWORD boundary.

After the DWORD boundary is established the S5933 can begin the task of PCI bus master data transfers.

The Master Write Address Register is continually updated during the transfer process and will always be pointing to the next unwritten location. Reading of this register during a transfer process (done when the S5933 controller is functioning as a target, i.e. not a bus master) is permitted and may be used to monitor the progress of the transfer. During the address phase for bus master write transfers, the two least significant bits presented on the PCI bus pins AD[31:0] will always be zero. This identifies to the target memory that the burst address sequence will be in a linear order rather than in an Intel 486 or Pentium™ cache line fill sequence. Also, the PCI bus address bit A1 will always be zero when this controller is the bus master. This signifies to the target that the S5933 controller is burst capable and that the target should not arbitrarily disconnect after the first data phase of this operation.

Under certain circumstances, MWAR can be accessed from the add-on bus instead of the PCI bus. See Section 10.1.4.1.

Figure 4-1. PCI Controlled Bus Master Write Address Register
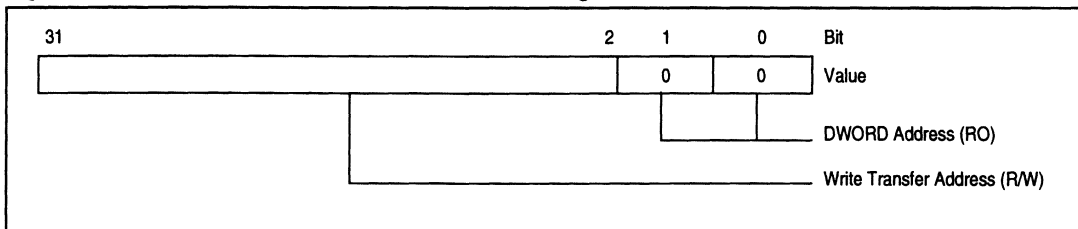
4.5 PCI CONTROLLED BUS MASTER WRITE TRANSFER COUNT REGISTER (MWTC)

Register Name: Master Write Transfer Count

PCI Address Offset: 28h

Power-up value: 00000000h

Attribute: Read/Write

Size: 32 bits

The master write transfer count register is used to convey to the S5933 controller the actual number of bytes that are to be transferred. The value in this register is decremented with each bus master PCI write operation until the transfer count reaches zero.

Upon reaching zero, the transfer operation ceases and an interrupt may be optionally generated to either the PCI or add-on bus interface. Transfers which are not whole multiples of DWORDs in size result in a partial word ending cycle. This partial word ending cycle is possible since all bus master transfers for this controller are required to begin on a DWORD boundary.

Under certain circumstances, MWTC can be accessed from the add-on bus instead of the PCI bus. See Section 10.1.4.1.

Figure 4-2. PCI Controlled Bus Master Write Transfer Count Register
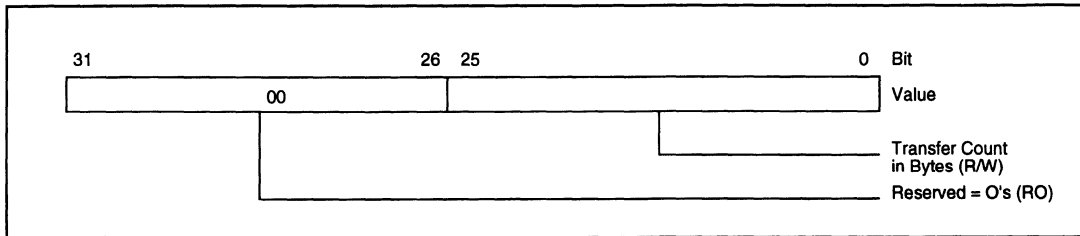
4.6 PCI CONTROLLED BUS MASTER READ ADDRESS REGISTER (MRAR)

Register Name: Master Read Address

PCI Address Offset: 2Ch

Power-up value: 00000000h

Attribute: Read/Write

Size: 32 bits

This register is used to establish the PCI address for data moving to the add-on bus from the PCI bus during PCI bus memory read operations. It consists of a 30-bit counter with the low-order two bits hardwired as zeros. Transfers may be any non-zero byte length as defined by the transfer count register, MRTC (Section 4.7) and must begin on a DWORD boundary. This DWORD boundary starting constraint is placed upon this controller's PCI bus master transfers so that byte lane alignment can be maintained between the S5933 controller's internal FIFO data path, the add-on interface and the PCI bus.

Note: Applications which require a non-DWORD starting boundary will need to move the first few bytes under software program control (and without using the FIFO) to establish a DWORD boundary.

After the DWORD boundary is established the S5933 can begin the task of PCI bus master data transfers.

The Master Read Address Register is continually updated during the transfer process and will always be pointing to the next unread location. Reading of this register during a transfer process (done when the S5933 controller is functioning as a target—i.e., not a bus master) is permitted and may be used to monitor the progress of the transfer. During the address phase for bus master read transfers, the two least significant bits presented on the PCI bus AD[31:0] will always be zero. This identifies to the target memory that the burst address sequence will be in a linear order rather than in an Intel 486 or Pentium™ cache line fill sequence. Also, the PCI bus address bit A1 will always be zero when this controller is the bus master. This signifies to the target that the controller is burst capable and that the target should not arbitrarily disconnect after the first data phase of this operation.

Under certain circumstances, MRAR can be accesed from the add-on bus instead of the PCI bus. See Section 10.1.4.1.

Figure 4-3. PCI Controlled Bus Master Read Address Register
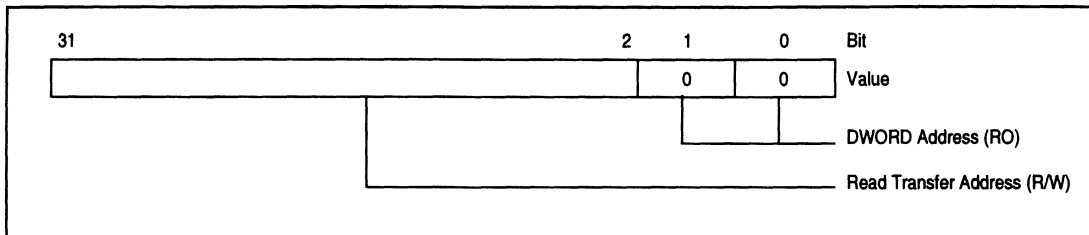
4.7   PCI CONTROLLED BUS MASTER READ TRANSFER COUNT REGISTER (MRTC)

Register Name: Master Read Transfer Count

PCI Address Offset: 30h

Power-up value: 00000000h

Attribute: Read/Write

Size: 32 bits

The master read transfer count register is used to convey to the PCI controller the actual number of bytes that are to be transferred. The value in this register is decremented with each bus master PCI read operation until the transfer count reaches zero. Upon reaching zero, the transfer operation ceases and an interrupt may be optionally generated to either the PCI or add-on bus interface. Transfers which are not whole multiples of DWORDs in size result in a partial word ending cycle. This partial word ending cycle is possible since all bus master transfers for this controller are required to begin on a DWORD boundary.

Under certain circumstances, MRTC can be accessed from the add-on bus instead of the PCI bus. See Section 10.1.4.1.

Figure 4-4. PCI Controlled Bus Master Read Transfer Count Register
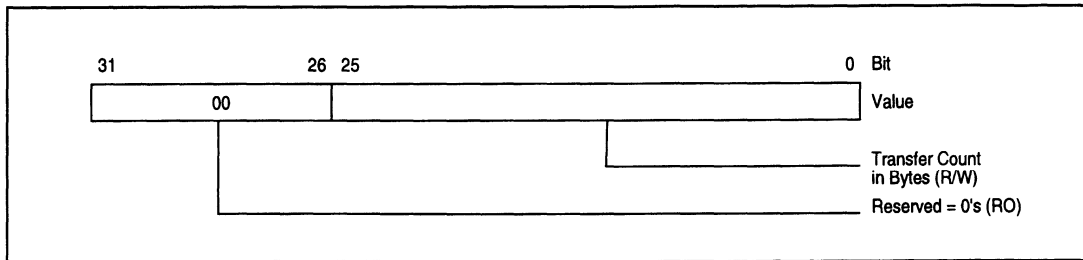
4.8 MAILBOX EMPTY FULL/STATUS REGISTER (MBEF)

Register Name: Mailbox Empty/Full Status

PCI Address Offset: 34h

Power-up value: 00000000h

Attribute: Read Only

Size: 32 bits

This register provides empty/full visibility of each byte within the mailboxes. The empty/full status for the Outgoing mailboxes is displayed on the low-order 16 bits and the empty/full status for the Incoming mailboxes is presented on the high-order 16 bits. A value of 1 signifies that a given mailbox has been written by one bus interface but has not yet been read by the corresponding destination interface. A PCI bus incoming mailbox is defined as one in which data travels from the add-on bus into the PCI bus, and an outgoing mailbox is defined as one where data travels out from the PCI bus to the add-on interface.

4

Figure 4-5. Mailbox Empty/Full Status Register
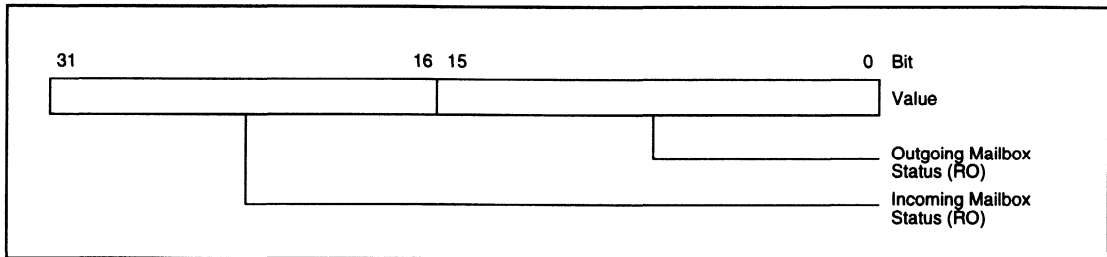
Table 4-2. Mailbox Empty/Full Status Register

Bit	Description
31:16	Incoming Mailbox Status. This field indicates which incoming mailbox registers have been written by the add-on interface but have not yet been read by the PCI bus. Each bit location corresponds to a specific byte within one of the four incoming mailboxes. A value of one for each bit signifies that the specified mailbox byte is full, and a value of zero signifies empty. The mapping of these status bits to bytes within each mailbox is as follows: Bit 31 = Incoming mailbox 4 byte 3 Bit 30 = Incoming mailbox 4 byte 2 Bit 29 = Incoming mailbox 4 byte 1 Bit 28 = Incoming mailbox 4 byte 0 Bit 27 = Incoming mailbox 3 byte 3 Bit 26 = Incoming mailbox 3 byte 2 Bit 25 = Incoming mailbox 3 byte 1 Bit 24 = Incoming mailbox 3 byte 0 Bit 23 = Incoming mailbox 2 byte 3 Bit 22 = Incoming mailbox 2 byte 2 Bit 21 = Incoming mailbox 2 byte 1 Bit 20 = Incoming mailbox 2 byte 0 Bit 19 = Incoming mailbox 1 byte 3 Bit 18 = Incoming mailbox 1 byte 2 Bit 17 = Incoming mailbox 1 byte 1 Bit 16 = Incoming mailbox 1 byte 0
15:00	Outgoing Mailbox Status. This field indicates which outgoingmailbox registers have been written by the PCI bus interface but have not yet been read by the add-on bus. Each bit location corresponds to a specific byte within one of the four outgoing mailboxes. A value of one for each bit signifies that the specified mailbox byte is full, and a value of zero signifies empty. The mapping of these status bits to bytes within each mailbox is as follows: Bit 15 = Outgoing mailbox 4 byte 3 Bit 14 = Outgoing mailbox 4 byte 2 Bit 13 = Outgoing mailbox 4 byte 1 Bit 12 = Outgoing mailbox 4 byte 0 Bit 11 = Outgoing mailbox 3 byte 3 Bit 10 = Outgoing mailbox 3 byte 2 Bit 09 = Outgoing mailbox 3 byte 1 Bit 08 = Outgoing mailbox 3 byte 0 Bit 07 = Outgoing mailbox 2 byte 3 Bit 06 = Outgoing mailbox 2 byte 2 Bit 05 = Outgoing mailbox 2 byte 1 Bit 04 = Outgoing Mailbox 2 byte 0 Bit 03 = Outgoing Mailbox 1 byte 3 Bit 02 = Outgoing Mailbox 1 byte 2 Bit 01 = Outgoing Mailbox 1 byte 1 Bit 00 = Outgoing Mailbox 1 byte 0

4.9 INTERRUPT CONTROL/STATUS REGISTER (INTCSR)

Register Name: Interrupt Control and Status

PCI Address Offset: 38h

Power-up value: 00000000h

Attribute: Read/Write (R/W),
                     Read/Write_One_Clear (R/WC)

Size: 32 bits

This register provides the method for chosing which conditions are to produce an interrupt on the PCI bus interface, a method for viewing the cause of the interrupt, and a method for acknowledging (removing) the interrupt's assertion.

Interrupt sources:

- Write Transfer Terminal Count = zero
- Read Transfer Terminal Count = zero
- One of the Outgoing mailboxes (1,2,3 or 4) becomes empty
- One of the Incoming mailboxes (1,2,3 or 4) becomes full.
- Target Abort
- Master Abort

4

Figure 4-6. Interrupt Control/Status Register

Figure 4-6. Interrupt Control/Status Register
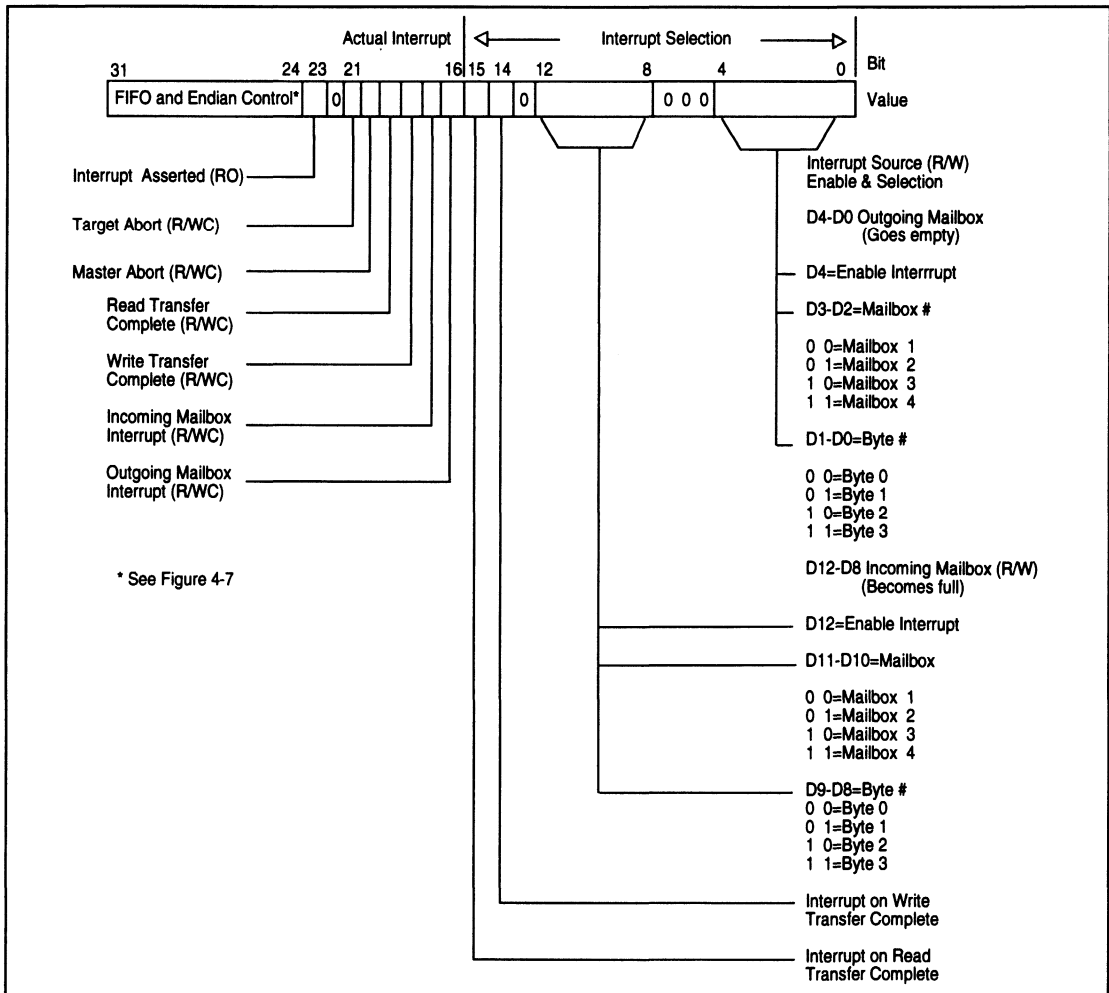
Figure 4-7. FIFO Management and Endian Control Byte

```
                    31 30 29  28 27  26 25  24
                   ┌──┬──┬──┬──┬──┬──┬──┬──┐
                   │  │  │  │  │  │  │  │  │
                   └──┴──┴──┴──┴──┴──┴──┴──┘

                                          0  0   NO CONVERSION (DEFAULT)
                                          0  1   16 BIT ENDIAN CONV.
  OUTBOUND FIFO                           1  0   32 BIT ENDIAN CONV.
  PCI→ADD-ON DWORD                        1  1   64 BIT ENDIAN CONV
  TOGGLE
  0 = BYTES 0-3 (DEFAULT)
  1 = BYTE 4-7 (NOTE1)                    FIFO ADVANCE CONTROL
                                          PCI INTERFACE
                                          0  0   BYTE 0    (DEFAULT)
                                          0  1   BYTE 1
                                          1  0   BYTE 2
                                          1  1   BYTE 3
  INBOUND FIFO
  ADD-ON→ PCI DWORD
  TOGGLE
  0 = BYTES 0-3 (DEFAULT)
  1 = BYTE 4-7 (NOTE1)                    FIFO ADVANCE CONTROL
                                          ADD-ON INTERFACE
                                          0  0   BYTE 0 (DEFAULT)
                                          0  1   BYTE 1
                                          1  0   BYTE 2
     NOTE 1: D24 and D25 MUST BE ALSO "1" 1  1   BYTE 3
```

Table 4-3. Interrupt Control/Status Register

Bit	Description
31:24	FIFO and Endian Control (see Section 10.1.1).
23	Interrupt asserted. This read only status bit indicates that one or more of the four possible interrupt conditions is present. This bit is nothing more than the ORing of the interrupt conditions described by bits 19 through 16 of this register.
22	Reserved. Always zero.
21	Target Abort. This bit signifies that an interrupt has been generated due to the S5933 encountering a target abort during a PCI bus cycle while the S5933 was the current bus master. This bit operates as read or write one clear. A write to this bit with the data of "one" will cause this bit to be reset, a write to this bit with the data of "zero" will not change the state of this bit.
20	Master Abort. This bit signifies that an interrupt has been generated due to the S5933 encountering a Master Abort on the PCI bus. A master abort occurs when there is no target response to a PCI bus cycle (see Section 7.1.4.3). This bit operates as read or write one clear. A write to this bit with the data of "one" will cause this bit be reset, a write to this bit with the data of "zero" will not change the state of this bit.
19	Read Transfer Complete. This bit signifies that an interrupt has been generated due to the completion of a PCI bus master operation involving the transfer of data from the PCI bus to the add-on. This interrupt will occur when the Master Read Transfer Count register reaches zero. This bit operates as read or write one clear. A write to this bit with the data of "one" will cause this bit to be reset; a write to this bit with the data of "zero" will not change the state of this bit.
18	Write Transfer Complete. This bit signifies that an interrupt has been generated due to the completion of a PCI bus master operation involving the transfer of data to the PCI bus from the add-on. This interrupt will occur when the Master Write Transfer Count register reaches zero. This bit operates as read or write one clear. A write to this bit with the data of "one" will cause this bit to be reset; a write to this bit with the data of "zero" will not change the state of this bit.
17	Incoming Mailbox Interrupt. This bit is set when the mailbox selected by bits 12 through 8 of this register are written by the add-on interface. This bit operates as read or write one clear. A write to this bit with the data of "one" will cause this bit to be reset; a write to this bit with the data as "zero" will not change the state of this bit.
16	Outgoing Mailbox Interrupt. This bit is set when the mailbox selected by bits 4 through 0 of this register is read by the add-on interface. This bit operates as read or write one clear. A write to this bit with the data of "one" will cause this bit to be reset; a write to this bit with the data of "zero" will not change the state of this bit.
15	Interrupt on Read Transfer Complete. This bit enables the occurrence of an interrupt when the read transfer count reaches zero. This bit is read/write.
14	Interrupt on Write Transfer Complete. This bit enables the occurrence of an interrupt when the write transfer count reaches zero. This bit is read/write.
13	Reserved. Always zero.
12	Enable incoming mailbox interrupt. This bit allows a write from the incoming mailbox register identified by bits 11 through 8 to produce a PCI interface interrupt. This bit is read/write.
11:10	Incoming Mailbox Interrupt Select. This field selects which of the four incoming mailboxes is to be the source for causing an incoming mailbox interrupt. [00]b selects mailbox 1, [01]b selects mailbox 2, [10]b selects mailbox 3 and [11]b selects mailbox 4. This field is read/write.

4

Table 4-3. Interrupt Control/Status Register (Continued)

Bit	Description
9:8	Incoming Mailbox Byte Interrupt select. This field selects which byte of the mailbox selected by bits 10 and 11 above is to actually cause the interrupt. [00]b selects byte 0, [01]b selects byte 1, [10]b selects byte 2, and [11]b selects byte 3. This field is read/write.
7:5	Reserved, Always zero.
4	Enable outgoing mailbox interrupt. This bit allows a read by the add-on of the outgoing mailbox register identified by bits 3 through 0 to produce a PCI interface interrupt. This bit is read/write.
3:2	Outgoing Mailbox Interrupt Select. This field selects which of the four outgoing mailboxes is to be the source for causing an outgoing mailbox interrupt. [00]b selects mailbox 1, [01]b selects mailbox 2, [10]b selects mailbox 3 and [11]b selects mailbox 4. This field is read/write.
1:0	Outgoing Mailbox Byte Interrupt select. This field selects which byte of the mailbox selected by bits 3 and 2 above is to actually cause the interrupt. [00]b selects byte 0, [01]b selects byte 1, [10]b selects byte 2, and [11]b selects byte 3. This field is read/write.

4.10 BUS MASTER CONTROL/STATUS REGISTER (MCSR)

Register Name: Master Control/Status
PCI Address Offset: 3Ch
Power-up value: 000000E6h
Attribute: Read/Write, Read Only,
                   Write Only
Size: 32 bits

This register provides for overall control of this device. It is used to enable bus mastering for both data directions as well as providing a method to perform software resets.

The following PCI bus controls are available:

- Write Priority over Read

- Read Priority over Write

- Write Transfer Enable

- Write master requests on 4 or more FIFO words available (full)

- Read transfer enable

- Read master requests on 4 or more FIFO available (empty)

- Assert reset to Add-on

- Reset Add-on to PCI FIFO flags

- Reset PCI to Add-on FIFO flags

- Reset mailbox empty full status flags

- Write external non-volatile memory

The following PCI interface status flags are provided:

- PCI to Add-on FIFO FULL

- PCI to Add-on FIFO has four or more empty locations

- PCI to Add-on FIFO EMPTY

- Add-on to PCI FIFO FULL

- Add-on to PCI FIFO has four or more words loaded

- Add-on to PCI FIFO EMPTY

- PCI to Add-on Transfer Count = Zero

- Add-on to PCI Transfer Count = Zero

Figure 4-8. Bus Master Control/Status Register

Table 4-4. Bus Master Control/Status Register

Bit	Description
31:29	nvRAM Access Control. This field provides a method for access to the optional external non-volatile memory. Write operations are achieved by a sequence of byte operations involving these bits and the 8-bit field of bits 23 through 16. The sequence requires that the low-order address, high order address, and then a data byte are loaded in order. Bit 31 of this field acts as a combined enable and ready for the access to the external memory. D31 must be written to a 1 before an access can begin, and subsequent accesses must wait for bit D31 to become zero (ready). D31 D30 D29 W/R 0 X X W Inactive 1 0 0 W Load low address byte 1 0 1 W Load high address byte 1 1 0 W Begin write 1 1 1 W Begin read 0 X X R Ready 1 X X R Busy Cautionary note: The nonvolatile memory interface is also available for access by the add-on interface. Accesses by both the add-on and PCI bus to the nv memory are not directly supported by the S5933 device. Software must be designed to prevent the simultaneous access of nv memory to prevent data corruption within the memory and provide for accurate data retrieval.
28	Reserved. This bit must always be written with a zero.
27	Mailbox Flag Reset. Writing a one to this bit causes all mailbox status flags to become reset (EMPTY). It is not necessary to write this bit as zero because it is used internally to produce a reset pulse. Since reading of this bit will always produce zeros, this bit is write only.
26	Add-on to PCI FIFO Status Reset. Writing a one to this bit causes the Add-on to PCI (Bus master memory writes) FIFO empty flag to set indicating empty and the FIFO FULL flag to reset and the FIFO Four Plus word flag to reset. It is not necessary to write this bit as zero because it is used internally to produce a reset pulse. Since reading of this bit will always produce zeros, this bit is write only.
25	PCI to Add-on FIFO Status Reset. Writing a one to this bit causes the PCI to Add-on (Bus master memory reads) FIFO empty flag to set indicating empty and the FIFO FULL flag to reset and the FIFO Four Plus words available flag to set. It is not necessary to write this bit as zero because it is used internally to produce a reset pulse. Since reading of this bit will always produce zeros, this bit is write only.
24	Add-on pin reset. Writing a one to this bit causes the reset output pin to become active. Writing a zero to this pin is necessary to remove the assertion of reset. This register bit is read/write.
23:16	Non-volatile memory address/data port. This 8-bit field is used in conjunction with bit 31, 30 and 29 of this register to access the external non-volatile memory. The contents written are either low address, high address, or data as defined by bits 30 and 29. This register will contain the external non-volatile memory data when the proper read sequence for bits 31 through 29 is performed.

Table 4-4. Bus Master Control/Status Register (Continued)

Bit	Description
15	Reserved. Always zero.
14	Read Transfer Enable. This bit must be set to a one for S5933 PCI bus master read transfers to take place. Writing a zero to this location will suspend an active transfer. An active transfer is one in which the transfer count is not zero.
13	Read FIFO management scheme. When set to a 1, this bit causes the controller to refrain from requesting the PCI bus unless it has four or more vacant FIFO locations to fill. Once the controller is granted the PCI bus or is in possession of the bus due to the write channel, this constraint is not meaningful. When this bit is zero the controller will request the PCI bus if it has at least one vacant FIFO word.
12	Read versus Write priority. This bit controls the priority of read transfers over write transfers. When set to a 1 with bit D8 as zero this indicates that read transfers always have priority over write transfers; when set to a one with D8 as one, this indicates that transfer priorities will alternate equally between read and writes.
11	Reserved. Always zero.
10	Write Transfer Enable. This bit must be set to a one for PCI bus master write transfers to take place. Writing a zero to this location will suspend an active transfer. An active transfer is one in which the transfer count is not zero.
9	Write FIFO management scheme. When set to a one this bit causes the controller to refrain from requesting the PCI bus unless it has four or more FIFO locations filled. Once the S5933 controller is granted the PCI bus or is in possession of the bus due to the write channel, this constraint is not meaningful. When this bit is zero the controller will request the PCI bus if it has at least one valid FIFO word.
8	Write versus Read priority. This bit controls the priority of write transfers over read transfers. When set to a one with bit D12 as zero this indicates that write transfers always have priority over read transfers; when set to a one with D12 as one, this indicates that transfer prioritiies will alternate equally between writes and reads.
7	Add-on to PCI Transfer Count Equal Zero (RO). This bit is a one to signify that the write transfer count is all zeros.
6	PCI to Add-on Transfer Count Equals Zero (RO). This bit is a one to signify that the read transfer count is all zeros.
5	Add-on to PCI FIFO Empty. This bit is a one when the Add-on to PCI bus FIFO is completely empty.
4	Add-on to PCI 4+ words. This bit is a one when there are four or more FIFO words valid within the Add-on to PCI bus FIFO.
3	Add-on to PCI FIFO Full. This bit is a one when the Add-on to PCI bus FIFO is completely full.
2	PCI to Add-on FIFO Empty. This bit is a one when the PCI bus to add-on FIFO is completely empty.
1	PCI to Add-on FIFO 4+ spaces. This bit signifies that there are at least four empty words within the PCI to Add-on FIFO.
0	PCI to Add-on FIFO Full.This bit is a one when the PCI bus to Add-on FIFO is completely full.

Add-on Operation Registers 5

5.0   ADD-ON BUS OPERATION REGISTERS

The Add-on bus interface provides access to 18 DWORDs (72 bytes) of data, control and status information. All of these locations are accessed by asserting the add-on bus chip select pin (SELECT#) in conjunction with either the read or write control strobes (signal pin RD# or WR#). Access to the FIFO can also be achieved through use of the dedicated pins, RDFIFO# and WRFIFO#. The dedicated pins for control of the FIFO are provided to optionally implement Direct Memory Access (DMA) on the add-on bus, or to connect with an external FIFO.

This register group represents the primary method for communication between the add-on and PCI buses as viewed by the add-on. The flexibility of this arrangement allows a number of user-defined software protocols to be built. For example, data, software assigned commands, and command parameters can be exchanged between the PCI and add-on buses using either the mailboxes or FIFOs with or without handshaking interrupts. The register structure is very similar to that of the PCI operation register set. The major difference between the PCI bus and add-on bus register complement are the absence of bus master control registers (4) on the add-on side and the addition of two "pass-through" registers. Table 5-1 lists the Add-on interface registers.

5

Table 5-1. Operation Registers — Add-on Interface

Address	Abbreviation	Register Name
00h	AIMB1	Add-on Incoming Mailbox Register #1
04h	AIMB2	Add-on Incoming Mailbox Register #2
08h	AIMB3	Add-on Incoming Mailbox Register #3
0Ch	AIMB4	Add-on Incoming Mailbox Register #4
10h	AOMB1	Add-on Outgoing Mailbox Register #1
14h	AOMB2	Add-on Outgoing Mailbox Register #2
18h	AOMB3	Add-on Outgoing Mailbox Register #3
1Ch	AOMB4	Add-on Outgoing Mailbox Register #4
20h	AFIFO	Add-on FIFO port
24h	MWAR (note 1)	Bus Master Write Address Register
28h	APTA	Add-on Pass-Through Address
2Ch	APTD	Add-on Pass-Through Data
30h	MRAR (note 1)	Bus Master Read Address Register
34h	AMBEF	Add-on Mailbox Empty/Full Status
38h	AINT	Add-on Interrupt control
3Ch	AGCSTS	Add-on General Control and Status Register
58h	MWTC (note 1)	Bus Master Write Transfer Count
5Ch	MRTC (note 1)	Bus Master Read Transfer Count

Note 1: See Section 10.1.4.1 for PCI Bus Master Control from the add-on interface.

5.1 AIMBx – ADD-ON INCOMING MAILBOX REGISTERS

Register Names: Add-on Incoming Mailboxes 1-4

Add-on Address Offset: 00h, 04h, 08h, 0Ch

Power-up value: XXXXXXXXh

Attribute: Read Only

Size: 32 bits

These four DWORD registers provide a method for receiving data, commands, or command parameters from the PCI interface. Add-on read operations to these registers may be in any width (byte, word, or DWORD). These registers are read-only. Writes to this address space have no effect. Reading from one of these registers can optionally cause a PCI bus interrupt (if desired) when the PCI interrupt control/status register (Section 4.9) is properly configured.

5.2 AOMBx – ADD-ON OUTGOING MAILBOX REGISTERS

Register Names: Add-on Outgoing Mailboxes 1-4

Add-on Address Offset: 10h, 14h, 18h, 1Ch

Power-up value: XXXXXXXXh

Attribute: Read/Write

Size: 32 bits

These four DWORD registers provide a method for sending data, commands, or command parameters or status to the PCI interface. Add-on write operations to these registers may be in any width (byte, word, or DWORD). These registers may also be read. Writing to one of these registers can optionally cause a PCI bus interrupt (if desired) when the PCI interrupt control/status register (Section 4.9) is properly configured.

Mailbox 4, byte 3 only exists as device pins on the S5933 device when used with a serial nonvolatile memory. This byte is not available if a byte-wide nv memory is used.

5.3 AFIFO – ADD-ON FIFO REGISTER PORT

Register Name: Add-on FIFO Port

Add-on Address Offset: 20h

Power-up value: XXXXXXXXh

Attribute: Read/Write

Size: 32 bits

This location provides access to the bidirectional FIFO. Separate registers are involved when reading and writing to this location. Accordingly, it is not possible to read what was written to this location.

The sequence of filling and emptying this FIFO is established by the PCI interface interrupt control and Status Register (Section 4.9).

The FIFO's fullness may be observed by reading the master control/status register, AGCSTS, described in Section 5.8. Additionally, two signal pins are provided which reveal whether data is available (RDEMPTY) or space to write into the FIFO is available (WRFULL). These signals may be used to interface with user supplied DMA logic. Caution must be exercised when using these flags for FIFO transfers involving 64 bit endian conversion since the FIFO must operate on DWORD pairs.

5.4 ADD-ON CONTROLLED BUS MASTER WRITE ADDRESS REGISTER (MWAR)

Register Name: Master Write Address

Add-on Address Offset: 24h

Power-up value: 00000000h

Attribute: Read/Write

Size: 32 bits

This register is only accessible when add-on initiated bus mastering is enabled. See Section 10.2.3.3.

This register is used to establish the PCI address for data moving from the add-on bus to the PCI bus during PCI bus memory write operations. It consists of a 30-bit counter with the low-order two bits hardwired as zeros. Transfers may be any non-zero byte length as defined by the transfer count register, MWTC (Section 5.11), and must begin on a DWORD boundary. This DWORD boundary starting constraint is placed upon this controller's PCI bus master transfers so that byte lane alignment can be maintained between the S5933 controller's internal FIFO data path, the add-on interface, and the PCI bus.

Note: Applications which require a non-DWORD starting boundary will need to move the first few bytes under software program control (and without using the FIFO) to establish a DWORD boundary.

After the DWORD boundary is established the S5933 can begin the task of PCI bus master data transfers.

The Master Write Address Register is continually updated during the transfer process and will always be pointing to the next unwritten location. Reading of this register during a transfer process (done when the S5933 controller is functioning as a target, i.e. not a bus master) is permitted and may be used to monitor the progress of the transfer. During the address phase for bus master write transfers, the two least significant bits presented on the PCI bus pins AD[31:0] will always be zero. This identifies to the target memory that the burst address sequence will be in a linear order rather than in an Intel 486 or Pentium™ cache line fill sequence. Also, the PCI bus address bit A1 will always be zero when this controller is the bus master. This signifies to the target that the S5933 controller is burst capable and that the target should not arbitrarily disconnect after the first data phase of this operation.

Figure 5-1. Add-on Controlled Bus Master Write Address Register
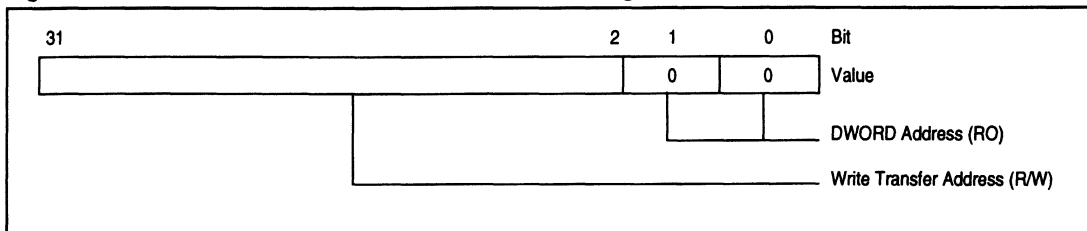
5.5 APTA – ADD-ON PASS-THRU ADDRESS REGISTER

Register Name: Add-on Pass-Thru Address

Add-on Address Offset: 28h

Power-up value: XXXXXXXXh

Attribute: Read Only

Size: 32 bits

This register is employed when a response is desired when one of the Base address decode regions (see Section 3.11) is selected during an active PCI bus cycle. When one of the base address decode registers 1-4 encounters a PCI bus cycle which selects the region defined by it, this device latches that current cycle's active address and asserts the signal PTATN# (Pass Thru ATtentioN). Wait states are generated on the PCI bus until either data is transferred or the PCI bus cycle is aborted by the initiator. (See Section 11.2)

This register provides a method for "live" data (registered) transfers. Intended uses include the emulating of other hardware as well as enabling the connection of existing external hardware to interface to the PCI bus through the S5933.

5.6 APTD – ADD-ON PASS-THRU DATA REGISTER

Register Name: Add-on Pass-Thru Data

Add-on Address Offset: 2Ch

Power-up value: XXXXXXXXh

Attribute: Read/Write

Size: 32 bits

This register, along with APTA described above, is employed when a response is desired should one of the Base address decode regions become selected during an active PCI bus cycle (see Section 3.11). When one of the base address decode registers 1-4 encounters a PCI bus cycle which selects the region defined by it, the APTA register will contain that current cycle's active address and the device asserts the signal PTATN# (Pass Through ATentioN). Wait states are generated on the PCI bus until this register is read (PCI bus writes) or this register is written (PCI bus reads).

5.7 ADD-ON CONTROLLED BUS MASTER READ ADDRESS REGISTER (MRAR)

Register Name: Master Read Address
Add-on Address Offset: 30h
Power-up value: 00000000h
Attribute: Read/Write
Size: 32 bits

This register is only accessible when add-on initiated bus mastering is enabled. See Section 10.2.3.3.

This register is used to establish the PCI address for data moving to the add-on bus from the PCI bus during PCI bus memory read operations. It consists of a 30-bit counter with the low-order two bits hardwired as zeros. Transfers may be any non-zero byte length as defined by the transfer count register, MRTC (Section 4.7) and must begin on a DWORD boundary. This DWORD boundary starting constraint is placed upon this controller's PCI bus master transfers so that byte lane alignment can be maintained between the S5395X controller's internal FIFO data path, the add-on interface and the PCI bus.

Note: Applications which require a non-DWORD starting boundary will need to move the first few bytes under software program control (and without using the FIFO) to establish a DWORD boundary. After the DWORD boundary is established the S5933 can begin the task of PCI bus master data transfers.

The Master Read Address Register is continually updated during the transfer process and will always be pointing to the next unread location. Reading of this register during a transfer process (done when the S5933 controller is functioning as a target—i.e., not a bus master) is permitted and may be used to monitor the progress of the transfer. During the address phase for bus master read transfers, the two least significant bits presented on the PCI bus AD[31:0] will always be zero. This identifies to the target memory that the burst address sequence will be in a linear order rather than in an Intel 486 or Pentium™ cache line fill sequence. Also, the PCI bus address bit A1 will always be zero when this controller is the bus master. This signifies to the target that the controller is burst capable and that the target should not arbitrarily disconnect after the first data phase of this operation.

Under certain circumstances, MRAR can be accessed from the add-on bus instead of the PCI bus. See Section 10.1.4.1.

5

Figure 5-2. Add-on Controlled Bus Master Read Address Register
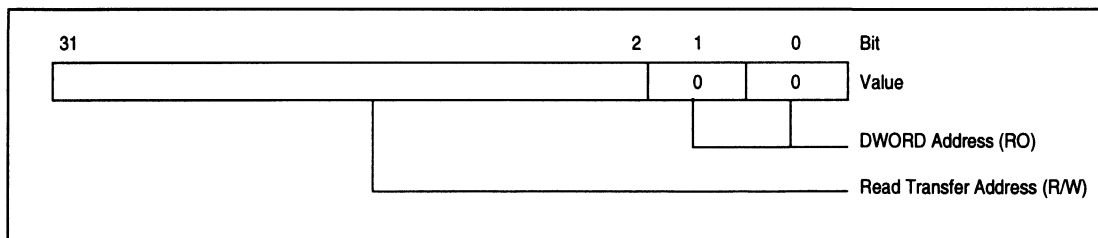
5.8   AMBEF – ADD-ON EMPTY/FULL STATUS REGISTER

Register Name: Add-on Mailbox Empty/Full Status

Add-on Address Offset: 34h

Power-up value: 00000000h

Attribute: Read Only

Size: 32 bits

This register provides empty/full visibility of each byte within the mailboxes. The empty/full status for the Outgoing mailboxes are displayed on the high order 16 bits and the empty/full status for the incoming mailboxes are presented on the low order 16 bits. A value of one signifies that a given mailbox had been written by the sourcing interface but had not yet been read by the corresponding destination interface. An incoming mailbox is defined as one in which data travels from the PCI bus into the add-on bus and an outgoing mailbox is defined as one where data goes OUT from the add-on bus to the PCI interface.

Figure 5-3. Add-on Mailbox Empty/Full Status Register

Applied Micro Circuits Corporation
6195 Lusk Blvd., San Diego, CA 92121 • (619) 450-9333
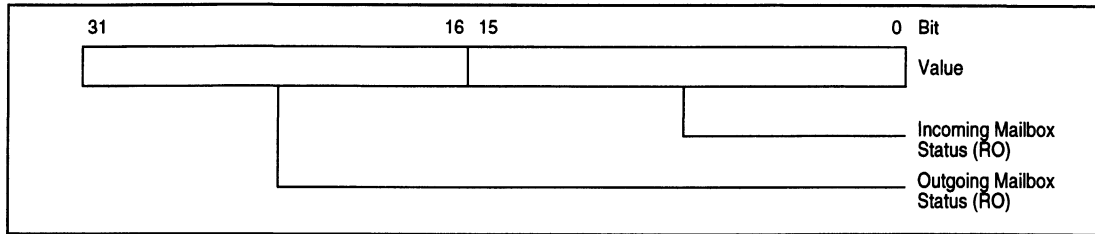
Table 5-2. Add-on Mailbox Empty/Full Status Register

Bit	Description
31:16	Outgoing Mailbox Status. This field indicates which outgoing mailbox registers have been written by the add-on bus interface but have not yet been read by the PCI bus. Each bit location corresponds to a specific byte within one of the four outgoing mailboxes. A value of one for each bit signifies that the specified mailbox byte is full, a value of zero signifies empty. The mapping of these status bits to bytes within each mailbox is as follows: Bit 31 = Outgoing mailbox 4 byte 3 Bit 30 = Outgoing mailbox 4 byte 2 Bit 29 = Outgoing mailbox 4 byte 1 Bit 28 = Outgoing mailbox 4 byte 0 Bit 27 = Outgoing mailbox 3 byte 3 Bit 26 = Outgoing mailbox 3 byte 2 Bit 25 = Outgoing mailbox 3 byte 1 Bit 24 = Outgoing mailbox 3 byte 0 Bit 23 = Outgoing mailbox 2 byte 3 Bit 22 = Outgoing mailbox 2 byte 2 Bit 21 = Outgoing mailbox 2 byte 1 Bit 20 = Outgoing mailbox 2 byte 0 Bit 19 = Outgoing mailbox 1 byte 3 Bit 18 = Outgoing mailbox 1 byte 2 Bit 17 = Outgoing mailbox 1 byte 1 Bit 16 = Outgoing mailbox 1 byte 0
15:00	Incoming Mailbox Status. This field indicates which incoming mailbox registers have been written by the PCI bus but not yet been read by the add-on interface. Each bit location corresponds to a specific byte within one of the four incoming mailboxes. A value of one for each bit signifies that the specified mailbox byte is full, a value of zero signifies empty. The mapping of these status bits to bytes within each mailbox is as follows: Bit 15 = Incoming mailbox 4 byte 3 Bit 14 = Incoming mailbox 4 byte 2 Bit 13 = Incoming mailbox 4 byte 1 Bit 12 = Incoming mailbox 4 byte 0 Bit 11 = Incoming mailbox 3 byte 3 Bit 10 = Incoming mailbox 3 byte 2 Bit 9 = Incoming mailbox 3 byte 1 Bit 8 = Incoming mailbox 3 byte 0 Bit 7 = Incoming mailbox 2 byte 3 Bit 6 = Incoming mailbox 2 byte 2 Bit 5 = Incoming mailbox 2 byte 1 Bit 4 = Incoming mailbox 2 byte 0 Bit 3 = Incoming mailbox 1 byte 3 Bit 2 = Incoming mailbox 1 byte 2 Bit 1 = Incoming mailbox 1 byte 1 Bit 0 = Incoming mailbox 1 byte 0

5

5.9 AINT – ADD-ON INTERRUPT CONTROL/ STATUS REGISTER

Register Name: Add-on Interrupt Control and Status

Add-on Address Offset: 38h

Power-up value: 00000000h

Attribute: Read/Write, Read/Write_One_Clear

Size: 32 bits

This register provides the method for choosing which conditions are to produce an interrupt on the add-on bus interface, a method for viewing the cause for the interrupt, and a method for acknowledging (removing) the interrupt's assertion.

Interrupt sources:

- One of the Incoming mailboxes (1,2,3 or 4) becomes full.

- One of the Outgoing mailboxes (1,2,3 or 4) becomes empty.

- Built-in self test issued.

- Write Transfer Count = zero

- Read Transfer Count = zero

- Target/Master Abort

Figure 5-4. Add-on Interrupt Control/Status Register

Table 5-3. Interrupt Control/Status Register

Bit	Description
31:24	Reserved. Always zero.
23	Interrupt asserted. This read-only status bit indicates that one or more interrupt conditions is present. This bit is nothing more than the ORing of the interrupt conditions described by bits, 20, 17 and 16 of this register.
22	Reserved. Always zero.
21	Master/Target Abort. This bit signifies that an interrupt has been generated due to the S5933 encountering a Master or Target abort during an S5933 initiated PCI bus cycle. This bit operates as read or write one clear. Writing a one to this bit causes it to be cleared. Writing a zero to this bit does nothing.
20	BIST. Built-In Self-Test interrupt. This interrupt occurs when a self test is initiated by the PCI interface writing of the BIST configuration register (Section 3.1.10). This bit will stay set until cleared by writing a one to this location. Self test completion codes may be passed to the PCI BIST register by writing to the AGCSTS register described in Section 5.10.
19	Read Transfer Complete. This bit signifies that an interrupt has been generated due to the completion of a PCI bus master operation involving the transfer of data from the PCI bus to the add-on. This interrupt will occur when the Master Read Transfer Count register reaches zero. This bit operates as read or write one clear. A write to this bit with the data of one will cause this bit to be reset; a write to this bit with the data of zero will not change the state of this bit.
18	Write Transfer Complete. This bit signifies that an interrupt has been generated due to the completion of a PCI bus master operation involving the transfer of data to the PCI bus from the add-on. This interrupt will occur when the Master Write Transfer Count register reaches zero. This bit operates as read or write one clear. A write to this bit with the data of one will cause this bit to be reset; a write to this bit with the data of zero will not change the state of this bit.
17	Outgoing Mailbox Interrupt. This bit sets when the mailbox selected by bits 12 through 8 of this register is read by the PCI interface. This bit operates as read or write one clear. A write to this bit with the data as one will cause this bit to be reset; a write to this bit with the data as zero will not change the state of this bit.
16	Incoming Mailbox Interrupt. This bit sets when the mailbox selected by bits 5 through 0 of this register are written by the PCI interface. This bit operates as read or write one clear. A write to this bit with the data of one will cause this bit to be reset; a write to this bit with the data as zero will not change the state of this bit.
15	Interrupt on Read Transfer Complete. This bit enables the occurrence of an interrupt when the read transfer count reaches zero. This bit is read/write.
14	Interrupt on Write Transfer Complete. This bit enables the occurrence of an interrupt when the write transfer count reaches zero. This bit is read/write.
13	Reserved. Always zero.
12	Enable outgoing mailbox interrupt. This bit allows a read by the PCI of the outgoing mailbox register identified by bits 11 through 8 to produce an add-on interface interrupt. This bit is read/write.
11:10	Outgoing Mailbox Interrupt Select. This field selects which of the four outgoing mailboxes is to be the source for causing an outgoing mailbox interrupt. [00]b selects mailbox 1, [01]b selects mailbox 2, [10]b selects mailbox 3 and [11]b selects mailbox 4. This field is read/write.
9:8	Outgoing Mailbox Byte Interrupt select. This field selects which byte of the mailbox selected by bits 11 and 10 above is to actually cause the interrupt. [00]b selects byte 0, [01]b selects byte 1, [10]b selects byte 2, and [11]b selects byte 3. This field is read/write.
7:5	Reserved, Always zero.

5

Table 5-3. Interrupt Control/Status Register (Continued)

Bit	Description
4	Enable incoming mailbox interrupt. This bit allows a write from the PCI bus to the incoming mailbox register identified by bits 3 through 0 to produce an add-on interface interrupt. This bit is read/write.
3:2	Incoming Mailbox Interrupt Select. This field selects which of the four incoming mailboxes is to be the source for causing an incoming mailbox interrupt. [00]b selects mailbox 1, [01]b selects mailbox 2, [10]b selects mailbox 3 and [11]b selects mailbox 4. This field is read/write.
1:0	Incoming Mailbox Byte Interrupt select. This field selects which byte of the mailbox selected by bits 3 and 2 above is to actually cause the interrupt. [00]b selects byte 0, [01]b selects byte 2, and

5.10 AGCSTS – ADD-ON GENERAL CONTROL/ STATUS REGISTER

Register Name: Add-on General Control and Status

Add-on Address Offset: 3Ch

Power-up value: 000000F4h
                    (PCI initiated bus mastering)
                    00000034h
                    (Add-on initiated bus mastering)

Attribute: Read/Write, Read Only, Write Only

Size: 32 bits

This register provides for overall control of the add-on portion of this device. It is used to provide a method to perform software resets of the mailbox and FIFO flags.

The following add-on controls are provided:

- Reset PCI to Add-on FIFO flags
- Reset Add-on to PCI FIFO flags
- Reset mailbox empty full status flags
- Write/read external non-volatile memory.

The following status flags are provided to the add-on:

- Add-on to PCI FIFO FULL
- Add-on to PCI FIFO has four or more empty locations
- Add-on to PCI FIFO EMPTY
- PCI to Add-on FIFO FULL
- PCI to Add-on FIFO has four or more words loaded
- PCI to Add-on FIFO EMPTY

Figure 5-5. Add-on General Control/Status Register

Table 5-4. Add-on General Control/Status Register

Bit	Description
31:29	nvRAM/EPROM Access Control. This field provides a method for access to the optional, external non-volatile memory. Write operations are achieved by a sequence of byte operations involving these bits and the 8-bit field of bits 23 through 16. The sequence requires that the low-order address, high-order address, and then a data byte be loaded in order. Bit 31 of this field acts as an enable/clock and ready for the access to the external memory. D31 must be written to a 1 before an access can begin, and subsequent accesses must wait for bit D31 to become zero (ready).

D31	D30	D29	W/R	
0	X	X	W	Inactive
1	0	0	W	Load low address byte
1	0	1	W	Load high address byte
1	1	0	W	Begin write
1	1	1	W	Begin read
0	X	X	R	Ready
1	X	X	R	Busy

Bit	Description
	Cautionary note: The non-volatile memory interface is also available for access by the PCI bus interface. Accesses by both the add-on and PCI bus to the nv memory are not directly supported by this component. Software must be designed to prevent the simultaneous access of nv memory to prevent data corruption within the memory and provide for accurate data retrieval.
28	Transfer Count Enable. When set, transfer counts are used for add-on initiated bus master transfers. When clear, transfer counts are ignored.
27	Mailbox Flag Reset. Writing a 1 to this bit causes all mailbox status flags to become reset (EMPTY). It is not necessary to write this bit as 0 because it is used internally to produce a reset pulse. Since reading of this bit will always produce zeros, this bit is write only.
26	PCI to Add-on FIFO Status Reset. Writing a 1 to this bit causes the Inbound (Bus master reads) FIFO empty flag to set indicating empty and the FIFO FULL flag to reset and the FIFO Four Plus spaces flag to set. It is not necessary to write this bit as 0 because it is used internally to produce a reset pulse. Since reading of this bit would always produce zeros, this bit is write only.
25	Add-on to PCI FIFO Status Reset. Writing a one to this bit causes the Outbound (Bus master writes) FIFO empty flag to set indicating empty and the FIFO FULL flag to reset and the FIFO Four Plus words available flag to reset. It is not necessary to write this bit as zero because it is used internally to produce a reset pulse. Since reading of this bit would always produce zeros, this bit is write only.
24	Reserved. Always zero.
23:16	Non-volatile memory address/data port. This 8-bit field is used in conjunction with bit 31, 30 and 29 of this register to access the external non-volatile memory. The contents written are either low address, high address, or data as defined by bits 30 and 29. This register will contain the external non-volatile memory data when the proper read sequence for bits 31 through 29 is performed.
15:12	BIST condition code. This field is directly connected to the PCI configuration self test register described in Section 3.10. Bit 15 through 12 maps with the BIST register bits 3 through 0, respectively.
11:8	Reserved. Always zero.

Table 5-4. Add-on General Control/Status Register (Continued)

Bit	Description
7	Add-on to PCI Transfer Count Equal Zero (RO). This bit as a one signifies that the write transfer count is all zeros. Only when add-on initiated bus mastering is enabled.
6	PCI to Add-on Transfer Count Equals Zero (RO). This bit as a one signifies that the read transfer count is all zeros. Only when add-on initiated bus mastering is enabled.
5	PCI to Add-on FIFO Empty. This bit is a 1 when the PCI to Add-on FIFO is empty.
4	PCI to Add-on FIFO 4+ spaces. This bit is a 1 when there are four or more open spaces in the PCI to Add-on FIFO.
3	PCI to Add-on FIFO Full. This bit is a 1 when the PCI to Add-on FIFO is full.
2	Add-on to PCI FIFO Empty. This bit is a 1 when the Add-on to PCI FIFO is empty.
1	Add-on PCI FIFO 4+ words. This bit is a 1 when there are four or more full locations in the Add-on to PCI FIFO.
0	Add-on to PCI FIFO Full. This bit is a 1 when the Add-on to PCI FIFO is full.

5

5.11 ADD-ON CONTROLLED BUS MASTER WRITE TRANSFER COUNT REGISTER (MWTC)

Register Name: Master Write Transfer Count

Add-on Address Offset: 58h

Power-up value: 00000000h

Attribute: Read/Write

Size: 32 bits

This register is only accessible when add-on initiated bus mastering is enabled. See Section 10.2.3.3.

The master write transfer count register is used to convey to the S5933 controller the actual number of bytes that are to be transferred. The value in this register is decremented with each bus master PCI write operation until the transfer count reaches zero.

Upon reaching zero, the transfer operation ceases and an interrupt may be optionally generated to either the PCI or add-on bus interface. Transfers which are not whole multiples of DWORDs in size result in a partial word ending cycle. This partial word ending cycle is possible since all bus master transfers for this controller are required to begin on a DWORD boundary.

Figure 5-6. Add-on Controlled Bus Master Write Transfer Count Register
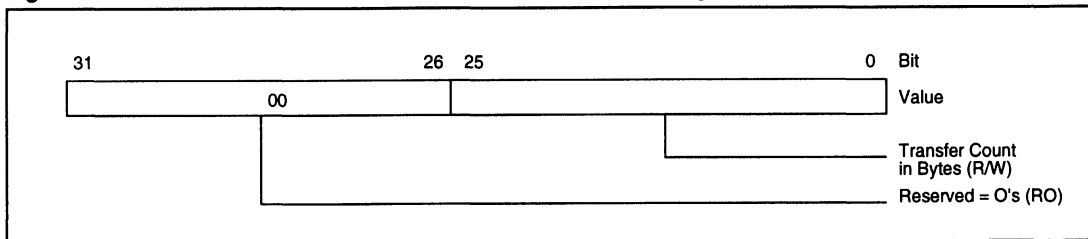
5.12 ADD-ON CONTROLLED BUS MASTER READ TRANSFER COUNT REGISTER (MRTC)

Register Name: Master Read Transfer Count

Add-on Address Offset: 5Ch

Power-up value: 00000000h

Attribute: Read/Write

Size: 32 bits

This register is only accessible when add-on initiated bus mastering is enabled. See Section 10.2.3.3.

The master read transfer count register is used to convey to the PCI controller the actual number of bytes that are to be transferred. The value in this register is decremented with each bus master PCI read operation until the transfer count reaches zero. Upon reaching zero, the transfer operation ceases and an interrupt may be optionally generated to either the PCI or add-on bus interface. Transfers which are not whole multiples of DWORDs in size result in a partial word ending cycle. This partial word ending cycle is possible since all bus master transfers for this controller are required to begin on a DWORD boundary.

Figure 5-7. Add-on Controlled Bus Master Read Transfer Count Register
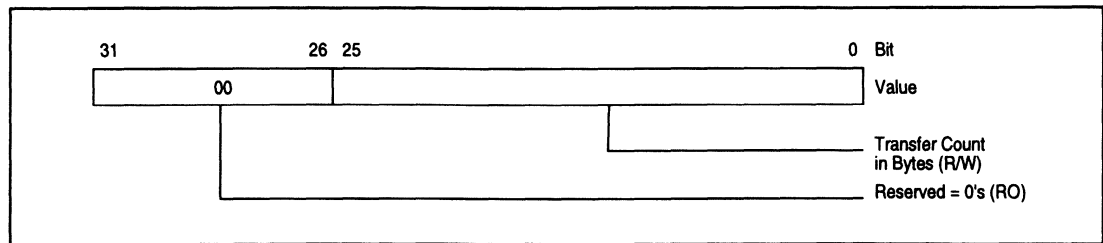
Initialization 6

6.0 INITIALIZATION

All PCI bus agents and bridges are required to implement PCI Configuration Registers. When multiple PCI devices are present, these registers must be unique to each device in the system. The specified PCI procedure for uniquely selecting a device's configuration space involves a dedicated signal, called IDSEL, connected to each motherboard PCI bus device and PCI slot (see Section 6.4).

The host executes configuration cycles after reset to each device on the PCI bus. The configuration registers provide information on PCI agent operation and memory or I/O space requirements. These allow the PCI BIOS to enable the device and locate it within system memory or I/O space.

After a PCI reset, the S5933 can be configured for a specific application by downloading device setup information from an external non-volatile memory into the device Configuration Registers. The S5933 can also be used in a default configuration, with no external boot device. Default states for the user-definable PCI Configuration Registers are described in Chapter 3.

When using a non-volatile boot memory to customize operation, 64 bytes are required for S5933 setup information. The rest of the boot device may be used to implement an Expansion BIOS (Section 6.5), if desired. Some of the setup information is used to initialize the S5933 PCI Configuration Registers, other information is not downloaded into registers, but is used to define S5933 operation (FIFO interface, pass-thru operation, etc.).

6.1 PCI RESET

Immediately following the assertion of the PCI RST# signal, the add-on reset output SYSRST# is asserted. Immediately following the deassertion of RST#, SYSRST# is deasserted. The add-on reset output may be used to initialize state machines, reset add-on microprocessors, or reset other add-on logic devices.

All S5933 Operation Registers and Configuration Registers are initialized to their default states at reset. The default values for the Configuration Registers may be overwritten with the contents of an external nv boot memory during device initialization, allowing a custom device configuration. This process is described, in detail, in Sections 6.2 and 6.3. Configuration accesses by the host CPU to the S5933 produce PCI bus wait states until one of the following events occurs:

- The S5933 identifies that there is no valid boot memory (and default Configuration Register values are used).

- The S5933 finishes downloading all configuration information from a valid boot memory.

6.2 LOADING FROM BYTE-WIDE NV MEMORIES

6

The SNV input on the S5933 indicates what type of external boot-load device is present (if any). If SNV is tied low, a byte-wide nv memory is assumed. In this case, immediately after the PCI bus reset is deasserted, the address 0040h is presented on the nv memory interface address bus EA[15:0]. Eight PCI clocks later (240 ns at 33 MHz), data is read from the nv memory data bus EQ[7:0] and address 0041h is presented. After an additional eight PCI clocks, data is again read from EQ7:0. If both accesses read are all ones (FFh), it implies an illegal Vendor ID value, and the external nv memory is not valid or not present. In this situation, the AMCC default configuration values are used (see Chapter 3).

If either of the accesses to address 0040h and 0041h contain zeros (not FFh), the next accesses are to locations 0050h, 0051h, 0052h, and 0053h. At these locations, the data must be C0h (or C1h or C2h), FFh, E8h, and 10h, respectively, for the external nv memory to be valid. Once a valid external nv memory has been recognized, it is read, sequentially, from location 0040h to 007Fh. The appropriate data is loaded into the PCI Configuration Registers as described in Chapter 3. Some of the boot device data is not downloaded into Configuration Registers, but is used to enable features and configure S5933 operation. Upon completion of this procedure, the boot-load sequence terminates and PCI configuration accesses to the S5933 are acknowledged with the PCI Target Ready (TRDY#) output.

Table 6-1 lists the required nv memory contents for a valid configuration nv memory device.

Table 6-1. Valid External Boot Memory Contents

Address	Data	Notes
0040h-41h	not FFFFh	This is the location that the S5933 PCI Controller will load a customized vendor ID. (FFFFh is an illegal vendor ID.)
0050h	C2h, C1h or C0h	This is the least significant byte of the region which initializes the base address register #0 of the S5933 configuration register (Section 3.11). A value of C1h assigns the 16 DWORD locations of the PCI operation registers into I/O space, a value of C0h defines memory space, a value of C2h defines memory space below 1 Mbyte.
0051	FFh	Required.
0052h	E8h	Required.
0053h	10h	Required.

6.3 LOADING FROM SERIAL NV MEMORIES

SNV tied high indicates that a serial nv memory (or no external device) is present. When serial nv memories are used, data transfer is performed through a two-wire, bidirectional data transfer protocol as defined by commercial serial EEPROM/Flash offerings. These devices have the advantages of low pin counts, small package size, and economical price.

A serial nv memory is considered valid if the first serial accesses contain the correct per-byte acknowledgments (see Figure 6-3). If the serial per-byte acknowledgment is not observed, the S5933 determines that no external serial nv memory is present and the AMCC default Configuration Register values are used (see Chapter 3).

Two pins are used to transfer data between the S5933 PCI controller and the external serial memory: a serial clock pin, SCL, and a serial data pin, SDA. The serial clock pin is an output from the S5933, and the serial data pin is bidirectional. The serial clock is derived by dividing the PCI bus clock by 512. This means that the frequency of the serial clock is approximately 65 kHz for a 33-MHz PCI bus clock.

Communications with the serial memory involve several clock transitions. A start event signals the beginning of a transaction and is immediately followed by an address transfer. Each address/data transfer consists of 8 bits of information followed by a 1-bit acknowledgment. When the exchange is complete, a stop event is issued. Figure 6-1 shows the unique relationship defining both a start and stop event. Figure 6-2 shows the required timing for address/data with respect to the serial clock.

For random accesses, the sequence involves one clock to define the start of the sequence, eight clocks to send the slave address and read/write command, followed by a one-clock acknowledge, and so on. Figure 6-3 shows the sequence for a random write access requiring 29 serial clock transitions. At the clock speed for the S5933, this corresponds to one byte of data transferred approximately every 0.5 milliseconds. Read accesses may be either random or sequential. Random read access requires a dummy write to load the word address and require 39 serial clock transitions. Figure 6-4 shows the sequence for a random byte read.

To initialize the S5933 controller's PCI Configuration Registers, the smallest serial device necessary is a 128 x 8 organization. Although the S5933 controller only requires 64 bytes, these bytes must begin at a 64-byte address offset (0040h through 007Fh). This offset constraint permits the configuration image to be shared with a memory containing expansion BIOS code and the necessary preamble to identify an expansion BIOS (see Section 6.5). The largest serial device which may be used is 2 Kbytes.

Figure 6-1. Serial Interface Definition of Start and Stop
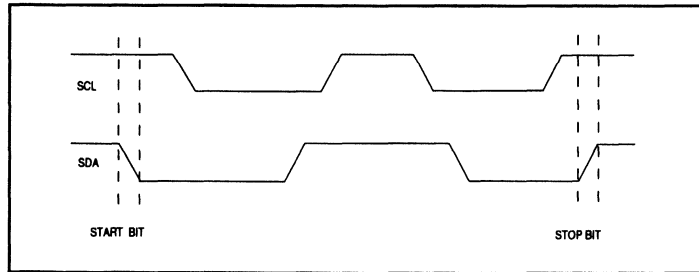
Figure 6-2. Serial Interface Clock/Data Relationship
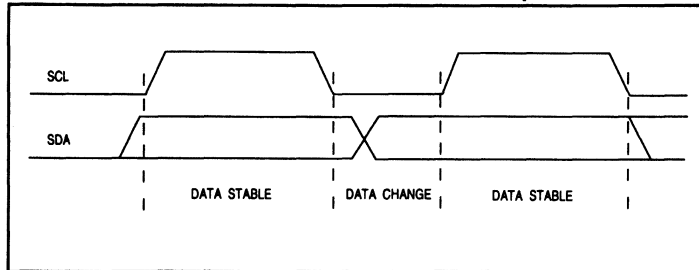
Figure 6-3. Serial Interface Byte Access — Write
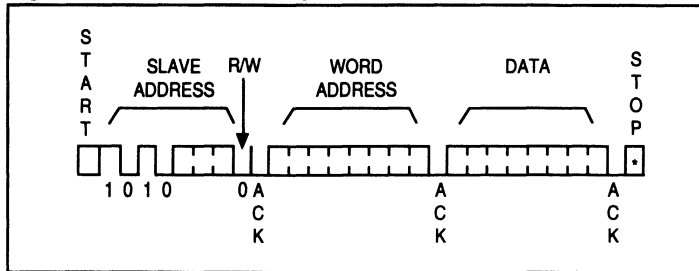
Figure 6-4. Serial Interface Byte Access — Read
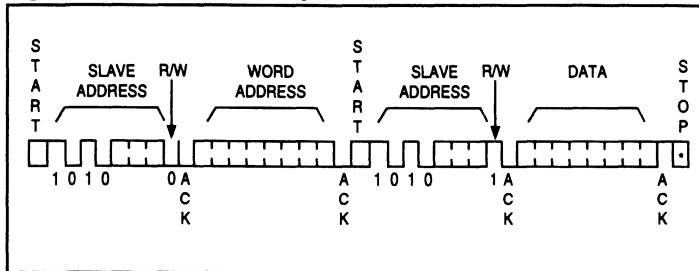
6.4 PCI BUS CONFIGURATION CYCLES

Cycles beginning with the assertion IDSEL and FRAME# along with the two configuration command states for C/BE[3:0] (configuration read or write) access an individual device's configuration space. During the address phase of the configuration cycle just described, the values of AD0 and AD1 identify if the access is a Type 0 configuration cycle or a Type 1 configuration cycle. Type 0 cycles have AD0 and AD1 equal to 0 and are used to access PCI bus agents. Type 1 configuration cycles are intended only for bridge devices and have AD0 as a 1 with AD1 as a 0 during the address phase.

The S5933 PCI device is a bus agent (not a bridge) and responds only to a Type 0 configuration accesses. Figure 6-5 depicts the state of the AD bus during the address phase of a Type 0 configuration access. The S5933 controller does not support the multiple function numbers field (AD[10:8]) and only responds to the all-zero function number value.

The configuration registers for the S5933 PCI controller (described in Chapter 3) can only be accessed under the following conditions:

- IDSEL high (PCI slot unique signal which identifies access to configuration registers) along with FRAME# low.

- Address bits A0 and A1 are 0 (Identifies a Type 0 configuration access).

- Address bits A6-A7 are 0 (Only the first 64 bytes select configuration registers).

- Address bits A8, A9, and A10 are 0 (Function number field of zero supported).

- Command bits, C/BE[3:0]# must identify a configuration cycle command (101X).

Figure 6-6 describes the signal timing relationships for configuration read cycles. Figure 6-7 describes configuration write cycles.

Figure 6-5. PCI AD Bus Definition During a Type 0 Configuration Access
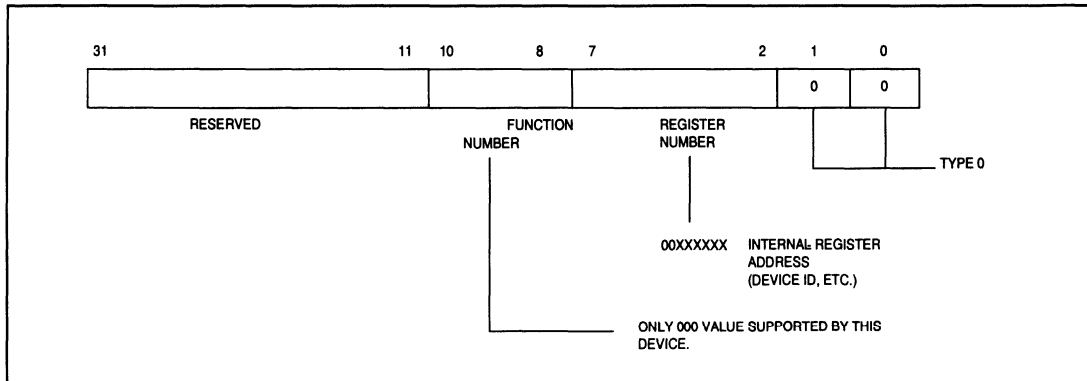
Figure 6-6. Type 0 Configuration Read Cycles

```
                1        2        3        4        NOTE

PCI CLOCK

FRAME #     (I)                                     IF FRAME # STILL ASSERTED
                                                    DURING CLOCK 2, CONTROLLER
                                                    ASSERTS STOP# DURING 3

AD [31:0]   (I)   ADDRESS         DATA      (T)     DRIVEN BY CONTROLLER
                                                    DURING CLOCK 3

C/BE [3:0]# (I)   CONFIG. READ CMD   BYTE ENABLES

IRDY#       (I)

TRDY#       (T)                                     DRIVEN BY CONTROLLER
                                                    DURING CLOCKS 2,3 +4

IDSEL       (I)

DEVSEL#     (T)                                     DRIVEN BY CONTROLLER
                                                    DURING CLOCKS 2,3 +4

            SELECT                (I) = DRIVEN BY INITIATOR
            CONDITION             (T) = DRIVEN BY TARGET
```
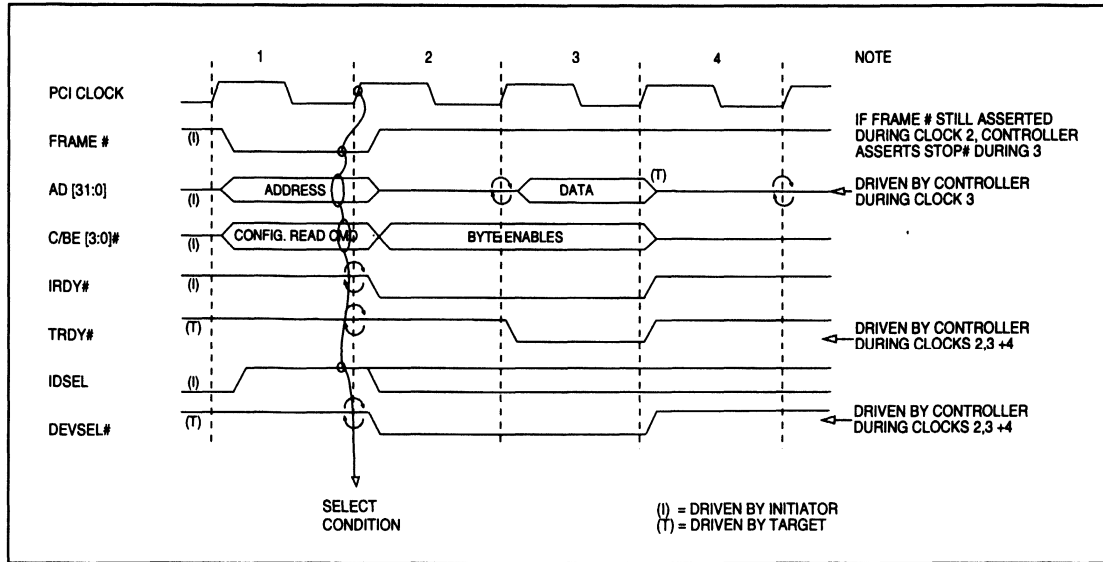
Figure 6-7. Type 0 Configuration Write Cycles

```
                1        2        3        4        NOTE

PCI CLOCK

FRAME #     (I)                                     FRAME # DEASSERTED
                                                    IN CLOCK 2, SIGNIFIES
                                                    ONLY ONE DATA PHASE

AD [31:0]   (I)   ADDRESS     DATA

C/BE [3:0]# (I)   CONFIG WRITE CMD  BYTE ENABLES

IRDY#       (I)

TRDY#       (T)                                     DRIVEN BY CONTROLLER
                                                    DURING CLOCKS 2+3

IDSEL       (I)

DEVSEL#     (T)                                     DRIVEN BY CONTROLLER
                                                    DURING CLOCKS 2+3

            SELECT                (I) = DRIVEN BY INITIATOR
            CONDITION             (T) = DRIVEN BY TARGET
```

6
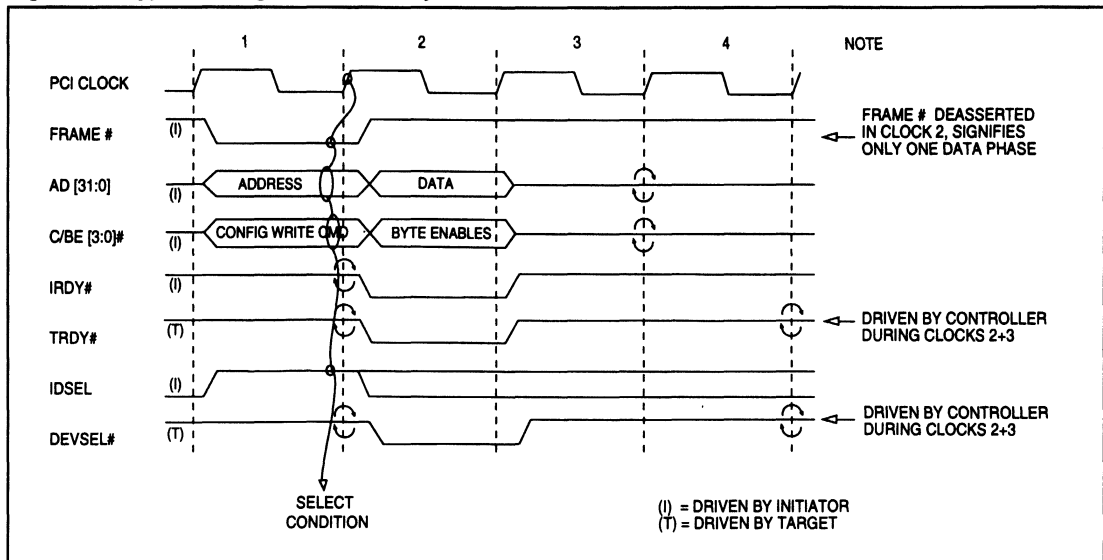
6.5 EXPANSION BIOS ROMS

This section provides an example of a typical PC-compatible expansion BIOS ROM. Address offsets 0040h through 007Fh represent the portion of the external nv memory used to boot-load the S5933 controller. Whether the expansion ROM is intended to be executable code is determined by the contents of the first three locations (starting at offset 0h) and a byte checksum over the defined length. The defined length is specified in the byte at address offset 0002h. Table 6-2 lists each field location by its address offset, its length, its value, and description.

Table 6-2. PC Compatible Expansion ROM

Byte Offset	Byte Length	Binary Value	Description	Example
0h	1	55h	BIOS ROM signature byte 1	55h
1h	1	AAh	BIOS ROM signature byte 2	AAh
2h	1	var.	Length in multiples of 512 bytes	01h
3h	4	var.	Entry point for INIT function.	
7h-17h	17h	var.	Reserved (application unique data)	
18h-19h	2	var.	Pointer to PCI Data Structure (see Table 6-3)	
20h-3Fh	32h	var	user-defined	
The following represents the boot-load image for the S5933 controller's PCI configuration register :				
40h	2	[your vendor ID]	(see Section 3.1)	1234h
42h	2	[your device ID]	(see Section 3.2)	5678h
44h	1	not used		xxh
45h	1	[Bus Master Config.]	(see Section 10.3.1)	80h
46h	2	not used		xxxxh
48h	1	[your revision ID]	(see Section 3.5)	00h
49h	3	[your class code]	(see Section 3.6)	FF0000h
4Ch	1	not used		xxh
4Dh	1	[your latency timer #]	(see Section 3.8)	00h
4Eh	1	[your header type]	(see Section 3.9)	00h
4Fh	1	[self-test if desired]	(see Section 3.10)	80h or 00h
50h	1	C0h, C1h or C2h	(required, see Section 3.11 and Table 6-1)	C0h, C1h or C2h
51h	1	FFh	(required per Table 6-1)	FFh
52h	1	E8h	(required per Table 6-1)	E8h
53h	1	10h	(required per Table 6-1)	10h
54h	4	[base addr. #1]	(see Section 3.11)	00000000h
58h	4	[base addr. #2]	" " "	00000000h
5Ch	4	[base addr. #3]	" " "	00000000h
60h	4	[base addr. #4]	" " "	00000000h
64h	4	[base addr. #5]	" " "	00000000h

Table 6-2. PC Compatible Expansion ROM (Continued)

Byte Offset	Byte Length	Binary Value	Description	Example
68h	8	not used		8 DUP(?)
70h	4	[Expansion	(see Section 3.12)	
		ROM base addr.]	(example shows 32K bytes)	FFFF8001h
74h	8	not used		8 DUP(?)
7Ch	1	[Interrupt line]	(see Section 3.13)	0Ch
7Dh	1	[Interrupt pin]	(see Section 3.14)	01h
7Eh	1	[Min-Grant]	(see Section 3.15)	00h
7Fh	1	[Max_lat]	(see Section 3.16)	00h
80h — (1FFh), or (2FFh), or (3FFh), etc.		application specific		
		Byte checksum, location dependent on value for length field at offset 0002h.		

A 16-bit pointer at location 18h of the PC expansion ROM identifies the start offset of the PCI data structure. The PCI data structure is shown in Table 6-3 and contains various vendor, product, and program evolutions. If a valid external nv memory is identified by the S5933 (as described in Sections 6.2 and 6.3), the PCI data structure is used to configure the S5933. The PCI data structure is not necessary for this device to operate. If no external nv memory is implemented, the S5933 boots with the default configuration values described in Chapter 3.

Note: If a serial BIOS ROM is used, the access time for large serial devices should be considered, since it may cause a lengthy system delay during initialization. For example, a 2-Kbyte serial device takes about 1 second to be read. Many systems, even when BIOS ROMs are ultimately shadowed into system RAM, may read this memory space twice (once to validate its size and checksum, and once to move it into RAM). Execution directly from a serial BIOS ROM, although possible, may be unacceptably slow.

Table 6-3. PCI Data Structure

Byte Offset	Byte Length	Binary Value	Description
0h	4	'PCIR'	Signature, the ASCII string 'PCIR' where 'P' is at offset 0, 'C' at offset 1, and so on.
4h	2	var.	Vendor Identification
6h	2	var.	Device Identification
8h	2	var.	Pointer to Vital Product Data
Ah	2	var.	PCI Data Structure Length (starts with signature field)
Ch	1	var.	PCI Data Structure Revision (=0 for this definition)
Dh	3	var.	Class Code
10h	2	var.	Image Length
12h	2	var.	Revision Level
14h	1	var.	Code Type
15h	1	var.	Indicator (bit D7=1 signifies "last image")
16h	2	0000h	Reserved

PCI Bus Interface 7

7.0 PCI BUS INTERFACE

This section describes the various events which occur on the S5933 PCI bus interface. Since the S5933 controller functions as both a target (slave) and an initiator (master), signal timing detail is given for both situations. Section 7.1 presents the signal relationships involved in performing basic read or write transfers on the PCI bus. Section 7.1 also describes the different ways these cycles may complete. In Section 7.2 the events associated with acquiring control of the PCI bus and becoming a PCI bus initiator (master) are detailed. PCI Interrupts are described in Section 7.3.

7.1 PCI BUS TRANSACTIONS

Because the PCI bus has multiplexed address/data pins, AD[31:0], each PCI bus transaction consists of two phases: Address and Data. An address phase is defined by the clock period when the signal FRAME# transitions from inactive (high) to active (low). During the address phase, a bus command is also driven by the initiator on signal pins C/BE[3:0]#. If the command indicates a PCI read, the clock cycle following the address phase is used to perform a "bus turn-around" cycle. A turn-around cycle is a clock period in which the AD bus is not driven by the initiator or the target device. This is used to avoid PCI bus contention. For a write command, a turn-around cycle is not needed, and the bus goes directly from the address phase to the data phase.

All PCI bus transactions consist of an address phase (described above), followed by one or more data phases. The address phase is only one PCI clock long and the bus cycle information (address and command)

is latched internally by the S5933. The number of data phases depends on how many data transfers are desired or are possible with a given initiator-target pair. A data phase consists of at least one PCI clock. FRAME# is deasserted to indicate that the final data phase of a PCI cycle is occurring. Wait states may be added to any data phase (each wait state is one PCI clock).

The PCI bus command presented on the C/BE[3:0]# pins during the address phase can represent 16 possible states. Table 7-1 lists the PCI commands and identifies those which are supported by the S5933 controller as a target and those which may be produced by the S5933 controller as an initiator. A "Yes" in the "Supported As Target" column in Table 7-1 indicates the S5933 controller asserts the signal DEVSEL# when that command is issued along with the appropriate PCI address (see Sections 3.11 and 6.4). Two commands are supported by the S5933 controller as an initiator: Memory Read and Memory Write.

The completion or termination of a PCI cycle can be signaled in several ways. In most cases, the completion of the final data phase is indicated by the assertion of ready signals from both the target (TRDY#) and initiator (IRDY#) while FRAME# is inactive. In some cases, the target is not be able to continue or support a burst transfer and asserts the STOP# signal. This is referred to as a target disconnect. There are also cases where an addressed device does not exist, and the signal DEVSEL# never becomes active. When no DEVSEL# is asserted in response to a PCI cycle, the initiator is responsible for ending the cycle. This is referred to as a master abort. The bus is returned to the idle phase when both FRAME# and IRDY# are deasserted.

Table 7-1. Supported PCI Bus Commands

C/BE[3:0]#	Command Type	Supported As Target	Supported As Initiator
0000	Interrupt Acknowledge	No	No
0001	Special Cycle	No	No
0010	I/O Read	Yes	No
0011	I/O Write	Yes	No
0100	Reserved	No	No
0101	Reserved	No	No
0110	Memory Read	Yes	Yes
0111	Memory Write	Yes	Yes
1000	Reserved	No	No
1001	Reserved	No	No
1010	Configuration Read	Yes	No
1011	Configuration Write	Yes	No
1100	Memory Read Multiple	Yes (1)	No
1101	Reserved	No	No
1110	Memory Read Line	Yes (1)	No
1111	Memory Write & Invalidate	Yes (2)	No

Note 1: Memory Read Multiple and Read Line are treated as Memory Reads.
Note 2: Memory Write & Invalidate commands are treated as Memory Writes.

7.1.1 PCI Burst Transfers

The PCI bus, by default, expects burst transfers to be executed. To successfully perform a burst transfer, both the initiator and target must order their burst address sequence in an identical fashion. There are two different ordering schemes: linear address incrementing and 80486 cache line fill sequencing. The exact ordering scheme for a bus transaction is defined by the state of the two least significant AD lines during the address phase. The decoding for these lines is shown below:

AD[1:0]	Burst Order
0 0	Linear sequence
0 1	Reserved
1 0	Cacheline Wrap Mode
1 1	Reserved

The S5933 supports both the linear and the cache line burst ordering. When the S5933 controller is an initiator, it always employs a linear ordering.

Some accesses to the S5933 controller (as a target) can not be burst transfers. For example, the S5933 does not allow burst transfers when accesses are made to the configuration or operation registers (including the FIFO as a target). Attempts to perform burst transfers to these regions cause STOP# to be asserted during the first data phase. The S5933 completes the initial data phase successfully, but asserting STOP# indicates that the next access needs to be a completely new cycle. Accesses to memory or I/O regions defined by the Base Address Registers 1-4 (see Section 3.11) may be bursts, if desired.

7.1.2 PCI Read Transfers

The S5933 responds to PCI bus memory or I/O read transfers when it is selected (target). As a PCI bus initiator, the S5933 controller may also produce PCI bus memory read operations.

Figure 7-1 depicts the fastest burst read transfer possible for the PCI bus. The timings shown in Figure 7-1 are representative of the S5933 as a PCI initiator with a fast, zero-wait-state memory target. The signals driven by the S5933 during the transfer are FRAME#, C/BE[3:0]#, and IRDY#. The signals driven by the target are DEVSEL# and TRDY#. AD[31:0] are driven by both the target and initiator during read transactions (only one during any given clock). Clock period 2 is a required bus turn-around clock which ensures bus contention between the initiator and target does not occur.

Targets drive DEVSEL# and TRDY# after the end of the address phase (boundary of clock periods 1 and 2 of Figure 7-1). TRDY# is not driven until the target can provide valid data for the PCI read. When the S5933 becomes the PCI initiator, it attempts to perform sustained zero-wait state burst reads until one of the following occurs:

- The memory target aborts the transfer
- PCI bus grant (GNT#) is removed
- The PCI to add-on FIFO becomes full
- A higher priority (add-on to PCI) S5933 transfer is pending (if programmed for priority)
- The read transfer byte count reaches zero
- Bus mastering is disabled from the add-on interface

Figure 7-1. Zero Wait State Burst Read PCI Bus Transfer (S5933 as Initiator)

Applied Micro Circuits Corporation
6195 Lusk Blvd., San Diego, CA 92121 • (619) 450-9333
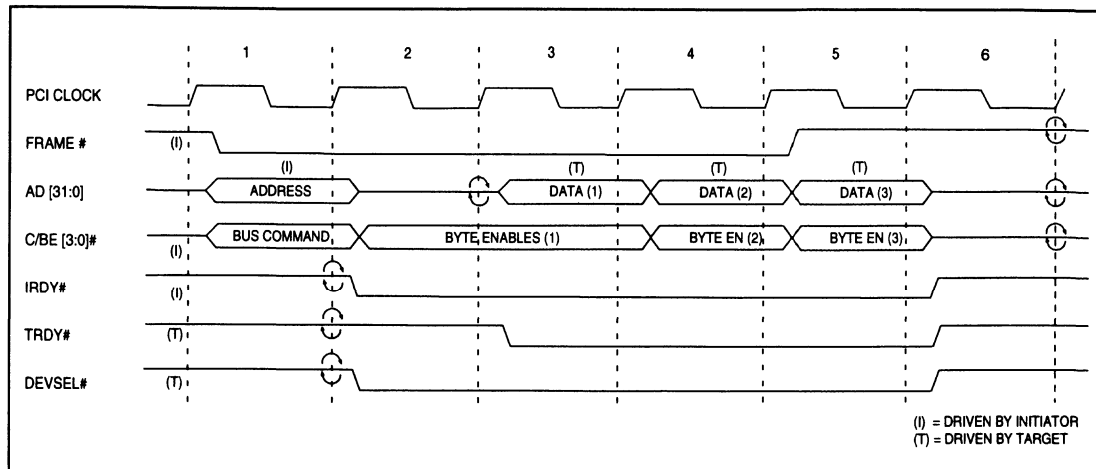
Read accesses from the S5933 operation registers (S5933 as a target) are shown in Figure 7-2. The S5933 conditionally asserts STOP# in clock period 3 if the initiator keeps FRAME# asserted during clock period 2 with IRDY# asserted (indicating a burst is being attempted). Wait states may be added by the initiator by not asserting the signal IRDY# during clock 3 and beyond. If FRAME# remains asserted, but IRDY# is not asserted, the initiator is just adding wait states, not necessarily attempting a burst.

There is only one condition where accesses to S5933 operation registers do not return TRDY# but do assert STOP#. This is called a target-initiated termina-

tion or target disconnect (described in more detail in Section 7.1.5) and occurs when a read attempt is made to an empty S5933 FIFO. The assertion of STOP# without the assertion of TRDY# indicates that the initiator should retry the operation later. This is discussed further in Section 7.1.5.2.

When burst read transfers are attempted to the S5933 operation registers, STOP# is asserted during the first data transfer to indicate to the initiator that no further transfers (data phases) are possible. This is a target-initiated termination where the target disconnects after the first data transfer. Figure 7-3 shows the signal relationships during a burst read attempt to the S5933 operation registers.

Figure 7-2. Single Data Phase PCI Bus Read of S5933 Registers (S5933 as Target)
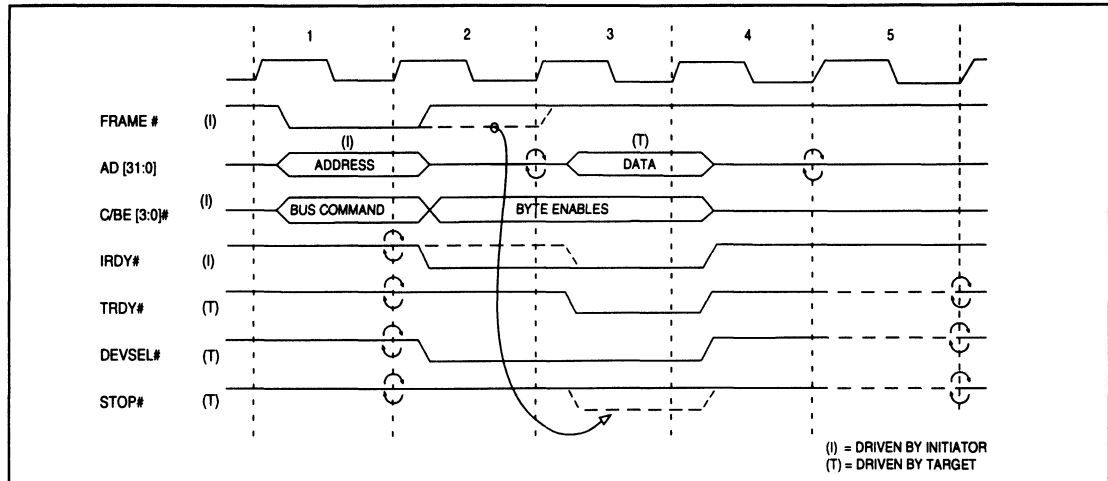
Figure 7-3. Burst PCI Bus Read Attempt to S5933 Registers (S5933 as Target)
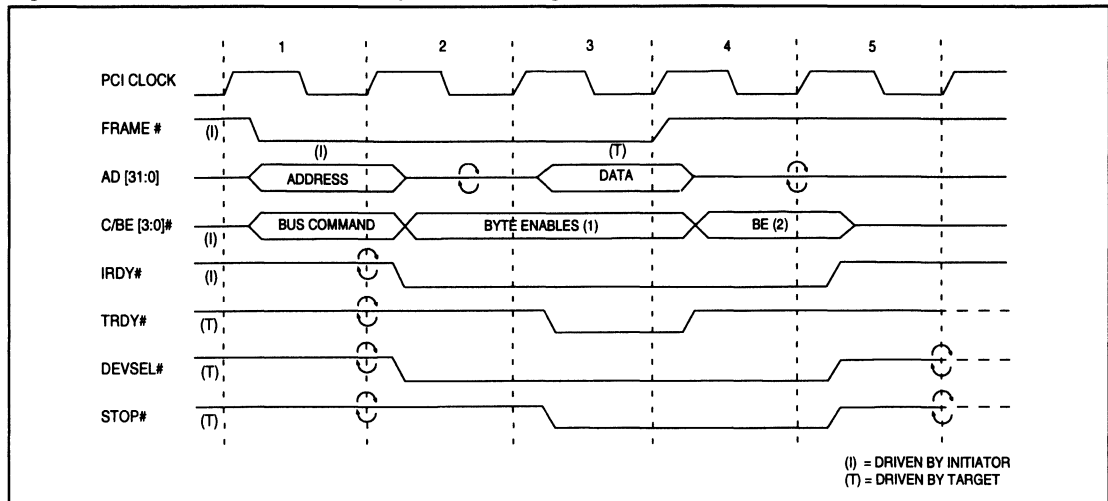
7.1.3 PCI Write Transfers

Write transfers on the PCI bus are one clock period shorter than read transfers. This is because the AD[31:0] bus does not require a turn-around cycle between the address and data phases. When the S5933 is accessed (target), it responds to a PCI bus memory or I/O transfers. As a PCI initiator, the S5933 controller can also execute PCI memory write operations.

The timing diagram in Figure 7-4 represents an S5933 initiator PCI write operation transferring to a fast, zero-wait-state memory target. The signals driven by the S5933 during the transfer are FRAME#, AD[31:0], C/BE[3:0]#, and IRDY#. The signals driven by the target are DEVSEL# and TRDY#. As with PCI reads, targets assert DEVSEL# and TRDY# after the clock defining the end of the address phase (boundary of clock periods 1 and 2 of Figure 7-4). TRDY# is not driven until the target has accepted the data for the PCI write. When the S5933 becomes the PCI initiator, it attempts sustained zero-wait state burst writes until one of the following occurs:

- The memory target aborts the transfer
- PCI bus grant (GNT# is removed)
- The add-on to PCI FIFO becomes empty
- A higher priority (PCI to add-on) S5933 transfer is pending (if programmed for priority)
- The write transfer byte count reaches zero
- Bus mastering is disabled from the add-on interface

Write accesses to the S5933 operation registers (S5933 as a target) are shown in Figure 7-5. Here, the S5933 asserts the signal STOP# in clock period 2. STOP# is asserted because the S5933 supports fast, zero-wait-state write cycles but does not support burst writes to operation registers. Wait states may be added by the initiator by not asserting the signal IRDY# during clock 2 and beyond. There is only one condition where writes to S5933 operation registers do not return TRDY# (but do assert STOP#). This is called a target-initiated termination or target disconnect (described in more detail in Section 7.1.5) and occurs when a write attempt is made to a full S5933 FIFO. As with the read transfers, the assertion of STOP# without the assertion of TRDY# indicates the initiator should retry the operation later. This is discussed further in Section 7.1.5.2.

Figure 7-4. Zero Wait State Burst Write PCI Bus Transfer (S5933 as Initiator)
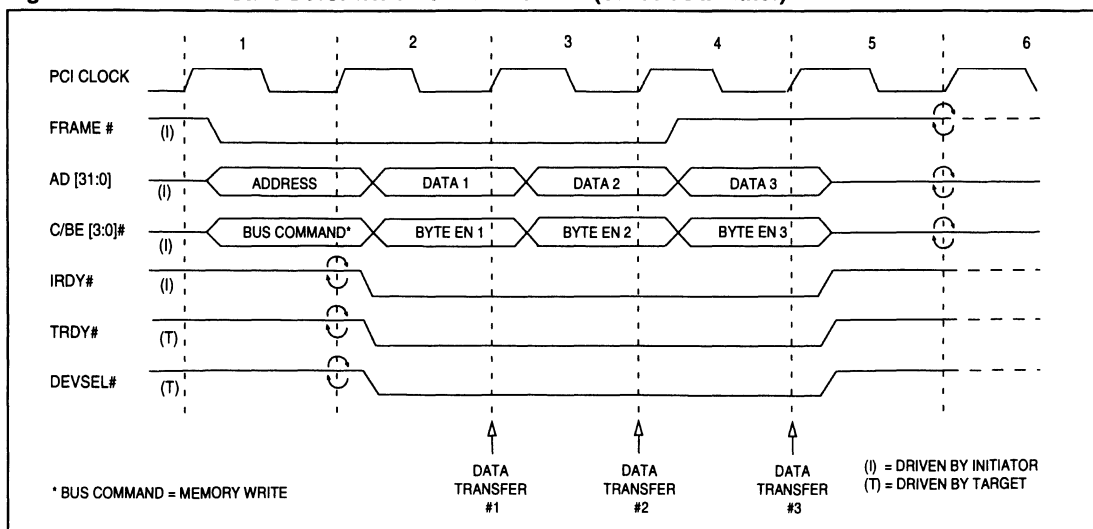
7.1.4 Master-Initiated Termination

Occasionally, a PCI transfer must be terminated by the initiator. Typically, the initiator terminates a transfer upon the successful completion of the transfer. These normal completions are described in Section 7.1.4.1. Sometimes, the initiator's bus mastership is relinquished by the bus arbiter (GNT# is removed), often because another device requires bus ownership. This is called initiator preemption and is discussed in Section 7.1.4.2. When the S5933 is an initiator and does not observe a DEVSEL# response to its assertion of FRAME#, it terminates the cycle (master abort), as described in Section 7.1.4.3.

7.1.4.1 Normal Cycle Completion

A successful data transfer occurs when both the initiator and target assert their respective ready signals, IRDY# and TRDY#. The last data phase is indicated by the initiator when FRAME# is deasserted during a data transfer. A normal cycle completion occurred if the target does not assert STOP#. Figure 7-6 shows the signal relationships defining a normal transfer completion.

Figure 7-5. Single Data Phase PCI Bus Write of S5933 Registers (S5933 as Target)
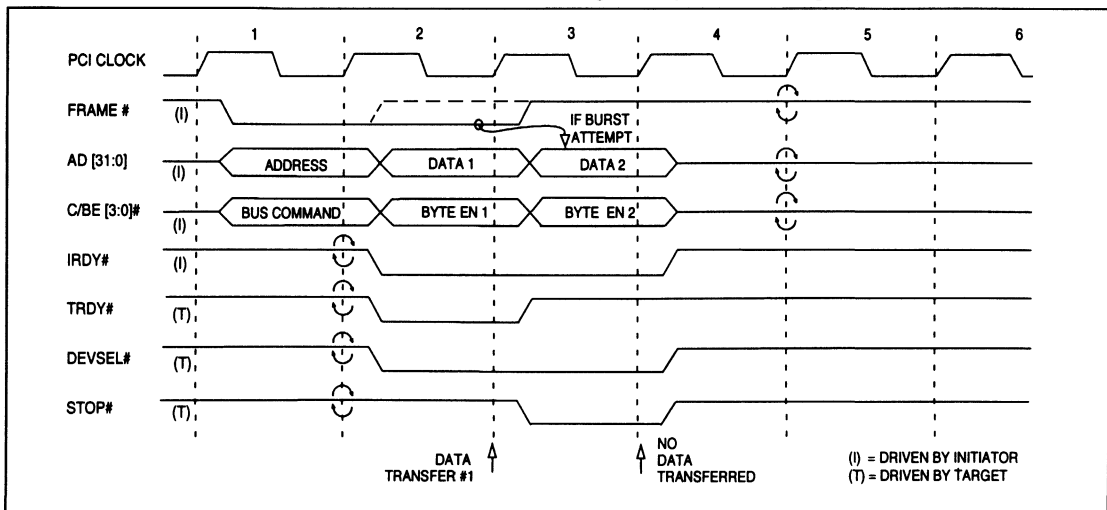
Figure 7-6. Master-Initiated, Normal Completion (S5933 as either Target or Initiator)
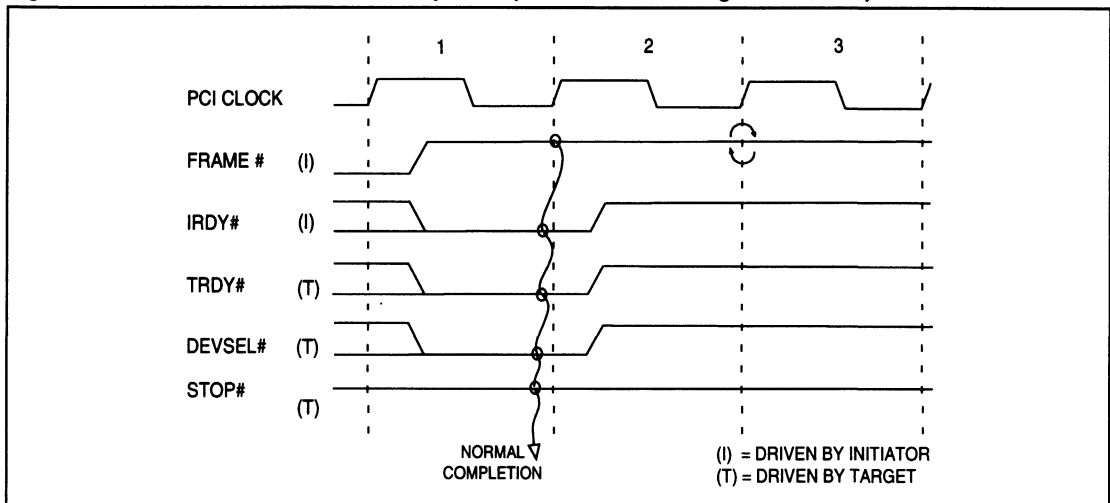
7.1.4.2 Initiator Preemption

A PCI initiator (bus master) is said to be preempted when the system platform deasserts the initiator's bus grant signal, GNT#, while it still requests the bus (REQ# asserted). This situation occurs if the initiator's latency timer expires and the system platform (bus arbitrator) has a bus master request from another device. The S5933 Master Latency Timer register is described in Section 3.8 and controls the S5933 responsiveness to the removal of a bus grant (preemption). The presence of a Master Latency Timer register can cause two preemption situations:

1) Removal of GNT# when the latency timer is non-zero (S5933 is guaranteed to still "own the bus").

2) Removal of the GNT# after the latency timer has expired.

The first situation is depicted in Figure 7-7, when the latency timer has not expired. Preemption with a zero or expired latency timer is shown in Figure 7-8.

Figure 7-7. Master Initiated Termination Due to Preemption and Latency Timer Active (S5933 as Master)
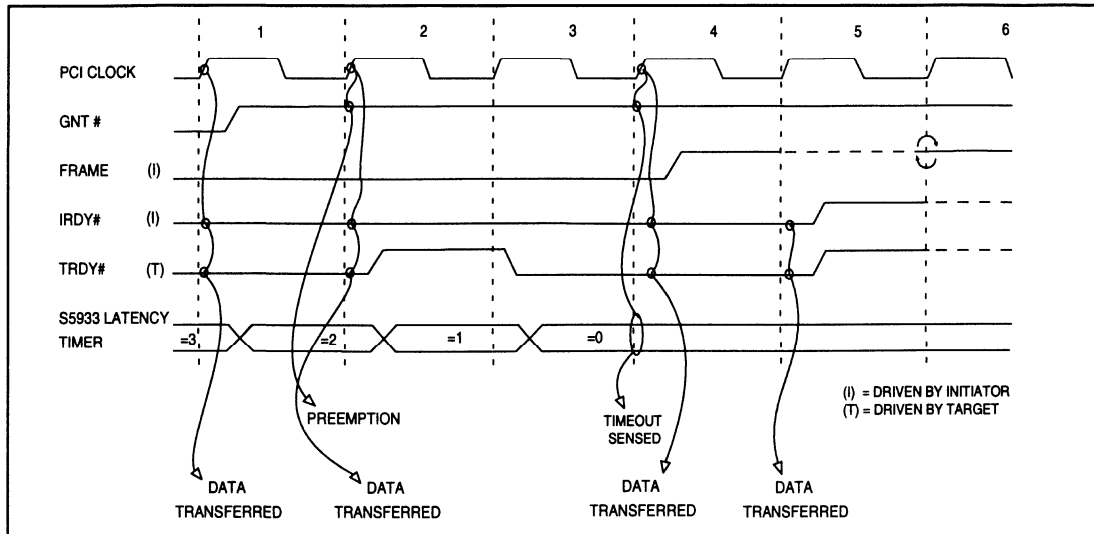
Figure 7-8. Master Initiated Termination Due to Preemption and Latency Timer Expired (S5933 as Master)
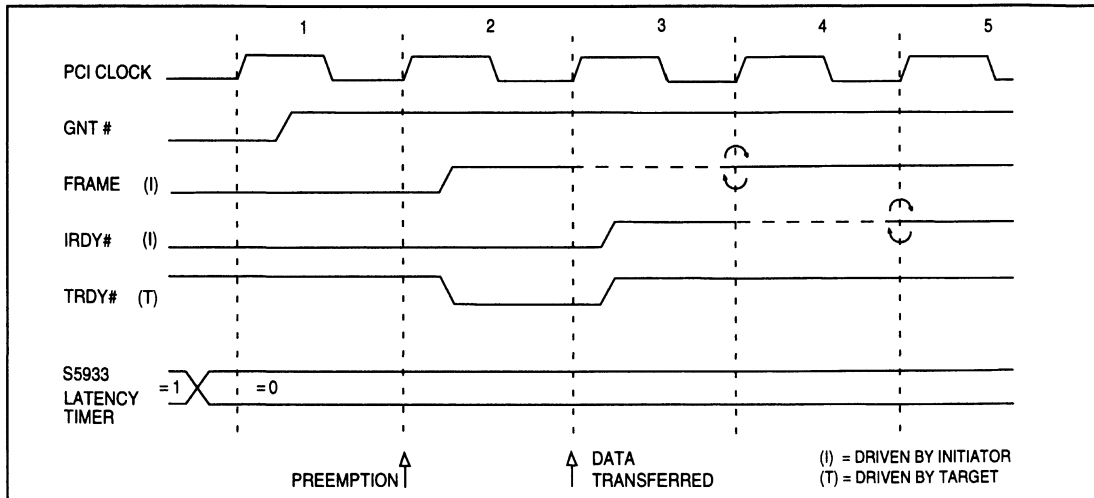
7.1.4.3 Master Abort

PCI accesses to nonexistent or disabled targets never observe DEVSEL# being asserted. In this situation, it is necessary for the initiator to abort the transaction (master abort). As an initiator, S5933 waits for six clock periods after asserting FRAME# to determine whether a master abort is required. These six clock periods allow slow targets, which may require several bus clocks before being able to assert DEVSEL#, to respond. It is also possible a PCI bridge device is present which employs "subtractive" decoding. A device which does a subtractive decode asserts DEVSEL#, claiming the cycle, when it sees that no other device has asserted it three clocks after the address phase.

If DEVSEL# is not asserted, the S5933 deasserts FRAME# (if asserted) upon the sixth clock period (Figure 7-9). IRDY# is deasserted by the S5933 during the next clock. The occurrence of a master abort causes the S5933 to set bit 13 (Master Abort) of the PCI Status Register (described in Section 3.4), indicating an error condition.

7.1.5 Target-Initiated Termination

There are situations where the target may end a transfer prematurely. This is called "target-initiated termination." Target terminations fall into three categories: disconnect, retry, and target abort. Only the disconnect termination completes a data transfer.

Figure 7-9. Master Abort, No Response

7

FAST DEVICE   MEDIUM DEVICE   SLOW DEVICE   BRIDGE DEVICE (SUBTRACTIVE DECODE)

(I) = DRIVEN BY INITIATOR
(T) = DRIVEN BY TARGET
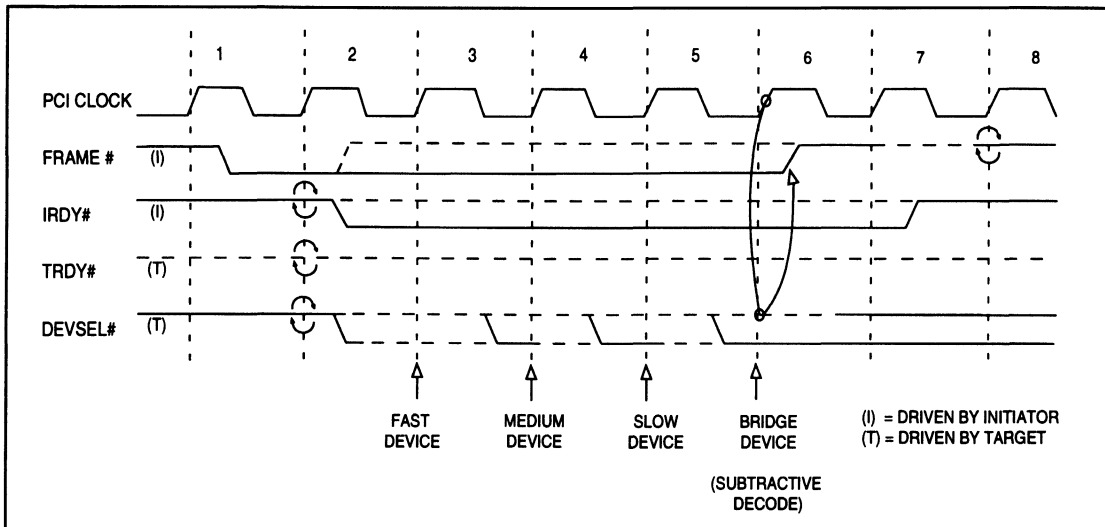
7.1.5.1 Target Disconnects

There are many situations where a target may disconnect. Slow responding targets may disconnect to permit more efficient (faster) devices to be accessed while they prepare for the next data phase, or a target may disconnect if it recognizes that the next data phase in a burst transfer is out of its address range. A target disconnects by asserting STOP#, TRDY#, and DEVSEL# as shown in Figures 7-10a and 7-10b. The initiator in Figure 7-10a responds to the disconnect condition by deasserting FRAME# on the follow-ing clock but does not complete the data transfer until IRDY# is asserted. This situation can only occur when the S5933 is a target. When the S5933 is an initiator, IRDY# is always asserted during the data phase (no initiator wait states). The timing diagram in Figure 7-10b applies to the S5933 as either a target disconnecting or an initiator with its target performing a disconnect. The S5933 performs a target disconnect if a burst access is attempted to the PCI Operation Registers.

Figure 7-10a. Target Disconnect Example 1 (IRDY# deasserted)
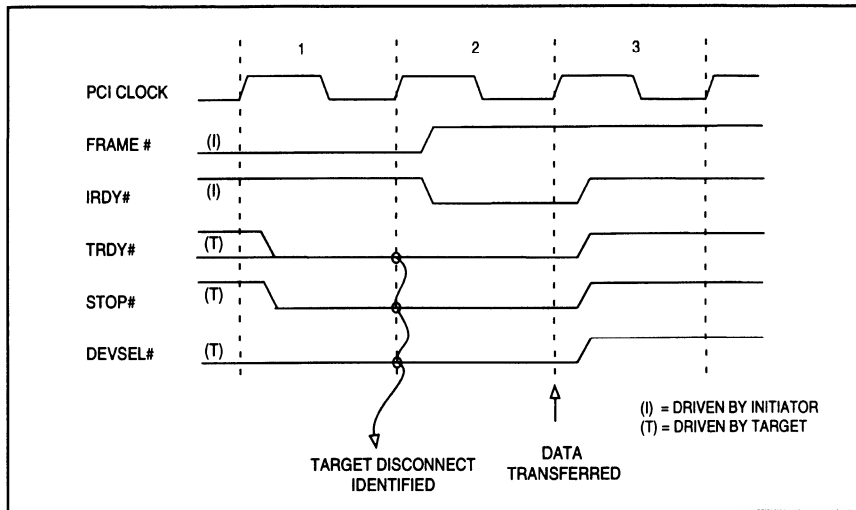
Figure 7-10b. Target Disconnect Example 2 (IRDY# asserted)
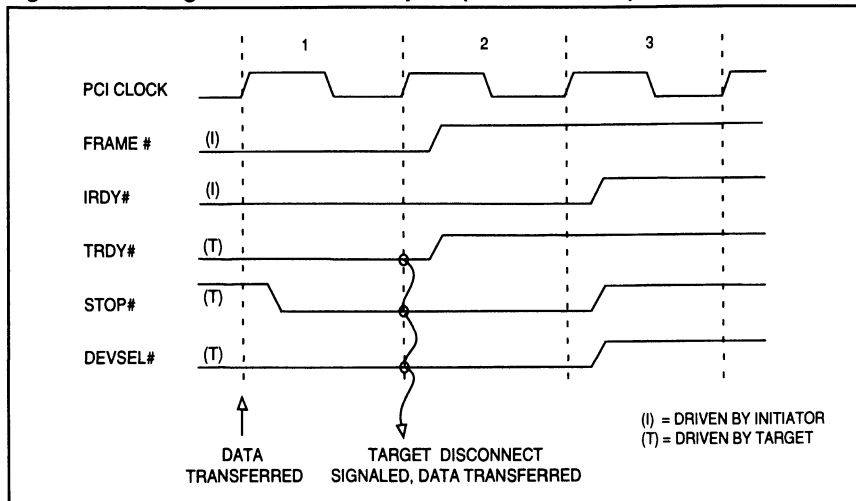
7.1.5.2 Target Requested Retries

When the S5933 FIFO registers are accessed (S5933 as a target) and data is unavailable (empty FIFO) for read transfers or cannot be accepted for write transfers (full FIFO), the S5933 immediately terminates the cycle by requesting a retry. The S5933 also initiates a retry for pass-thru writes where the add-on has not completed the preceding pass-thru write by asserting PTRDY#, and for pass-thru reads where the add-on cannot supply data within 8 PCI clocks (16 clocks for the first data phase of a burst). A retry is requested by a target asserting both STOP# and DEVSEL# while TRDY# is deasserted. Figure 7-11 shows the behavior of the S5933 when performing a target-initiated retry.

7.1.5.3 Target Aborts

A target abort termination represents an error condition where no number of retries will produce a successful target access. A target abort is uniquely identified by the target deasserting DEVSEL# and TRDY# while STOP# is asserted. When a target performs an abort, it must also set bit 11 of its PCI Status register (Section 3.4). The S5933 configuration and operation registers never respond with a target abort when accessed. If the S5933 encounters this condition when operating as a PCI initiator, the S5933 sets bit 12 (received target abort) in the PCI Status register. Figure 7-12 depicts a target abort cycle.

Target termination types are summarized in Table 7-2.

Figure 7-11. Target-Initiated Retry
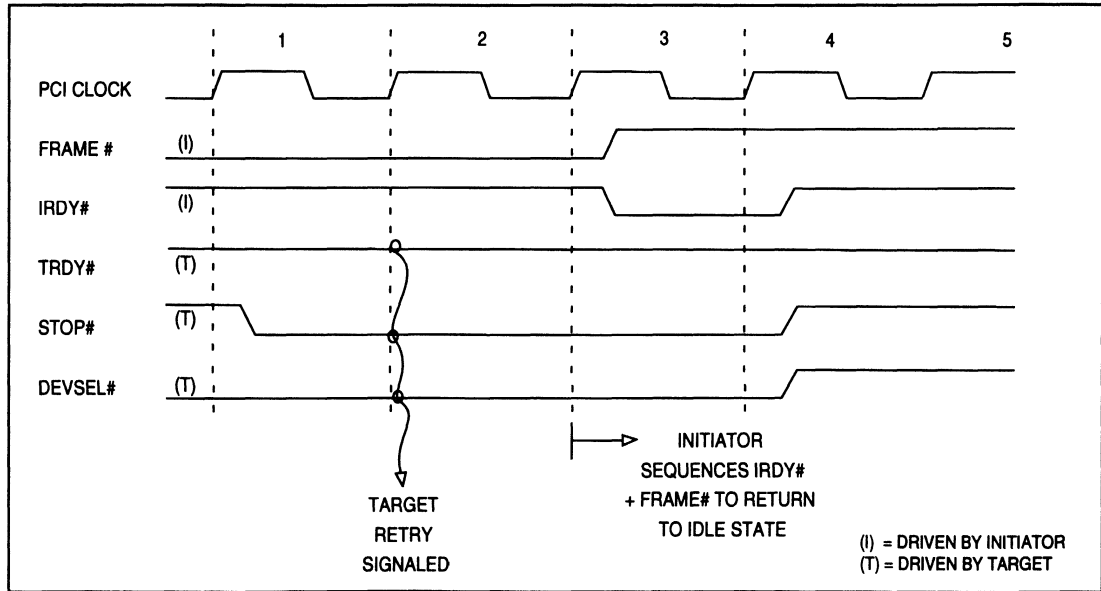
Table 7-2. Target Termination Types

Termination	DEVSEL#	STOP#	TRDY#	Comment
Disconnect	on	on	on	Data is transferred. Transaction needs to be re-initiated to complete.
Retry	on	on	off	Data was not transferred. Transaction should be tried later.
Abort	off	on	off	Data was not transferred. Fatal error.

Figure 7-12. Target Abort Example

TARGET ABORT
IDENTIFIED

(I) = DRIVEN BY INITIATOR
(T) = DRIVEN BY TARGET
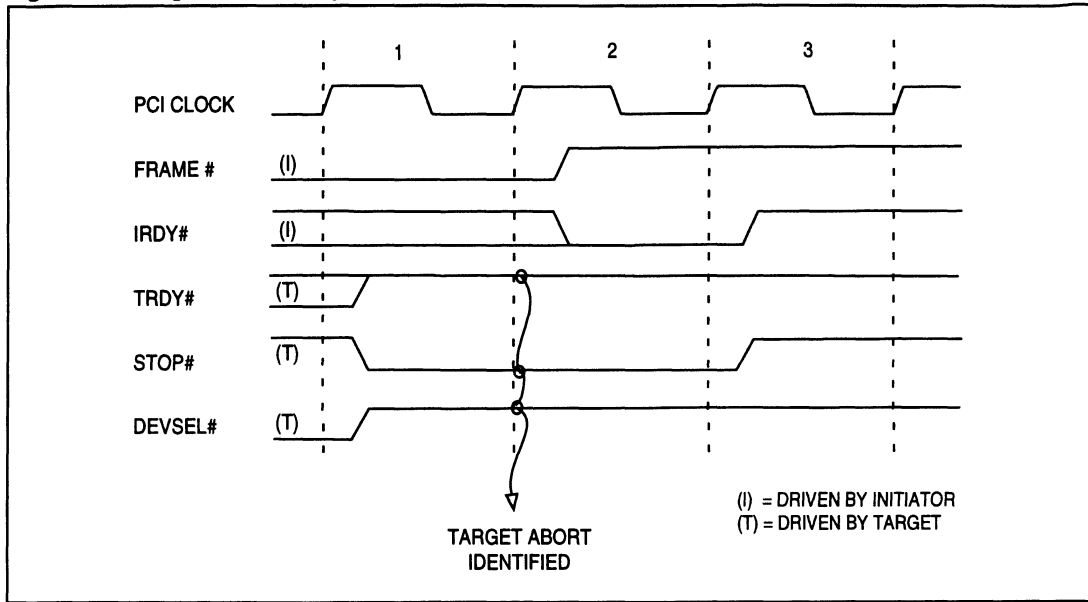
Figure 7-13. PCI Bus Arbitration and S5933 Bus Ownership Example

| IDLE | S5933 TRANSACTION | IDLE (TURN-AROUND) | "OTHER", PREEMPTING MASTER TRANSACTION | IDLE (TURN-AROUND) | S5933 TRANSACTION(S) |

(I) = DRIVEN BY INITIATOR
(T) = DRIVEN BY TARGET
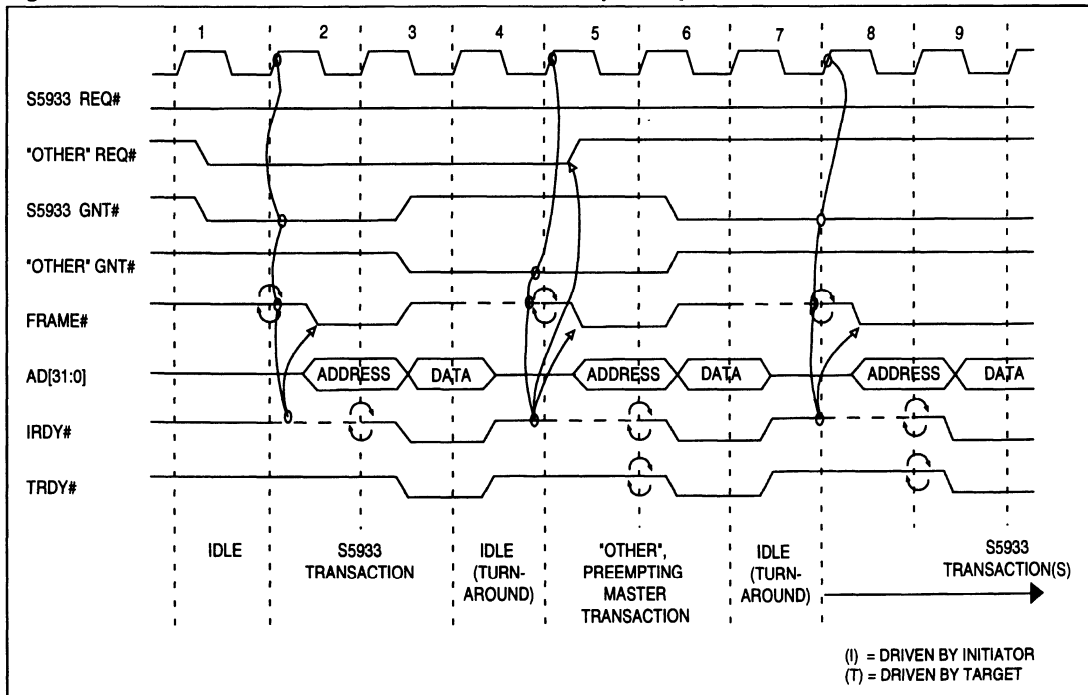
7.2 PCI BUS MASTERSHIP

When the S5933 requires PCI bus mastership, it presents a request via the REQ# signal. This signal is connected to the system's PCI bus arbiter.

Only one initiator (bus master) may control the PCI bus at a given time. The bus arbiter determines which initiator is given control of the bus. Control is granted to a requesting device by the arbiter asserting that device's grant signal (GNT#). Each REQ#/GNT# signal pair is unique to a given PCI agent.

After asserting REQ#, the S5933 assumes bus ownership on the first PCI clock edge where its GNT# input is asserted along with FRAME# and IRDY# deasserted (indicating no other device is generating PCI bus cycles). Once ownership is established by the S5933, it maintains ownership as long as the arbiter keeps its GNT# asserted. If GNT# is deasserted, the S5933 completes the current transaction. The S5933 does this by deasserting FRAME# and then deasserting IRDY# upon data transfer. Figure 7-13 shows a sequence where the S5933 is granted ownership of the bus and then is preempted by another master before the S5933 can finish its current transaction.

7.2.1 Bus Mastership Latency Components

It is often necessary for system designers to predict and guarantee that a minimum data transfer rate is sustainable to support a particular application. In the design of a bus mastering application, knowledge of the maximum delay a device might encounter from the time it requests the PCI bus to the time in which it is actually granted the bus is desirable. This allows the design to provide adequate data buffering. The PCI specification refers to this bus request to grant delay as "arbitration latency."

Once a PCI initiator has been granted the bus, the PCI specification defines the delay from the grant to the new initiator's assertion of FRAME# as the "bus acquisition latency." Afterwards, the delay from FRAME# asserted to target ready (TRDY#) asserted is defined as "target latency." Figure 7-14 shows a time-line depicting the components of PCI bus access latency.

There are numerous configuration variations possible with the PCI specification. A system designer can determine whether a bus master can support a critical, timely transfer by establishing a specific configuration and by defining these latency values. The S5933, as an initiator, produces the fastest response allowable for its bus acquisition latency (GNT# to FRAME# asserted). The S5933 also implements the PCI Master Latency Timer. Once granted the bus, the S5933 is guaranteed ownership for a minimum amount of time defined by the Master Latency Timer. The S5933, as an initiator, cannot control the responsiveness of a particular target nor the bus arbitration delay.

The PCI specification provides two mechanisms to control the amount of time a master may own the bus. One mechanism is through the master and is covered in Section 7.1.4 (master-initiated termination). The other is by the target and is achieved through a target-initiated disconnect.

7.2.1.1 Bus Arbitration

Although the PCI specification defines the condition that constitutes bus ownership, it does not provide rules to be used by the system's PCI bus arbiter in deciding which master is to be granted the PCI bus next. The arbitration priority scheme implemented by a system may be fixed, rotational, or custom. The arbitration latency is a function of the system, not the S5933.

Figure 7-14. PCI Bus Access Latency Components
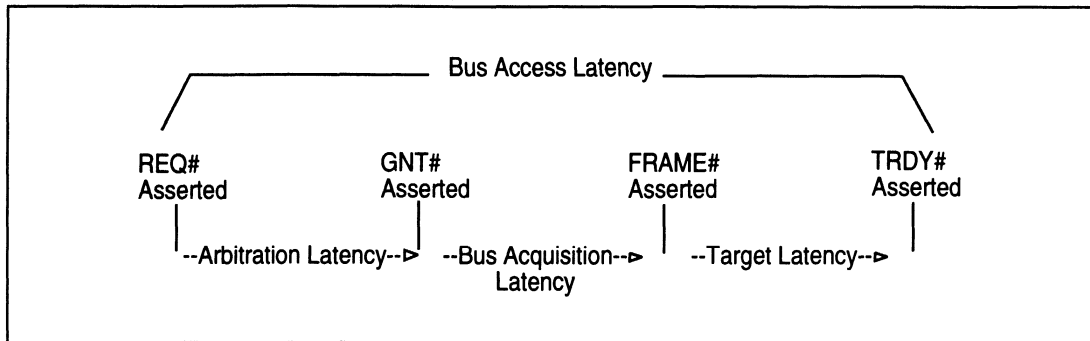
7.2.1.2 Bus Acquisition

Once GNT# is asserted, giving bus ownership to the S5933, the S5933 must wait until the PCI bus becomes idle. This delay is called bus acquisition latency and involves the state of the signals FRAME# and IRDY#. The current bus master must complete its current transaction before the S5933 may drive the bus. Table 7-3 depicts the four possible combinations of FRAME# and IRDY# with their interpretation.

7.2.1.3 Target Latency

The PCI specification requires that a selected target relinquish the bus should an access to that target require more than eight PCI clock periods (16 clocks for the first data phase in a burst). Slow targets can exist within the PCI specification by using the target initiated retry described in Section 7.1.5.2. This prevents slow target devices from potentially monopolizing the PCI bus and also allows more accurate estimations for bus access latency.

7.2.2 Target Locking

It is possible for a PCI bus master to obtain exclusive access to a target ("locking") through use of the PCI bus signal LOCK#. LOCK# is different from the other PCI bus signals because its ownership may belong to any bus master, even if it does not currently have ownership of the PCI bus. The ownership of LOCK#, if not already claimed by another master, may be achieved by the current PCI bus master on the clock period following the initial assertion of FRAME#. Figure 7-15 describes the signal relationship for establishing a lock. The ownership of LOCK#, once established, persists even while other bus masters control the bus. Ownership can only be relinquished by the master which originally established the lock.

Figure 7-15. Engaging the LOCK# Signal
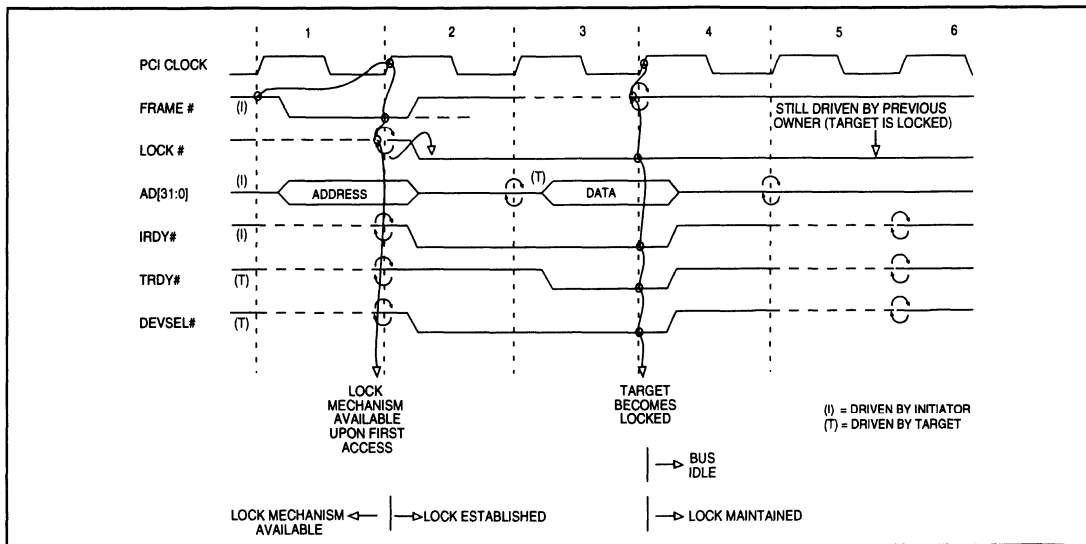
Table 7-3. Possible Combinations of FRAME# and IRDY#

FRAME#	IRDY#	Description
deasserted	deasserted	Bus Idle
deasserted	asserted	The initiator is ready to complete the last data transfer of a transaction.
asserted	deasserted	An Initiator has a transaction in progress but is not able to complete the data transfer on this clock.
asserted	asserted	An initiator has a transaction in progress and is able to complete a data transfer.

Targets selected with LOCK# deasserted during the assertion of FRAME# (clock period 1 of Figure 7-15), which encounter the assertion of LOCK# during the following clock (clock period 2 of Figure 7-15) are thereafter considered "locked." A target, once locked, requires that subsequent accesses to it deassert LOCK# while FRAME# is asserted. Figure 7-16 show a subsequent access to a locked target by the master which locked it. Because LOCK# is owned by a single master, only that master is able to deassert it at the beginning of a transaction (allowing successful access to the locked target). A locked target can only be unlocked during the clock period following the last data transfer of a transaction when the LOCK# signal is deasserted.

An unlocked target ignores LOCK# when it observes that LOCK# is already asserted during the first clock period of a transaction. This allows other masters to access other (unlocked) targets. If an access to a locked target is attempted by a master other than the one that locked it, the target responds with a retry request, as shown in Figure 7-17.

The S5933 responds to and supports bus masters which lock it as a target. When the S5933 is a bus master, it never attempts to lock a target, but it honors a target's request for retry if that target is locked by another master.

Figure 7-16. Access to a Locked Target by its Owner
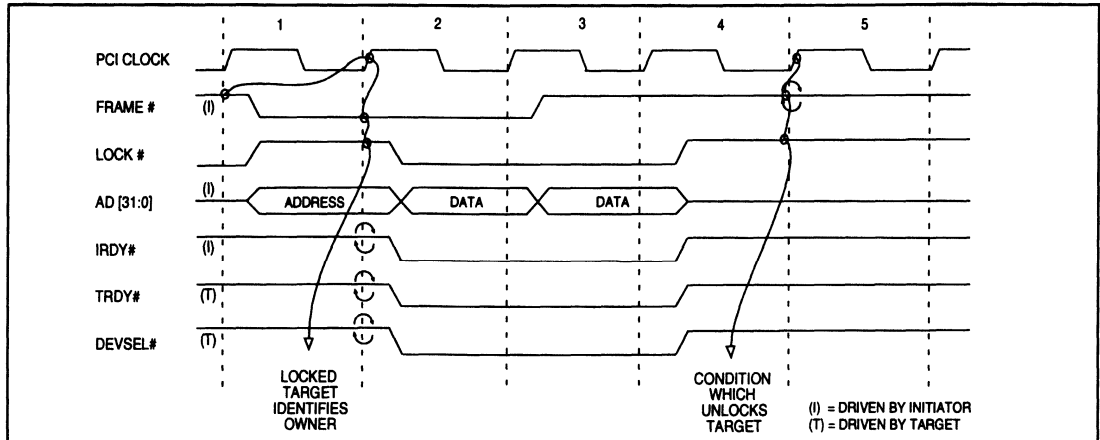
Figure 7-17. Access Attempt to a Locked Target
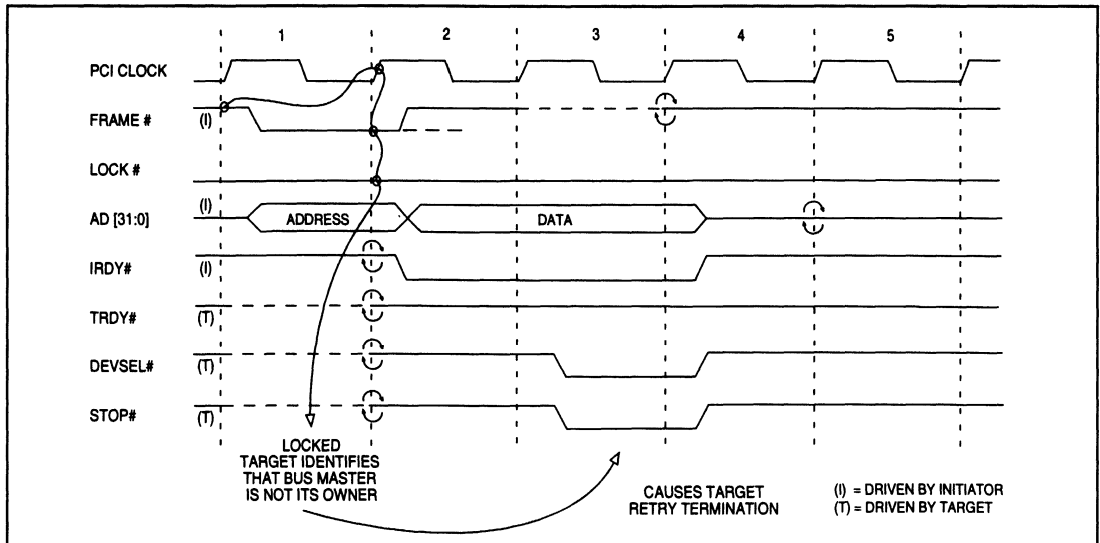
7.3 PCI BUS INTERRUPTS

The S5933 controller is able to generate PCI bus interrupts by asserting the PCI bus interrupt signal (INTA#). INTA# is a multisourced, wire-ORed signal on the PCI bus and is driven by an open drain output on the S5933. Section 2.1.6 describes the physical signal pin and Section 4.9 describes the interrupt's control and status register. The assertion and deassertion of INTA# have no fixed timing relationship with respect to the PCI bus clock. Once the S5933 asserts INTA#, it remains asserted until the interrupt source is cleared by a write to the Interrupt Control/Status Register (INTCSR) (see Section 4.9).

7.4 PCI BUS PARITY ERRORS

The PCI specification defines two error-reporting signals, PERR# and SERR#. These signals indicate a parity error condition on the signals AD[31:0], C/BE[3:0]#, and PAR. The validity of the PAR signal is delayed one clock period from its corresponding AD[31:0] and C/BE[3:0]# signals. Even parity exists when the total number of ones in the group of signals is equal to an even number. PERR# is the error-reporting mechanism for parity errors that occur during the data phase for all but PCI Special Cycle commands. SERR# is the error-reporting mechanism for parity errors that occur during the address phase.

The timing diagram in Figure 7-18 shows the timing relationships between the signals AD[31:0], C/BE[3:0]#, PAR, PERR# and SERR#.

The S5933 asserts SERR# if it detects odd parity during an address phase, if enabled. The SERR# enable bit is bit 8 in the S5933 PCI Command Register (Section 3.3). The odd parity error condition involves the state of signals AD[31:0] and C/BE[3:0]# when FRAME# is first asserted and the PAR signal during the following clock. If an error is detected, the

S5933 asserts SERR# on the following (after PAR valid) clock. Since many targets may observe an error on an address phase, the SERR# signal is an open drain multisourced, wire-ORed signal on the PCI bus. The S5933 drives SERR# low for one clock period when an address phase error is detected. Once an SERR error is detected by the S5933, the PCI Status register bit 14, System Error (Section 3.4), is set and remains until cleared through software or a hardware reset.

The PERR# signal is similar to the SERR# with two differences: it reports errors for the data phase and is only asserted by the device receiving the data. The S5933 drives this signal (removed from tri-state) when it is the selected target for write transactions or when it is the current master for bus read transactions. The parity error conditions are only reflected by the PERR# pin if the Parity Error Enable bit (bit 6) of the PCI Command register is set. Upon the detection of a data parity error, the Detected Parity Error bit (bit 15) of the PCI Status Register is set (Section 3.4). Unlike the PERR# signal pin, this Status bit sets regardless of the state of the PCI Command register Parity Error Enable bit. An additional status bit (bit 8) called "Data Parity Reported" of the PCI Status register is employed to report parity errors that occur when the S5933 is the bus master. The "Data Parity Error Reported" status requires that the Parity Error Enable bit be set in the PCI Command register.

The assertion of PERR# occurs two clock periods following the data transfer. This two-clock delay occurs because the PAR signal does not become valid until the clock following the transfer, and an additional clock is provided to generate and assert PERR# once an error is detected. PERR# is only asserted for one clock cycle for each error sensed. The S5933 only qualifies the parity error detection during the actual data transfer portion of a data phase (when both IRDY# and TRDY# are asserted).

Figure 7-18. Error Reporting Signals

Add-on Bus Interface　**8**
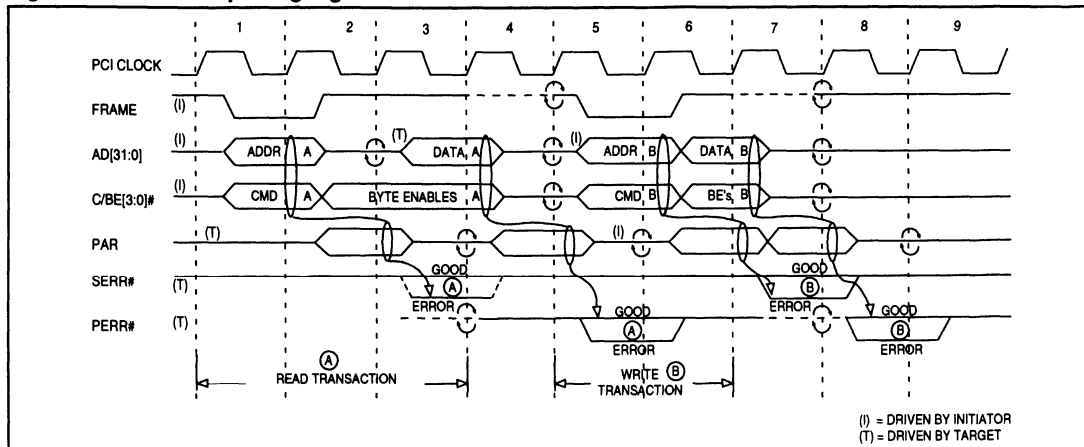
8.0   ADD-ON BUS INTERFACE

This chapter describes the add-on bus interface for the S5933. The S5933 is designed to support connection to a variety of microprocessor buses and/or peripheral devices. The add-on interface controls S5933 operation through the Add-on Operation Registers. These registers act as the pass-thru, FIFO, non-volatile memory and mailbox interfaces as well as offering control and status information.

Depending on the register being accessed, the interface may be synchronous, asynchronous, or configurable. To enhance performance and simplify add-on logic design, some registers allow direct access with a single device input pin. The following sections describe the various interfaces to the PCI bus and how they are accessed from the add-on interface.

8.1   ADD-ON OPERATION REGISTER ACCESSES

The S5933 add-on bus interface is very similar to that of a memory or peripheral device found in a microprocessor-based system. A 32-bit data bus with individual read and write strobes, a chip enable and byte enables are provided. Other add-on interface signals are provided to simplify add-on logic design.

Accesses to the S5933 registers are done synchronous or asynchronous to BPCLK. For S5933 functions that are compatible with an add-on microprocessor interface, it is helpful to allow an asynchronous interface, as the processor may not operate at the PCI bus clock frequency.

8.1.1   Add-on Interface Signals

The add-on interface provides a small number of system signals to allow the add-on to monitor PCI bus activity, indicate status conditions (interrupts), and allow add-on bus configuration. A standard bus interface is provided for Add-on Operation Register accesses.

8.1.1.1   System Signals

BPCLK and SYSRST# allow the add-on interface to monitor the PCI bus status. BPCLK is a buffered version of the PCI clock. The PCI clock can operate from 0 MHz to 33 MHz. SYSRST# is a buffered version of the PCI reset signal, and may also be toggled by host application software through bit 24 of the Bus Master Control/Status Register (MCSR).

IRQ# is the add-on interrupt output. This signal is active low and can indicate a number of conditions. Add-on interrupts may be generated from the mailbox or FIFO interfaces. The exact conditions which generate an interrupt are discussed in the mailbox and FIFO chapters. The interrupt output is deasserted when acknowledged by an access to the Add-on Interrupt Control/Status Register (AINT). All interrupt sources are cleared by writing a one to the corresponding interrupt bit.

The MODE input on the add-on interface configures the datapath width for the add-on interface. MODE low indicates a 32-bit data bus. MODE high indicates a 16-bit data bus. For 16-bit operation, BE3# is redefined as ADR1, providing an extra address input. ADR1 selects the low or high words of the 32-bit S5933 Add-on Operation Registers.

8.1.1.2   Register Access Signals

Simple register accesses to the S5933 Add-on Operation Registers take two forms: synchronous to BPCLK and asynchronous. The following signals are required to complete a register access to the S5933.

BE[3:0]# Byte Enable Inputs. These S5933 inputs identify valid byte lanes during add-on transactions. When MODE is set for 16-bit operation, BE2# is not defined and BE3# becomes ADR1.

ADR[6:2] Address Inputs. These address pins identify the specific Add-on Operation Register being accessed. When configured for 16-bit operation (MODE=1), an additional input, ADR1 is available to allow the 32-bit operation registers to be accessed with two 16-bit cycles.

RD# Read Strobe Input.

WR# Write Strobe Input.

SELECT# Chip Select Input. This input identifies a valid S5933 access.

DQ[31:0] Bidirectional Data Bus. These I/O pins are the S5933 data bus. When configured for 16-bit operation, only DQ[15:0] are valid.

In addition, there are dedicated signals for FIFO accesses (RDFIFO# and WRFIFO#) and pass-thru address accesses (PTADR#). These are discussed separately in the FIFO and pass-thru sections of this chapter.

8

The internal interfaces of the S5933 allow Add-on Operation Registers to be accessed asynchronous to BPCLK (synchronous to the rising edge of the read or write strobe). The exception to this is the Add-on General Control/Status Register, as described in Section 8.1.4. For pass-thru operations, the Pass-thru Data Register accesses are synchronous to BPCLK to support burst transfers. The FIFO port may also be accessed synchronous to BPCLK, if configured to do so (see Section 10.3.1).

8.1.2 Asynchronous Register Accesses

For many add-on applications, add-on logic does not operate at the PCI bus frequency. This is especially true for add-ons implementing a microprocessor, which may be operating at a lower (or higher) frequency. Figures 8-1 and 8-2 show asynchronous Add-on Operation Register accesses. Exact AC timings are detailed in the Electrical and AC Characteristics chapter (Chapter 12).

For asynchronous reads (Figure 8-1), data is driven on the data bus when RD# is asserted. When RD# is not asserted, the DQ[31:0] outputs float. A valid address and valid byte enables must be presented before correct data is driven. RD# has both a minimum inactive time and a minimum active time for asynchronous accesses.

For asynchronous writes (Figure 8-2), data is clocked into the S5933 on the rising edge of the WR# input. Address, byte enables, and data must all meet setup and hold times relative to the rising edge or WR#. WR# has both a minimum inactive time and a minimum active time for asynchronous accesses

8.1.3 Synchronous FIFO and Pass-thru Data Register Accesses

To obtain the highest data transfer rates possible, add-on logic should operate synchronously with the PCI clock. The buffered PCI clock (BPCLK) is provided for this purpose. A synchronous interface with pass-thru mode or the FIFO allows data to be transferred at the maximum PCI bus bandwidth (132 MBytes/sec) by allowing burst accesses with the add-on interface. The RD# and WR# inputs become enables, using BPCLK to clock data into and out of registers. This section applies only to synchronous accesses to the FIFO (AFIFO) and Pass-thru Data (APTD) registers.

Figures 8-3 and 8-4 show single-cycle, synchronous FIFO and pass-thru Operation Register accesses. Exact AC timings are detailed in the Electrical and AC Characteristics chapter (Chapter 12).

For synchronous reads (Figure 8-3), data is driven onto the data bus when RD# (or RDFIFO#) is asserted. When RD# is not asserted, the DQ[31:0] outputs float. The address, byte enable, and RD# inputs must meet setup and hold times relative to the rising edge of BPCLK. Burst reads may be performed by holding RD# low.

For synchronous writes (Figure 8-4), data is clocked into the register on the rising edge of BPCLK. Address, byte enables, and data must all meet setup and hold times relative to the rising edge or BPCLK. Burst writes may be performed by holding WR# (or WRFIFO#) low. When holding WR# low, data is clocked in on each BPCLK rising edge.

8.1.4 nv Memory Accesses Through the Add-on General Control/Status Register

All Add-on Operation Registers may be accessed as described in Sections 8.1.2 and 8.1.3 (although only the FIFO and Pass-thru Data registers support bursting). To access nv memory contents through the Add-on General Control/Status Register (AGCSTS), special considerations must be made. Internally, all nv memory accesses by the S5933 are synchronized to a divided-down version of the PCI bus clock. Because of this, if nv memory accesses are performed through the AGCSTS register, the register access must be synchronized to BPCLK. The rising edge RD# or WR# is still used to clock data, but these inputs along with the address and byte enables are synchronized to BPCLK. Accesses to AGCSTS for monitoring FIFO or mailbox status, etc., may be done asynchronous to BPCLK.

8.2   MAILBOX BUS INTERFACE

The mailbox registers are accessed asynchronously using the method described in Section 8.1.2. The mailbox register names may need some clarification. For the add-on interface, an outgoing mailbox refers to a mailbox sending information to the PCI bus. An incoming mailbox refers to a mailbox receiving information from the PCI bus. An outgoing mailbox on the add-on interface is, internally, the same as the corresponding incoming mailbox on the PCI interface and vice-versa. Chapter 9 provides detailed information on the S5933 mailbox functionality.

Figure 8-1. Asynchronous Add-on Operation Register Read

BE[3:0]#

ADR[6:2]

DQ[31:0]

SELECT#

RD#

Valid Byte Enables

Valid Address

Valid Data Out

8
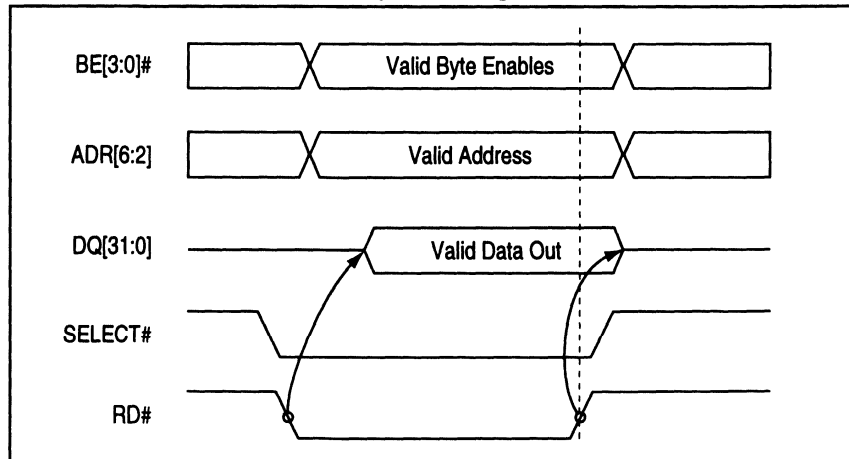
Figure 8-2. Asynchronous Add-on Operation Register Write

BE[3:0]#

ADR[6:2]

DQ[31:0]

SELECT#

WR#

Valid Byte Enables

Valid Address

Valid Data In
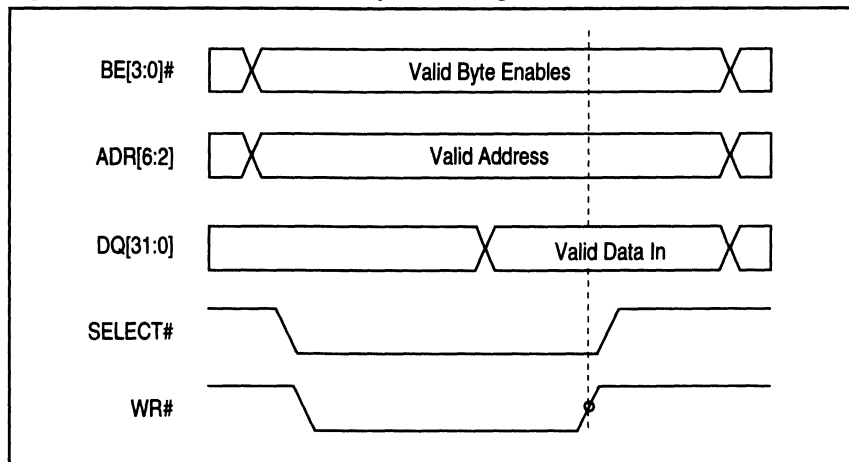
Figure 8-3. Synchronous FIFO or Pass-thru Data Register Read
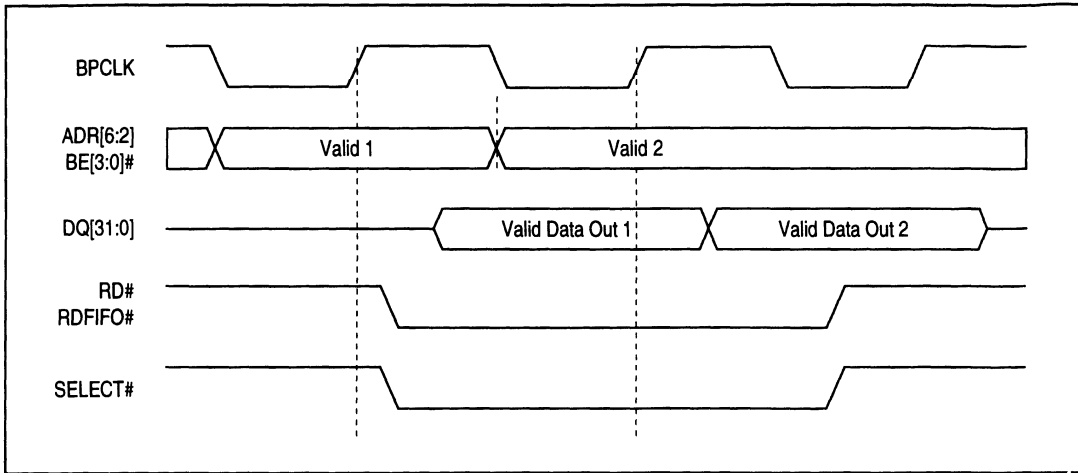
Figure 8-4. Synchronous FIFO or Pass-thru Data Register Write
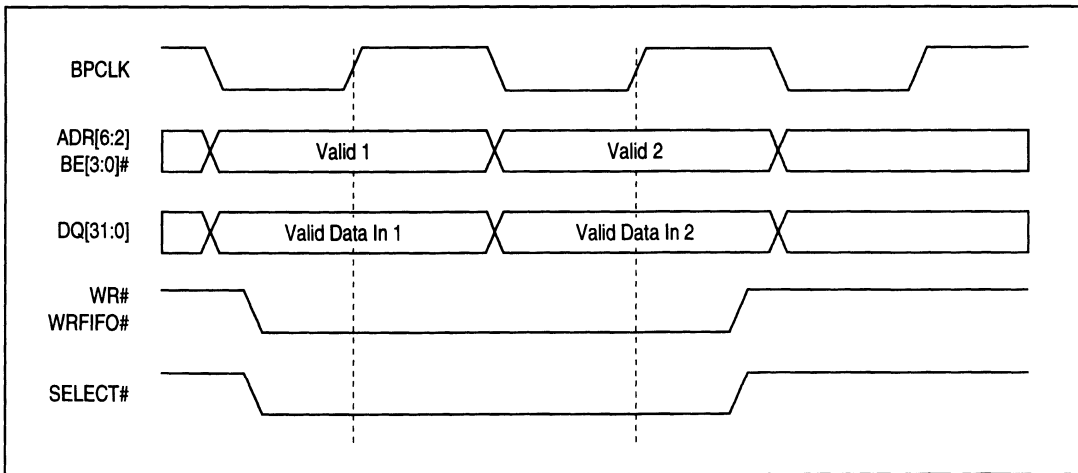
8.2.1 Mailbox Interrupts

Mailboxes can be configured to generate add-on interrupts (IRQ#) and/or allow the add-on to generate PCI interrupts (INTA#). Mailbox empty/full status conditions be can used to interrupt the add-on or PCI host to indicate some action is required. An individual mailbox byte is selected to generate an interrupt when accessed. An outgoing mailbox becoming empty or an incoming mailbox becoming full asserts the interrupt output (if enabled).

When used with a serial nv memory boot device, the mailboxes also provide a way to generate PCI interrupts (INTA#) through hardware. When a serial nv memory boot device is used, the device pin functions EA0 - EA8 are redefined. These pins then provide direct, external access to the add-on outgoing mailbox 4, byte 3 (which is also PCI incoming mailbox 4, byte 3). See Section 9.1.3 for details.

8.3 FIFO BUS INTERFACE

The FIFO register on the add-on interface may be accessed synchronously using the method described in section 8.1.2 or asynchronously using the method described in Section 8.1.3. Location 45h, bits 6 and 5 in the nv memory boot device determine the interface method (see Section 10.3.1). If no boot device is used, the default condition is an asynchronous FIFO interface.

8.3.1 FIFO Direct Access Inputs

RDFIFO# and WRFIFO# are referred to as FIFO 'direct access' inputs. Asserting RDFIFO# is functionally identical to accessing the FIFO with RD#, SELECT#, BE[3:0]#, and ADR[6:2]. Asserting WRFIFO# is functionally identical to accessing the FIFO with WR#, SELECT#, BE[3:0]#, and ADR[6:2]. RD# and WR# must be deasserted when RDFIFO# or WRFIFO# is asserted, but SELECT# may be asserted. These inputs automatically drive the address (internally) to 20h and assert all byte enables. The ADR[6:2] and BE[3:0]# inputs are ignored when using the FIFO direct access inputs. RDFIFO# and WRFIFO# are useful for add-on designs which cascade an external FIFO into the S5933 FIFO or use dedicated external logic to access the FIFO.

Direct access signals always access the FIFO as 16-bits or 32-bits, whatever the MODE pin is configured for. For 16-bit mode, two consecutive accesses fill or empty the 32-bit FIFO register (see Section 10.2.3.5).

8.3.2 FIFO Status Signals

The FIFO Status signals (Section 10.1.2) indicate to the add-on logic the current state of the S5933 FIFO. A FIFO status change caused by a PCI FIFO access is reflected one PCI clock period after the PCI access is completed (TRDY# asserted). A FIFO status change caused by an add-on FIFO access is reflected immediately (after a short propagation delay) after the access occurs. For add-on accesses, FIFO status is updated after the rising edge of BPCLK for synchronous interfaces or after the rising edge of the read or write strobe for asynchronous interfaces (See chapter 12 for exact timings).

8.3.3 FIFO Control Signals

For add-on initiated PCI bus mastering, the FIFO status reset controls FWC# (add-on to PCI FIFO clear) and FRC# (PCI to add-on FIFO clear) are available. FWC# and FRC# must be asserted for a minimum of one BPCLK period to be recognized. These inputs are sampled at the rising edge of BPCLK. These inputs should not be asserted unless the FIFO is idle. Asserting a FIFO status reset input during a PCI or add-on FIFO access results in indeterminate operation.

For add-on initiated bus master transfers (Section 10.2.3.3), AMREN (add-on bus master read enable) and AMWEN (add-on bus master write enable) are used, in conjunction with the appropriate FIFO status signals, to enable the S5933 to assert its PCI bus request (REQ#).

8.4 PASS-THRU BUS INTERFACE

The S5933 pass-thru interface is synchronous. The Add-on Pass-thru Address (APTA) and Add-on Pass-thru Data (APTD) registers may be accessed pseudo-synchronously using the method described in Section 8.1.3.

Although BPCLK is used to clock data into and out of the pass-thru registers, accesses may be performed asynchronously. For reads, APTA or APTD data remains valid as long as RD# (or PTADR#) is asserted. A new value is not driven until PTRDY# is asserted by add-on logic. For writes to APTD, data is clocked into the S5933 on every BPCLK rising edge, but is not passed to the PCI bus until PTRDY# is asserted. PTRDY# must by synchronized to BPCLK.

8

8.4.1 Pass-thru Status Indicators

The pass-thru status indicators indicate that a pass-thru access is in process and what action is required by the add-on logic to complete the access. All pass-thru status indicators are synchronous with the PCI clock.

8.4.2 Pass-thru Control Inputs

Some pass-thru implementations may require an address corresponding to the pass-thru data. The Add-on Pass-thru Address Register (APTA) contains the PCI address for the pass-thru cycle. To allow access to the pass-thru address without generating an add-on read cycle, PTADR# is provided. PTADR# is a direct access input for the pass-thru address. Asserting PTADR# is functionally identical to accessing the pass-thru address register with RD#, SELECT#, BE[3:0]#, and ADR[6:2]. RD# and WR# must be deasserted when PTADR# is asserted, but SELECT# may be asserted. These inputs automatically drive the address (internally) to 28h and assert all byte enables. The ADR[6:2] and BE[3:0]# are ignored when using the PTADR# direct access input. When PTADR# is asserted, the contents of the APTA register are immediately driven onto the add-on data bus.

The PTADR# direct access signal accesses the pass-thru address register as 16-bits or 32-bits, whatever the MODE pin is configured for. For 16-bit mode, PTADR# only presents the lower 16-bits of the APTA register. For add-ons requiring a region larger than 64 Kbytes of addressability, the upper 16-bits of APTA may be read using PTADR# with ADR1 set.

PTRDY# indicates that the add-on has completed the current pass-thru access. Multiple add-on reads or writes may occur to the pass-thru data (APTD) register before asserting PTRDY#. This may be required for 8-bit or 16-bit add-on interfaces using multiple accesses to the 32-bit pass-thru data register. In some cases, the add-on bus may be 32-bits, but logic may require multiple BPCLK periods to read or write data. In this situation, accesses may be extended by holding off PTRDY#. PTRDY# must be synchronized to BPCLK.

8.5 NON-VOLATILE MEMORY INTERFACE

The S5933 allows read and write access to the nv memory device used for configuration. Reads are necessary during device initialization as configuration information is downloaded into the S5933. If an expansion BIOS is implemented in the nv memory, the host transfers (shadows) the code into system DRAM. Writes are useful for in-field updates to expansion BIOS code. This allows software to update the nv memory contents without altering hardware.

8.5.1 Non-Volatile Memory Interface Signals

For serial nv memory devices, there are only two signals used to interface with nv memory. SCL is the serial clock, and SDA is the serial data line. The functionality of these signals is described in-detail in Section 6.3. The designer does not need to generate the timings for SCL and SDA. The S5933 automatically performs the correct serial access when programmed as described in Section 8.5.2.

For byte-wide nv memory devices, there is an 8-bit data bus (EQ7:0), and a 16-bit address bus (EA15:0) dedicated for the nv memory interface. When a serial nv memory is implemented, many of these pins have alternate functions. The S5933 also has read (ERD#) and write (EWR#) outputs to drive the OE# and WR# inputs on a byte-wide nv memory. The designer does not need to generate the timings for these outputs. The S5933 automatically performs the read and write accesses when programmed as described in Section 8.5.2.

8.5.2 Accessing Non-Volatile Memory

The nv memory, if implemented, can be accessed through the PCI interface or the add-on interface. Accesses from both the PCI side and the add-on side must be synchronous with the PCI clock (BPCLK for the add-on). Accesses to the nv memory from the PCI interface are through the Bus Master Control/ Status Register (MCSR) PCI Operation Register. Accesses to the nv memory from the add-on interface are through the Add-on General Control/Status Register (AGCSTS) Add-on Operation Register.

Accesses to the MCSR register are from the PCI bus and are, therefore, automatically synchronous to the PCI clock. Accesses to the AGCSTS register from the add-on side must be synchronous with respect to BPCLK (as described in Section 8.1.4).

Some nv memories may contain Expansion ROM BIOS code for use by the host software. During initialization, the Expansion BIOS is located within system memory. The starting location of the nv memory is stored in the Expansion ROM Base Address Register in the S5933 PCI Configuration Registers. A PCI read from this region results in the S5933 performing four consecutive byte access to the nv memory device. Writes to the nv memory are not allowed by writing to this region. Writes to the nv memory must be performed as described below.

In the MCSR and AGCSTS registers, bits D31:29 are command/status bits and bits D23:16 are address/data bits. These operation registers occupy the same offset (3Ch-3Fh) on their respective interfaces (add-on and PCI). The sequence used to access the nv memory is the same, regardless of whether the device is serial or byte-wide. The sequence is shown for 8-bit accesses to the operation register. Steps may be combined by using 16-bit accesses to the operation register. The following sequence is required for writes:

1) Write low address byte:
  Write offset 3Fh = 80h
  Write offset 3Eh = [low address byte]

2) Write high address byte:
  Write offset 3Fh = A0h
  Write offset 3Eh = [high address byte]

3) Wait for nv device-not-busy:
  Read offset 3Fh until D7 = 0

4) Write data to be written:
  Write offset 3Eh = [data]

5) Issue write command:
  Write offset 3Fh = C0h

6) Wait for nv device-not-busy:
  Read offset 3Fh until D7 = 0

If a valid expansion ROM is not found in the nv memory, the Expansion ROM Base Address Register may not be used to read nv memory locations. In this situation, nv memory reads must be performed through MCSR or AGCSTS. The following sequence is required for reads:

1) Write low address byte:
  Write offset 3Fh = 80h
  Write offset 3Eh = [low address byte]

2) Write high address byte:
  Write offset 3Fh = A0h
  Write offset 3Eh = [high address byte]

3) Wait for nv device-not-busy:
  Read offset 3Fh until D7 = 0

4) Issue read command:
  Write offset 3Fh = E0h

5) Wait for nv device-not-busy:
  Read offset 3Fh until D7 = 0

6) Read data:
  Read offset 3Eh = [nv memory data]

8.5.3 nv Memory Device Timing Requirements

For serial nv memory devices, the serial clock output frequency is the PCI clock frequency divided by 512. This is approximately 65 KHz (with a 33 MHz PCI clock). Any serial memory device that operates at this frequency is compatible with the S5933.

For byte-wide accesses, the S5933 generates the waveforms shown in Figures 8-5 and 8-6. Figure 8-5 shows an nv memory read operation. Figure 8-6 shows an nv memory write operation. Exact timings are provided in the Electrical and AC Characteristics chapter (Chapter 12). Read operations are always the same length. Write operations, due to the characteristics of reprogrammable nv memory devices, may be controlled through the programming sequence described in Section 8.5.2.

8

Memory Device Requirements for Read Accesses

Timing	Spec.	T = 30 ns
Read cycle time	8T(max)	240 ns
Address valid to data valid	7T−10(max)	200 ns
Address valid to read active	T(max)	30 ns
Read active to data valid	6T−10(max)	170 ns
Read pulse width	6T(max)	180 ns
Data hold from read inactive	—	2 ns

Figure 8-5. nv Memory Read Operation
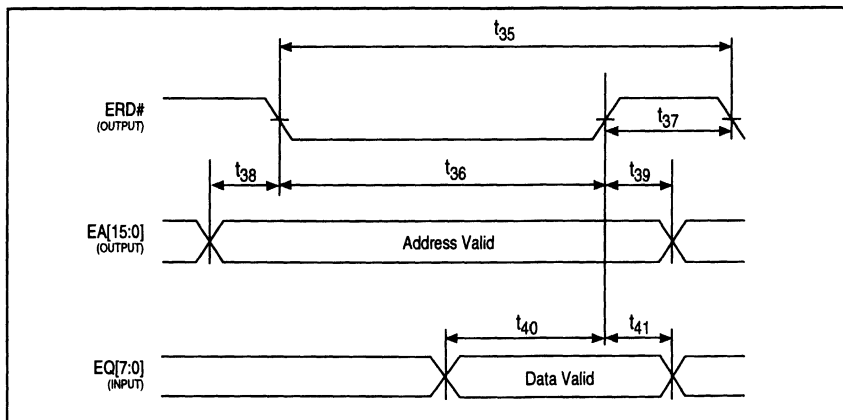
Memory Device Requirements for Write Accesses

Timing	Spec.	T = 30 ns
Write cycle time	8T	Note 1
Address valid to write active	T(max)	30 ns
Data valid to write inactive	6T+10(max)	190 ns
Data hold from write inactive	T(max)	30 ns
Write pulse width	6T(max)	180 ns
Write inactive	Note 2	2 ns

Note 1: The write cycle time is determined by the programming sequence described in Section 8.5.2.

Note 2: The write inactive time is determined by the programming sequence described in Section 8.5.2.

Figure 8-6. nv Memory Write Operation

8

Mailbox Description 9
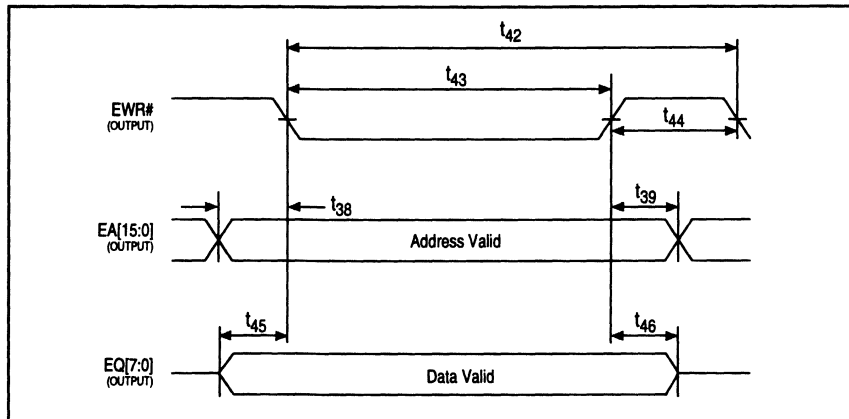
9.0 MAILBOX OVERVIEW

The S5933 has eight 32-bit mailbox registers. The mailboxes are useful for passing command and status information between the add-on and the PCI bus. The PCI interface has four incoming mailboxes (add-on to PCI) and four outgoing mailboxes (PCI to add-on). The add-on interface has four incoming mailboxes (PCI to add-on) and four outgoing mailboxes (add-on to PCI). The PCI incoming and add-on outgoing mailboxes are the same, internally. The add-on incoming and PCI outgoing mailboxes are also the same, internally.

The mailbox status may be monitored in two ways. The PCI and add-on interfaces each have a mailbox status register to indicate the empty/full status of bytes within the mailboxes. The mailboxes may also be configured to generate interrupts to the PCI and/or add-on interface. One outgoing and one incoming mailbox on each interface can be configured to generate interrupts.

9.1 FUNCTIONAL DESCRIPTION

Figure 9-1 shows a block diagram of the PCI to add-on mailbox registers. Add-on incoming mailbox read accesses pass through an output interlock latch. This prevents a PCI bus write to a PCI outgoing mailbox from corrupting data being read by the add-on. Figure 9-2 shows a block diagram of the add-on to PCI mailbox registers. PCI incoming mailbox reads also pass through an interlocking mechanism. This prevents an add-on write to an outgoing mailbox from corrupting data being read by the PCI bus. The following sections describe the mailbox flag functionality and the mailbox interrupt capabilities.

Figure 9-1. Block Diagram - PCI to Add-on Mailbox Register
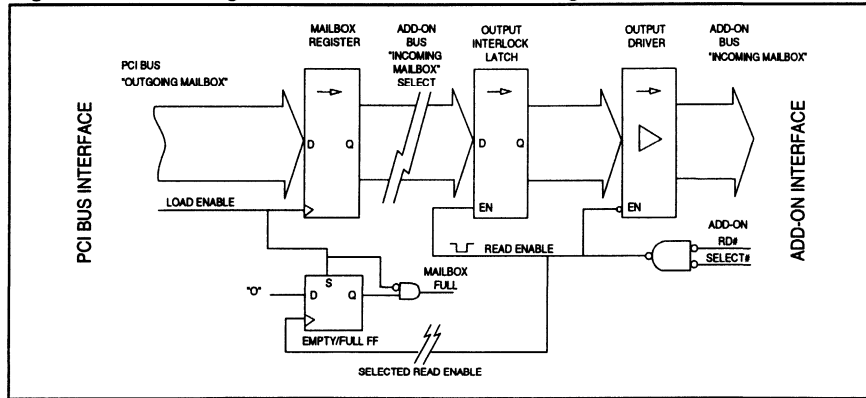
Figure 9-2. Block Diagram - Add-on to PCI Mailbox Register
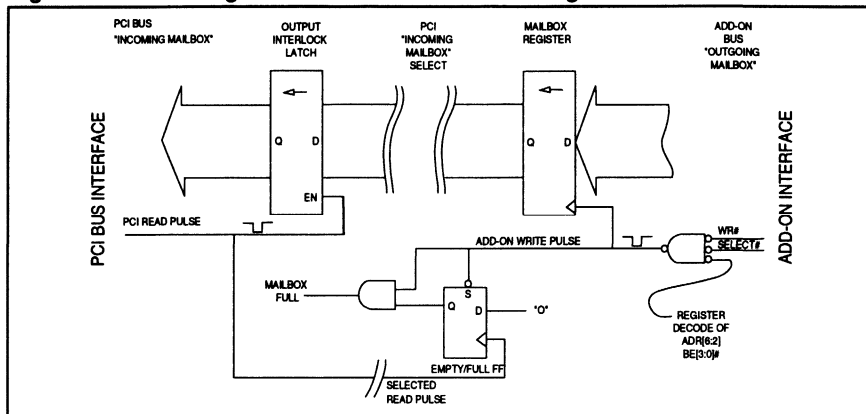
9.1.1 Mailbox Empty/Full Conditions

The PCI and add-on interfaces each have a mailbox status register. The PCI Mailbox Empty/Full Status (MBEF) and Add-on Mailbox Empty/Full Status (AMBEF) Registers indicate the status of all bytes within the mailbox registers. A write to an outgoing mailbox sets the status bits for that mailbox. The byte enables determine which bytes within the mailbox become full (and which status bits are set).

An outgoing mailbox for one interface is an incoming mailbox for the other. Therefore, incoming mailbox status bits on one interface are identical to the corresponding outgoing mailbox status bits on the other interface. The following list shows the relationship between the mailbox registers on the PCI and add-on interfaces.

PCI Interface		Add-on Interface
Outgoing Mailbox 1	=	Incoming Mailbox 1
Outgoing Mailbox 2	=	Incoming Mailbox 2
Outgoing Mailbox 3	=	Incoming Mailbox 3
Outgoing Mailbox 4	=	Incoming Mailbox 4
Incoming Mailbox 1	=	Outgoing Mailbox 1
Incoming Mailbox 2	=	Outgoing Mailbox 2
Incoming Mailbox 3	=	Outgoing Mailbox 3
Incoming Mailbox 4	=	Outgoing Mailbox 4
PCI Mailbox Empty/Full	=	Add-on Mailbox Empty/Full

A write to an outgoing mailbox also writes data into the incoming mailbox on the other interface. It also sets the status bits for the outgoing mailbox and the status bits for the incoming mailbox on the other interface. Reading the incoming mailbox clears all corresponding status bits in the add-on and PCI mailbox status registers (AMBEF and MBEF).

For example, a PCI write is performed to the PCI outgoing mailbox 2, writing bytes 0 and 1 (BE0# and BE1# asserted). Reading the PCI Mailbox Empty/Full Status Register (MBEF) indicates that bits 4 and 5 are set. These bits indicate that outgoing mailbox 2, bytes 0 and 1 are full. Reading the Add-on Mailbox Empty/Full Status Register (AMBEF) shows that bits 4 and 5 in this register are also set, indicating add-on incoming mailbox 2, bytes 0 and 1 are full. An add-on read of incoming mailbox 2, bytes 0 and 1 clears the status bits in both the MBEF and AMBEF status registers.

To reset individual flags in the MBEF and AMBEF registers, the corresponding byte must be read from the incoming mailbox. The PCI and add-on mailbox status registers, MBEF and AMBEF, are read-only. Mailbox flags may be globally reset from either the PCI interface or the add-on interface. The PCI Bus Master Control/Status Register (MCSR) and the add-on General Control/Status Register (AGCSTS) each have a bit to reset all of the mailbox status flags.

9.1.2 Mailbox Interrupts

The designer has the option to generate interrupts to the PCI and add-on interfaces when specific mailbox events occur. The PCI and add-on interfaces can each define two conditions where interrupts may be generated. An interrupt can be generated when an incoming mailbox becomes full and/or when an outgoing mailbox becomes empty. A specific byte within a specific mailbox is selected to generate the interrupt. The conditions defined to generate interrupts to the PCI interface do not have to be the same as the conditions defined for the add-on interface. Interrupts are cleared through software as described in Section 9.3.2.

For incoming mailbox interrupts, when the specified byte becomes full, an interrupt is generated. The interrupt might be used to indicate command or status information has been provided, and must be read. For PCI incoming mailbox interrupts, the S5933 asserts the PCI interrupt, INTA#. For add-on incoming mailbox interrupts, the S5933 asserts the add-on interrupt, IRQ#.

For outgoing mailbox interrupts, when the specified byte becomes empty, an interrupt is generated. The interrupt might be used to indicate that the other interface has received the last information sent and more may be written. For PCI outgoing mailbox interrupts, the S5933 asserts the PCI interrupt, INTA#. For add-on outgoing mailbox interrupts, the S5933 asserts the add-on interrupt, IRQ#.

9.1.3 Add-on Outgoing Mailbox 4, Byte 3 Access

PCI incoming mailbox 4, byte 3 (add-on outgoing mailbox 4, byte 3) does not function exactly like the other mailbox bytes. When an a serial nv memory boot device or no external boot device is used, the S5933 pins EA7:0 are redefined to provide direct external access to add-on outgoing mailbox 4, byte 3. EA8 is redefined to provide a load clock which may be used to generate a PCI interrupt. The pins are redefined as follows:

Signal Pin	Add-on Outgoing Mailbox
EA0/EMB0	Mailbox 4, bit 24
EA1/EMB1	Mailbox 4, bit 25
EA2/EMB2	Mailbox 4, bit 26
EA3/EMB3	Mailbox 4, bit 27
EA4/EMB4	Mailbox 4, bit 28
EA5/EMB5	Mailbox 4, bit 29
EA6/EMB6	Mailbox 4, bit 30
EA7/EMB7	Mailbox 4, bit 31
EA8/EMBCLK	Mailbox 4, byte 3 load clock

If the S5933 is programmed to generate a PCI interrupt (INTA#), on an add-on write to outgoing mailbox 4, byte 3, a rising edge on EMBCLK generates a PCI interrupt. The bits EMB7:0 can be read by the PCI bus interface by reading the PCI incoming mailbox 4, byte 3. These bits are useful to indicate various conditions which may have caused the interrupt.

When using the S5933 with a byte-wide boot device, the capability to generate PCI interrupts with add-on hardware does not exist. In this configuration, PCI incoming mailbox 4, byte 3 (add-on incoming mailbox 4, byte 3) cannot be used to transfer data from the add-on - it always returns zeros when read from the PCI bus. This mailbox byte is sacrificed to allow the added functionality provided when a byte-wide boot device is not used.

9.2 BUS INTERFACE

The mailboxes appear on the add-on and PCI bus interfaces as eight operation registers. Four are outgoing mailboxes, four are incoming mailboxes. The mailboxes may be used to generate interrupts to each of the interfaces. The following sections describe the add-on and PCI bus interfaces for the mailbox registers.

9.2.1 PCI Bus Interface

The mailboxes are only accessible with the S5933 as a PCI target. The mailbox operation registers do not support burst accesses by an initiator. A PCI initiator attempting to burst to the mailbox registers causes the S5933 to respond with a target disconnect with data (Section 7.1.5.1.). PCI writes to full outgoing mailboxes overwrite data currently in that the mailbox. PCI reads from empty incoming mailboxes return the data that was previously contained in the mailbox. Neither of these situations cause a target retry or abort.

PCI incoming and outgoing mailbox interrupts are enabled in the Interrupt Control/Status Register (INTCSR). The mailboxes can generate a PCI interrupt (INTA#) under two conditions (individually enabled). For an incoming mailbox full interrupt, INTA# is asserted on the PCI clock rising edge after the add-on mailbox write completes. For an outgoing mailbox empty interrupt, INTA# is asserted on the PCI clock rising edge after the add-on mailbox read completes (the rising edge of RD#). INTA# is deasserted on the next PCI clock rising edge after the PCI access to clear the mailbox interrupt completes (TRDY# deasserted).

9.2.2 Add-on Bus Interface

The add-on mailbox interface behaves similar to the PCI bus interface. Add-on writes to full outgoing mailboxes overwrite data currently in that mailbox. PCI reads from empty incoming mailboxes return the data that was previously contained in the mailbox.

Add-on incoming and outgoing mailbox interrupts are enabled in the Add-on Interrupt Control/Status Register (AINT). The mailboxes can generate the add-on IRQ# interrupt under two conditions (individually enabled). For an incoming mailbox full interrupt, IRQ# is asserted one PCI clock period after the PCI mailbox write completes (TRDY# deasserted). For an outgoing mailbox empty interrupt, IRQ# is asserted one PCI clock period after the PCI mailbox read completes (TRDY# deasserted). IRQ# is deasserted immediately when the add-on clears the mailbox interrupt (see Chapter 12 for exact timing).

When the S5933 is used with a serial nv memory boot device or no external boot device, the device pins EA8:0 are redefined as shown in Section 9.1.3. EA7:0 become EMB7:0 data inputs and EA8 becomes EMBCLK, a load clock. This configuration allows the add-on to generate PCI interrupts with a low-to-high transition on EMBCLK. The PCI incoming mailbox interrupt must be enabled and set for mailbox 4, byte3 in the PCI Interrupt Control/Status Register (INTCSR). EMBCLK should begin high and be pulsed low, then high to be recognized (see Chapter 12 for exact timing). The rising edge of EMBCLK generates the interrupt. The rising edge of EMBCLK also latches in the values on EMB7:0. The S5933 interrupt logic must be cleared (INTA# deasserted) through INTCSR before further EMBCLK interrupts are recognized.

9.2.2.1 8-Bit and 16-Bit Add-on Interfaces

Some add-on designs may implement an 8-bit or 16-bit bus interface. The mailboxes do not require a 32-bit add-on interface. For 8-bit interfaces, the 8-bit data bus may be externally connected to all four bytes of the 32-bit add-on interface (DQ 31:24, 23:16, 15:8, 7:0 are all connected). The add-on device reading or writing the mailbox registers may access all mailbox bytes by cycling through the add-on byte enable inputs. A similar solution applies to 16-bit add-on buses. This solution works for add-ons which always use just 8-bit or just 16-bit accesses.

If the MODE pin is high, indicating a 16-bit add-on interface, the previous solution may be modified for an 8-bit interface. The difference is that ADR1 must be toggled after the first two accesses to steer the S5933 internal data bus to the upper 16-bits of the mailboxes.

9

9.3 CONFIGURATION

The PCI interface and the add-on interface each have four incoming mailboxes (IMBx or AIBMx) and four outgoing mailboxes (OMBx or AOMBx) along with a single mailbox status register (MBEF or AMBEF). Outgoing mailboxes are read/write, incoming mailboxes and the mailbox status registers are read-only.

The following sections discuss the registers associated with the mailboxes and accesses required for different modes of mailbox operation.

9.3.1 Mailbox Status

Every byte in each mailbox has a status bit in the Mailbox Empty/Full Status Registers (MBEF and AMBEF). Writing a particular byte into an outgoing mailbox sets the corresponding status bit in both the MBEF and AMBEF registers. A read of a 'full' byte in a mailbox clears the status bit. The MBEF and AMBEF are read-only. Status bits cannot cleared by writes to the status registers.

The S5933 allows the mailbox status bits to be reset through software. The Bus Master Control/Status (MCSR) PCI Operation Register and the Add-on General Control/Status (AGCSTS) Add-on Operation Register each have a bit to reset mailbox status. Writing a '1' to Mailbox Flag Reset bit in the MCSR or the AGCSTS register immediately clears all bits in the both the MBEF and AMBEF registers. Writing a '0' has no effect. The Mailbox Flag Reset bit is write-only.

The flag bits should be monitored when transferring data through the mailboxes. Checking the mailbox status before performing an operation prevents data from being lost or corrupted. The following sequences are suggested for PCI mailbox operations using status polling (interrupts disabled):

Reading a PCI Incoming Mailbox:

1) Check Mailbox Status. Read the mailbox status register to determine if any information has been passed from the add-on interface.

MBEF	Bits 31:16	If a bit is set, valid data is contained in the corresponding mailbox byte.

2) Read Mailbox(es). Read the mailbox bytes which MBEF indicates are full. This automatically resets the status bits in the MBEF and AMBEF registers.

IMBx	Bits 31:0	Mailbox data.

Writing a PCI Outgoing Mailbox:

1) Check Mailbox Status. Read the mailbox status register to determine if information previously written to the mailbox has been read by the add-on interface. Writes to full mailbox bytes overwrite data currently in the mailbox (if not already read by the add-on interface). Repeat until the byte(s) to be written are empty.

MBEF	Bits 15:0	If a bit is set, valid data is contained in the corresponding mailbox byte and has not been read by the add-on.

2) Write Mailbox(es). Write to the outgoing mailbox byte(s).

OMBx	Bits 31:0	Mailbox data.

Mailbox operations for the add-on interface are functionally identical. The following sequences are suggested for add-on mailbox operations using status polling (interrupts disabled):

Reading an Add-on Incoming Mailbox:

1) Check Mailbox Status. Read the mailbox status register to determine if any information has been passed from the PCI interface.

AMBEF	Bits 15:0	If a bit is set, valid data is contained in the corresponding mailbox byte.

2) Read Mailbox(es). Read the mailbox bytes which AMBEF indicates are full. This automatically resets the status bits in the AMBEF and MBEF registers.

AIMBx	Bits 31:0	Mailbox data.

Writing an Add-on Outgoing Mailbox:

1) Check Mailbox Status. Read the mailbox status register to determine if information previously written to the mailbox has been read by the PCI interface. Writes to full mailbox bytes overwrite data currently in the mailbox (if not already read by the PCI interface). Repeat until the byte(s) to be written are empty.

AMBEF	Bits 31:16	If a bit is set, valid data is contained in the corresponding mailbox byte and has not been read by the PCI bus.

2) Write Mailbox(es). Write to the outgoing mailbox byte(s).

AOMBx	Bits 31:0	Mailbox data.

9.3.2 Mailbox Interrupts

Although polling status is useful, in some cases, polling requires continuous actions by the processor reading or writing the mailbox. Mailbox interrupt capabilities are provided to avoid much of the processor overhead required by continuously polling status bits.

The add-on and PCI interface can each generate interrupts on an incoming mailbox condition and/or an outgoing mailbox condition. These can be individually enabled/disabled. A specific byte in one incoming mailbox and one outgoing mailbox is identified to generate the interrupt(s). The tasks required to setup mailbox interrupts are shown below:

Enabling PCI mailbox interrupts:

1) Enable PCI outgoing mailbox interrupts. A specific byte within one of the outgoing mailboxes is identified to assert INTA# when read by the add-on interface.

INTCSR	Bit 4	Enable outgoing mailbox interrupts
INTCSR	Bits 3:2	Identify mailbox to generate interrupt
INTCSR	Bits 1:0	Identify mailbox byte to generate interrupt

2) Enable PCI incoming mailbox interrupts. A specific byte within one of the incoming mailboxes is identified to assert INTA# when written by the add-on interface.

INTCSR	Bit 12	Enable incoming mailbox interrupts
INTCSR	Bits 11:10	Identify mailbox to generate interrupt
INTCSR	Bits 9:8	Identify mailbox byte to generate interrupt

Enabling add-on mailbox interrupts:

1) Enable add-on outgoing mailbox interrupts. A specific byte within one of the outgoing mailboxes is identified to assert IRQ# when read by the PCI interface.

AINT	Bit 12	Enable outgoing mailbox interrupts
AINT	Bits 11:10	Identify mailbox to generate interrupt
AINT	Bits 9:8	Identify mailbox byte to generate interrupt

2) Enable add-on incoming mailbox interrupts. A specific byte within one of the incoming mailboxes is identified to assert IRQ# when written by the PCI interface.

AINT	Bit 4	Enable incoming mailbox interrupts
AINT	Bits 3:2	Identify mailbox to generate interrupt
AINT	Bits 1:0	Identify mailbox byte to generate interrupt

With either the add-on or PCI interface, these two steps can be performed with a single access to the appropriate register. They are shown separately here for clarity.

Once interrupts are enabled, the interrupt service routine must access the mailboxes and clear the interrupt source. A particular application may not require all of the steps shown. For instance, a design may only use incoming mailbox interrupts and not require support for outgoing mailbox interrupts. The interrupt service routine tasks are shown below:

Servicing a PCI mailbox interrupt (INTA#):

1) Identify the interrupt source(s). Multiple interrupt sources are available on the S5933. The interrupt service routine must verify that a mailbox generated the interrupt (and not some other interrupt source).

| INTCSR | Bit 16 | PCI outgoing mailbox interrupt indicator |
| INTCSR | Bit 17 | PCI incoming mailbox interrupt indicator |

2) Check mailbox status. The mailbox status bits indicate which mailbox bytes must be read or written.

| MBEF | Bits 31:16 | Full PCI incoming mailbox bytes |
| MBEF | Bits 15:0 | Empty PCI outgoing mailbox bytes |

3) Access the mailbox. Based on the contents of MBEF, mailboxes are read or written. Reading an incoming mailbox byte clears the corresponding status bit in MBEF.

| OMBx | Bits 31:0 | PCI outgoing mailboxes |
| IMBx | Bits 31:0 | PCI incoming mailboxes |

4) Clear the interrupt source. The PCI INTA# signal is deasserted by clearing the interrupt request. The request is cleared by writing a '1' to the appropriate bit.

| INTCSR | Bit 16 | Clear PCI outgoing mailbox interrupt |
| INTCSR | Bit 17 | Clear PCI incoming mailbox interrupt |

Servicing an add-on mailbox interrupt (IRQ#):

1) Identify the interrupt source(s). Multiple interrupt sources are available on the S5933. The interrupt service routine must verify that a mailbox generated the interrupt (and not some other interrupt source).

AINT	Bit 16	Add-on incoming mailbox interrupt indicator
AINT	Bit 17	Add-on outgoing mailbox interrupt indicator

2) Check mailbox status. The mailbox status bits indicate which mailbox bytes must be read or written.

AMBEF	Bits 31:16	Empty add-on outgoing mailbox bytes
AMBEF	Bits 15:0	Full add-on incoming mailbox bytes

3) Access the mailbox. Based on the contents of AMBEF, mailboxes are read or written. Reading an incoming mailbox byte clears the corresponding status bit in AMBEF.

AIMBx	Bits 31:0	Add-on incoming mailboxes
AOMBx	Bits 31:0	Add-on outgoing mailboxes

4) Clear the interrupt source. The add-on IRQ# signal is deasserted by clearing the interrupt request. The request is cleared by writing a '1' to the appropriate bit.

AINT	Bit 16	Clear add-on incoming mailbox interrupt
AINT	Bit 17	Clear add-on outgoing mailbox interrupt

In both cases, step 3 involves accessing the mailbox. To allow the incoming mailbox interrupt logic to be cleared, the mailbox status bit must also be cleared. Reading an incoming mailbox clears the status bits. Another option for clearing the status bits is to use the Mailbox Flag Reset bit in the MCSR and AGCSTS registers, but this clears all status bits, not just for a single mailbox or mailbox byte. For outgoing mailbox interrupts, the read of a mailbox register is what generated the interrupt; this ensures the status bits are already clear.

9

FIFO Description 10

10.0 FIFO OVERVIEW

The S5933 has two internal FIFOs. One FIFO is for PCI bus to add-on bus, the other FIFO is for add-on bus to PCI bus transfers. Each of these has eight 32-bit registers. The FIFOs are both addressed through a single PCI/Add-on Operation Register offset, but which internal FIFO is accessed is determined by whether the access is a read or write.

The FIFO may be either a PCI target or a PCI initiator. As a target, the FIFO allows a PCI bus master to access add-on data. The FIFO also allows the S5933 to become a PCI initiator. Read and write address registers and transfer count registers allow the S5933 to perform DMA transfers across the PCI bus. The FIFO may act as initiator and a target at different times in the same application.

The FIFO can be configured to support various add-on bus configurations. FIFO status and control signals allow simple cascading into an external FIFO, the add-on bus can be 8-, 16-, or 32-bits wide, and data endian conversion is optional to support any type of add-on CPU. PCI and add-on interrupt capabilities are available to support bus mastering through the FIFO.

10.1 FUNCTIONAL DESCRIPTION

The S5933 FIFO interface allows a high degree of functionality and flexibility. Different FIFO management schemes, endian conversion schemes, and advance conditions allow for a wide variety of add-on interfaces. Applications may implement the FIFO as either a PCI target or program it to enable the S5933 to be a PCI initiator (bus master). The following sections describe, on a functional level, the capabilities of the S5933 FIFO interface.

10.1.1 FIFO Buffer Management and Endian Conversion

The S5933 provides a high degree of flexibility for controlling the data flow through the FIFO. Each FIFO (PCI to add-on and add-on to PCI) has a specific FIFO advance condition. For FIFO writes, the byte which signifies a location is full is configurable. For FIFO reads, the byte which signifies a location is empty is configurable. This ability is useful for transferring data through the FIFO with add-ons which are not 32-bits wide. Endian conversion may also be performed on data passing through the FIFO.

10.1.1.1 FIFO Advance Conditions

The specific byte lane used to advance the FIFO, when accessed, is determined individually for each FIFO interface (PCI and add-on). The control bits to set the advance condition are D29:26 of the Interrupt Control/Status Register (INTCSR) in the PCI Operation Registers (Figure 10-1). The default FIFO advance condition is set to byte 0. With the default setting, a write to the FIFO with BE0# asserted indicates that the FIFO location is now full, advancing the FIFO pointer to the next location. BE0# does not have to be the only byte enable asserted. Note, the FIFO advance condition may be different for the PCI to add-on FIFO and the add-on to PCI FIFO directions.

10

Figure 10-1. INTCSR FIFO Advance and Endian Control Bits
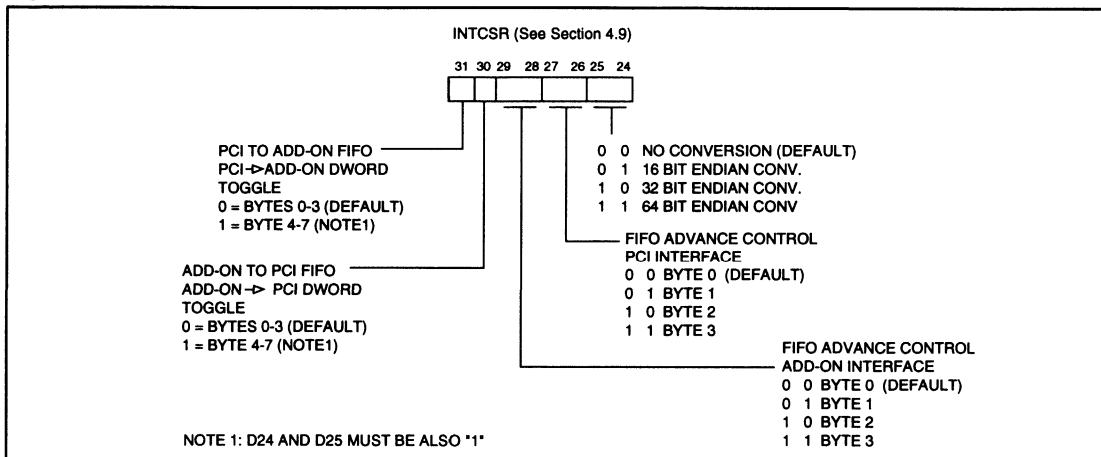
The configurable FIFO advance condition may be used to transfer data to and from add-on interfaces which are not 32-bits wide. For a 16-bit add-on bus, the add-on to PCI FIFO advance condition can be set to byte 2. This allows a 16-bit write to the lower 16-bits of the FIFO register (bytes 0 and 1) and a second write to the upper 16-bits of the FIFO register (bytes 2 and 3). The FIFO does not advance until the second access. This allows the add-on to operate with 16-bit data, while the PCI bus maintains a 32-bit datapath.

Note: During operation, the INTCSR FIFO advance condition bits (D29:26) should only be changed when the FIFO is empty and is idle on both the add-on and PCI interfaces.

10.1.1.2 Endian Conversion

Bits D31:30 and D25:24 of the INTCSR PCI Operation Register control endian conversion operations for the FIFO (Figure 10-1). When endian conversion is performed, it affects data passing in either direction through the FIFO interface. Figures 10-2a and 10-2b show 16-bit and 32-bit endian conversion. It is important to note that endian conversion is performed on data BEFORE it enters the FIFO. This affects the FIFO advance condition. Example: the FIFO is configured to perform 32-bit endian conversion on data, and the FIFO advance condition is set to byte 0. Byte 3 is written into the FIFO (BE3# asserted). After the endian conversion, byte 3 becomes byte 0, and the FIFO advances. This behavior must be considered when not performing full 32-bit accesses to the FIFO.

Note: During operation, the INTCSR FIFO endian conversion bits (D25:24) and 64-bit access bits (D31:30) should only be changed when the FIFO is empty and is idle on both the add-on and PCI interfaces.

Figure 10-2a. 16-bit Endian Conversion
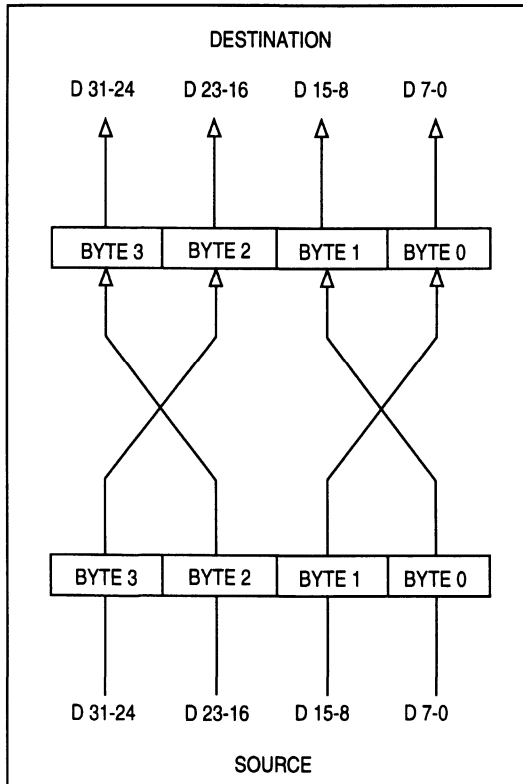
Figure 10-2b. 32-bit Endian Conversion
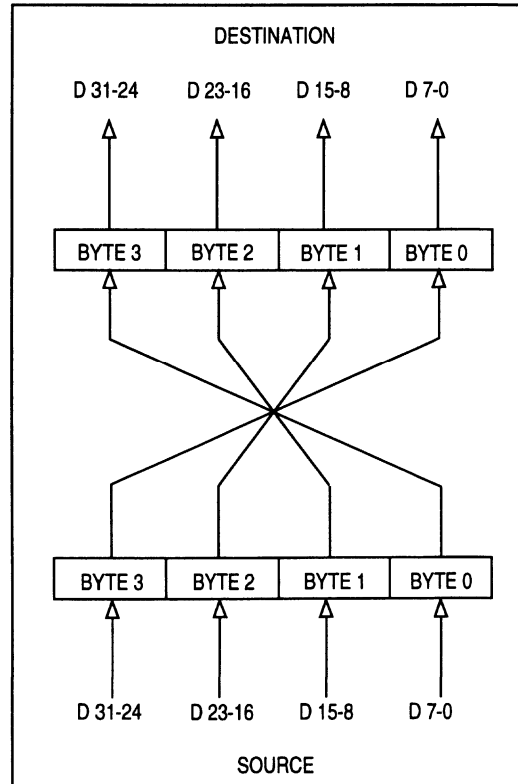
10.1.1.3 64-Bit Endian Conversion

Because the S5933 interfaces to a 32-bit PCI bus, special operation is required to handle 64-bit data endian conversion. Figure 10-2c shows 64-bit endian conversion. The S5933 must know whether the lower 32-bits enter the FIFO first or the upper 32-bits enter the FIFO first. INTCSR D31:30 identify which method is used by the application. These bits toggle after each 32-bit operation to indicate if half or all of a 64-bit data operation has been completed. The initial state of these bits establishes the loading and emptying order for 64-bit data during operation.

Figure 10-2c. 64-bit Endian Conversion

10
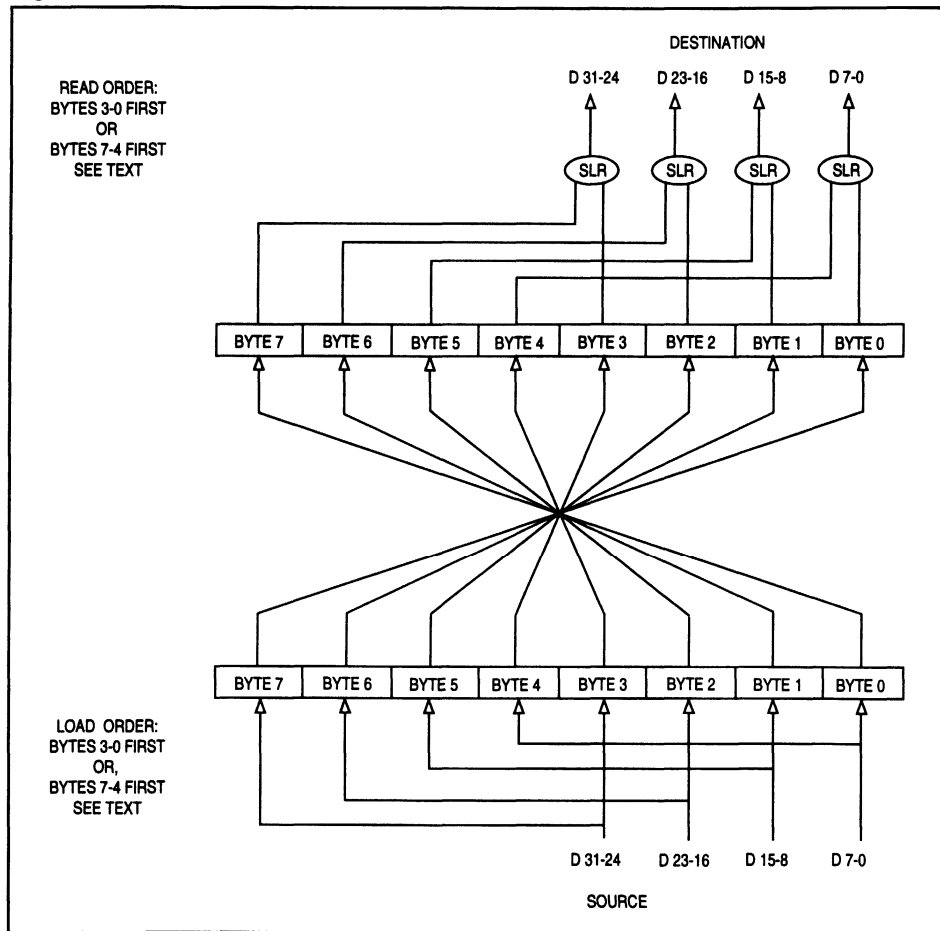
10.1.2 Add-on FIFO Status Indicators

The add-on interface implements FIFO status pins to indicate the full and empty conditions of the PCI to add-on and add-on to PCI FIFOs. These may be used by the add-on to allow data transfers between the FIFO and memory, a peripheral, or even a cascaded external FIFO. The RDEMPTY and WRFULL status outputs are always available to the add-on. Additional status signals are multiplexed with the byte-wide, non-volatile memory interface pins. If the S5933 is configured for add-on initiated bus mastering, these status signals also become available to the add-on. FIFO status is also indicated by bits in the Add-on General Control/Status and Bus Master Control/Status Registers. The table below lists all FIFO status outputs and their functions.

Signal	Function
RDEMPTY	Indicates empty condition of the PCI to add-on FIFO
WRFULL	Indicates full condition of the add-on to PCI FIFO
FRF	Indicates full condition of the PCI to add-on FIFO (note 1)
FWE	Indicates the empty condition of the add-on to PCI FIFO (note 1)

Note 1: These signals are only available when a serial non-volatile memory is used and the device is configured for add-on initiated bus mastering (Section 10.3.1).

10.1.3 Add-on FIFO Control Signals

The add-on interface implements FIFO control pins to manipulate the S5933 FIFOs. These may be used by add-on to control data transfer between the FIFO and memory, a peripheral, or even a cascaded external FIFO. The RDFIFO# and WRFIFO# inputs are always available. These pins allow direct access to the FIFO without generating a standard add-on register access using RD#, WR#, SELECT#, address pins and the byte enables.

Additional control signals are multiplexed with the byte-wide, non-volatile memory interface pins. If a serial non-volatile memory is used and the S5933 is configured for add-on initiated bus mastering, these control signals also become available. For PCI initiated bus mastering, AMREN, AMWEN, FRC#, and FWC# functionality is always available through bits in the Bus Master Control/Status and Add-on General Control/Status Registers. The FIFO control inputs are listed below.

Signal	Function
RDFIFO#	Reads data from the PCI to add-on FIFO
WRFIFO#	Writes data into the add-on to PCI FIFO
FRC#	Reset PCI to add-on FIFO pointers and status indicators (note 1)
FWC#	Reset add-on to PCI FIFO pointers and status indicators (note 1)
AMREN	Enable bus mastering for add-on initiated PCI reads (note 1)
AMWEN	Enable bus mastering for add-on initiated PCI writes (note 1)

Note 1: These signals are only available when a serial non-volatile memory is used and the S5933 is configured for add-on initiated bus mastering (see Section 10.3.1).

10.1.4 PCI Bus Mastering with the FIFO

The S5933 may initiate PCI bus cycles through the FIFO interface. The S5933 allows blocks of data to be transferred to and from the add-on by specifying a source/destination address on the PCI bus and a transfer byte count. This DMA capability allows data to be transferred across the PCI bus without host CPU intervention.

Initiating a bus master transfer requires programming the appropriate address registers and transfer byte counts. This can be done from either the PCI interface or the add-on interface (configurable at reset, see section 10.3.1). Initiating bus master transfers from the add-on is advantageous because the host CPU does not have to intervene for the S5933 to become a PCI Initiator. At the end of a transfer the S5933 may generate an interrupt to either the PCI bus (for PCI initiated transfers) or add-on interface (for add-on initiated transfers).

10.1.4.1 Add-on Initiated Bus Mastering

If bit 7 in location 45h of an external serial non-volatile memory is zero, the Master Read Address Register (MRAR), Master Write Address Register (MWAR), Master Read Transfer Count (MRTC), and Master Write Transfer Count (MWTC) are accessible only from the add-on interface. Add-on initiated bus mastering is not possible when a byte-wide boot device is used due to shared device pins (see Section 10.3.1). When configured for add-on initiated bus mastering, the S5933 transfers data until the transfer count reaches zero, or it may be configured to ignore the transfer count.

For bus master transfers initiated by the add-on interface, some applications may not know the size of the data block to be transferred. To avoid constantly updating the transfer count register, the transfer count may be disabled. Bit 28 in the Add-on General Control/Status Register (AGCSTS) performs this function. Disabling the transfer count also disables the interrupt capabilities. Regardless of whether add-on transfer count is enabled or disabled, the Add-on Master Read Enable (AMREN) and Add-on Master Write Enable (AMWEN) inputs control when the S5933 asserts or deasserts its request to the PCI bus. When add-on transfer count is enabled, the S5933 will only request the bus when both the transfer count (read or write) is not zero and the appropriate enable line (AMREN or AMWEN) is active. For add-on initiated bus mastering, AMWEN and AMREN override the read and write bus mastering enable bits in the Bus Master Control/Status Register (MCSR).

10.1.4.2 PCI Initiated Bus Mastering

If bit 7 in location 45h of the external non-volatile memory is one, the Master Read Address Register (MRAR), Master Write Address Register (MWAR), Master Read Transfer Count (MRTC), and Master Write Transfer Count (MWTC) are accessible only from the PCI bus interface. In this configuration, the S5933 transfers data until the transfer count reaches zero. The transfer count cannot be disabled for PCI initiated bus mastering. If no external nv memory boot device is used, the S5933 defaults to PCI initiated bus mastering.

10.1.4.3 Address and Transfer Count Registers

The S5933 has two sets of registers used for bus master transfers. There are two operation registers for bus master read operations and two operation registers for bus master write operations. One operation register is for the transfer address (MWAR and MRAR). The other operation register is for the transfer byte count (MWTC and MRTC).

The address registers are written with the first address of the transfer before bus mastering is enabled. Once a transfer begins, this register is automatically updated to reflect the address of the current transfer. If a PCI target disconnects from an S5933 initiated cycle, the transfer is retried starting from the current address in the register. If bus grant (GNT#) is removed or bus mastering is disabled (using AMREN or AMWEN), the value in the address register reflects the next address to be accessed. Transfers must begin on DWORD boundaries.

The transfer count registers contain the number of bytes to be transferred. The transfer count may be written before or after bus mastering is enabled. If bus mastering is enabled, no transfer occurs until the transfer count is programmed with a non-zero value. Once a transfer begins, this register is automatically updated to reflect the number of bytes remaining to be transferred. If the transfer count registers are disabled (for add-on initiated bus mastering), transfers begin as soon as bus mastering is enabled.

Although transfers must begin on DWORD boundaries, transfer counts do not have to be multiples of four bytes. For example, if the write transfer count (MWTC) register is programmed with a value of 10 (decimal), the S5933 performs two DWORD writes and a third write with only BE0# and BE1# asserted.

10.1.4.4 Bus Mastering FIFO Management Schemes

The S5933 provides flexibility in how the FIFO is managed for bus mastering. The FIFO management scheme determines when the S5933 requests the bus to initiate PCI bus cycles. The management scheme is configurable for the PCI to add-on and add-on to PCI FIFO (and may be different for each). Bus mastering must be enabled for the management scheme to apply (via the enable bits or AMREN/AMWEN).

For the PCI to add-on FIFO, there are two management options. The PCI to add-on FIFO management option is programmed through the Bus Master Control/Status Register (MCSR). The FIFO can be programmed to request the bus when any DWORD location is empty or only when four or more locations are empty. After the S5933 is granted control of the PCI bus, the management scheme does not apply. The device continues to read as long as there is an open FIFO location. When the PCI to add-on FIFO is full or bus mastering is disabled, the PCI bus request is removed by the S5933.

For the add-on to PCI FIFO, there are two management options. The add-on to PCI FIFO management option is programmed through the Bus Master Control/Status Register (MCSR). The FIFO can be programmed to request the bus when any DWORD location is full or only when four or more locations are full. After the S5933 is granted control of the PCI bus, the management scheme does not apply. The device continues to write as long as there is data in the FIFO. When the add-on to PCI FIFO is empty or bus mastering is disabled, the PCI bus request is removed by the S5933.

10

There are two special cases for the add-on to PCI FIFO management scheme. The first case is when the FIFO is programmed to request the PCI bus only when four or more locations are full, but the transfer count is less than 16 bytes. In this situation, the FIFO ignores the management scheme and finishes transferring the data. The second case is when the S5933 is configured for add-on initiated bus mastering with transfer counts disabled. In this situation, the FIFO management scheme must be set to request the PCI bus when one or more locations are full. AMREN and AMWEN may be used to implement a specific FIFO management scheme.

10.1.4.5 FIFO Bus Master Cycle Priority

In many applications, the FIFO is used as a PCI initiator performing both PCI reads and writes. This requires a priority scheme be implemented. What happens if the FIFO condition for initiating a PCI read and a PCI write are both met?

Bits D12 and D8 in the Bus Master Control/Status Register (MCSR) control the read and write cycle priority, respectively. If these bits are both set or both clear, priority alternates, beginning with a read cycle. If the read priority is set and the write priority is clear, read cycles take priority. If the write priority is set and the read priority is clear, write cycles take priority. Priority arbitration is only done when neither FIFO has control of the PCI bus (the PCI to add-on FIFO would never interrupt an add-on to PCI FIFO transfer).

10.1.4.6 FIFO Generated Bus Master Interrupts

Interrupts may be generated under certain conditions from the FIFO. If PCI initiated bus mastering is used, INTA# is generated to the PCI interface. If add-on initiated bus mastering is used, IRQ# is generated to the add-on interface. Interrupts may be disabled.

FIFO Interrupts may be generated from one or more of the following during bus mastering: read transfer count reaches zero, write transfer count reaches zero, or an error occurs during bus mastering. Error conditions include a target or master abort on the PCI bus. Interrupts on PCI error conditions are only enabled if one or both of the transfer count interrupts are enabled.

The Add-on Interrupt Control/Status Register (AINT) or the Interrupt Control Status Register (INTCSR) indicates the interrupt source. The interrupt service routine may read these registers to determine what action is required. As mailboxes are also capable of generating interrupts, this must also be considered in the service routine. Interrupts are also cleared through these registers.

10.2 BUS INTERFACE

The S5933 FIFO may be accessed from the add-on interface or the PCI interface. Add-on FIFO control and status signals allow a simple interface to the FIFO with either an add-on CPU or programmable logic. The following section describes the PCI and add-on interface behavior and hardware interface.

10.2.1 FIFO PCI Interface (Target Mode)

The S5933 FIFO may act as a standard PCI target. FIFO empty/full status may be determined by the PCI initiator by reading the status bits in the PCI Bus Master Control/Status Register (MCSR).

The FIFO occupies a single 32-bit register location within the PCI Operation Registers. **A PCI initiator may not perform burst accesses on the FIFO**. Each data phase of a burst causes the PCI initiator to increment its address counter (even though only the first address is driven at the beginning of the burst). The initiator keeps track of the current address in case a disconnect occurs. This allows the initiator to continue the burst from where the disconnect occurred. If the S5933 FIFO initiated a disconnect during a PCI burst to the FIFO register, the burst would be resumed at an address other than the FIFO location (because the initiator address counter has incremented). The S5933 always signals a disconnect if a burst to any PCI Operation Register is attempted.

Because the PCI to add-on FIFO and the add-on to PCI FIFO occupy a single location within the PCI and Add-on Operation Registers, which FIFO is accessed is determined by whether the access is a read or write. This means that once data is written into the FIFO, the value written cannot be read back.

For PCI reads from the add-on to PCI FIFO, the S5933 asserts TRDY# and completes the PCI cycle (Figure 10-3). If the PCI bus attempts to read an empty FIFO, the S5933 immediately issues a disconnect with retry (Figure 10-4). The add-on to PCI FIFO status indicators change one PCI clock after a PCI read.

For PCI writes to the PCI to add-on, the S5933 asserts TRDY# and completes the PCI cycle (Figure 10-5). If the PCI bus attempts to write a full FIFO, the S5933 immediately issues a disconnect with retry (Figure 10-6). The PCI to add-on FIFO status indicators change one PCI clock after a PCI write.

10.2.2 FIFO PCI Interface (Initiator Mode)

The S5933 can act as an initiator on the PCI bus. This allows the device to gain control of the PCI bus to transfer data to or from the FIFO. Internal address and transfer count registers control the number of PCI transfers and the locations of the transfers. The following paragraphs assume the proper registers and bits are programmed to enable bus mastering (see section 10.3.3).

PCI read and write transfers from the S5933 are very similar. The FIFO management scheme (section 10.1.4.4) determines when the S5933 asserts its PCI bus request (REQ#). When bus grant (GNT#) is returned, the device begins running PCI cycles. Once the S5933 controls the bus, the FIFO management scheme is not important. It only determines when PCI bus control is initially requested. PCI bus reads and writes are always performed as bursts by the S5933, if possible.

Figure 10-3. PCI Read from a Full S5933 FIFO
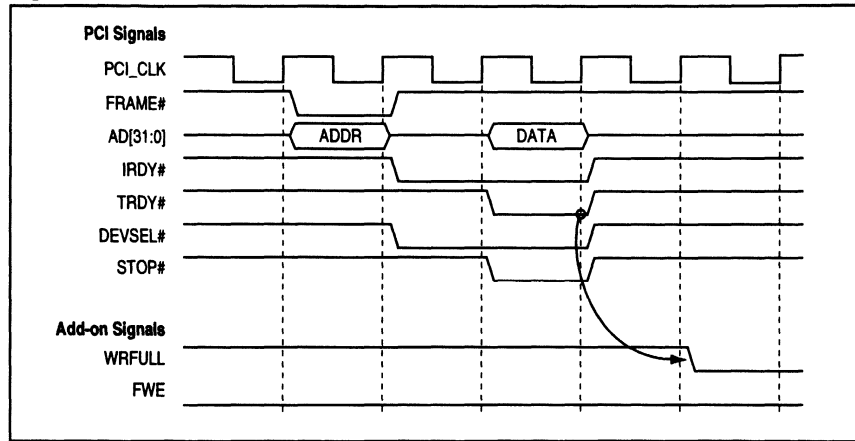
Figure 10-4. PCI Read from an Empty S5933 FIFO (Target Disconnect)

10
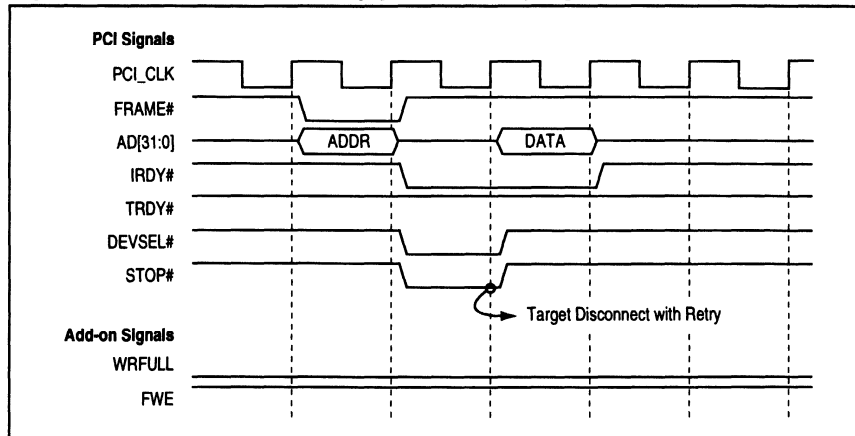
Figure 10-5. PCI Write to an Empty S5933 FIFO
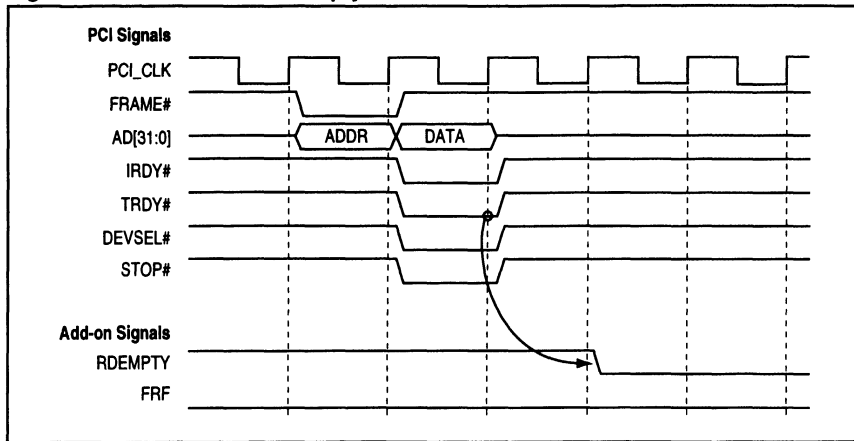
Figure 10-6. PCI Write to a Full S5933 FIFO (Target Disconnect)
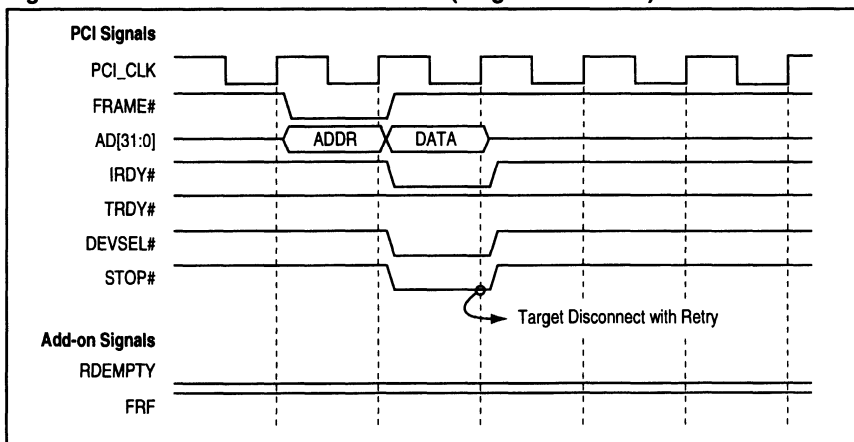
10.2.2.1 FIFO PCI Bus Master Reads

For PCI read transfers (filling the PCI to add-on FIFO), read cycles are performed until one of the following occurs:

- Bus Master Read Transfer Count Register (MRTC), if used, reaches zero
- The PCI to add-on FIFO is full
- GNT# is removed by the PCI bus arbiter
- AMREN is deasserted (See section 10.1.4.1)

If the transfer count is not zero, GNT# remains asserted, and AMREN is asserted, the FIFO continues to read data from the PCI bus until there are no empty locations in the PCI to add-on FIFO. If the add-on can empty the FIFO as quickly as it can be filled from the PCI bus, very long bursts are possible. The S5933 deasserts REQ# when it completes the access to fill the last location in the FIFO. Once REQ# is deasserted, it will not be reasserted until the FIFO management condition is met.

10.2.2.2 FIFO PCI Bus Master Writes

For PCI write transfers (emptying the add-on to PCI FIFO), write cycles are performed until one of the following occurs:

- Bus Master Write Transfer Count Register (MWTC), if used, reaches zero
- The add-on to PCI FIFO is empty
- GNT# is removed by the PCI bus arbiter
- AMWEN is deasserted (See section 10.1.4.1)

If the transfer count is not zero, GNT# remains asserted, and AMWEN is asserted, The FIFO continues to write data to the PCI bus until there are is no data in the add-on to PCI FIFO. If the add-on can fill the FIFO as quickly as it can be emptied to the PCI bus, very long bursts are possible. The S5933 deasserts REQ# when it completes the access to transfer the last data in the FIFO. Once REQ# is deasserted, it will not be reasserted until the FIFO management condition is met.

10.2.3 Add-on Bus Interface

The FIFO register may be accessed in two ways from the add-on interface. It can be accessed through normal register accesses or directly with the RDFIFO# and WRFIFO# inputs. In addition, the FIFO register can be accessed with synchronous or asynchronous to BPCLK, depending on the S5933 configuration. The add-on interface also supports datapaths which are not 32-bits. The method used to access the FIFO from the add-on interface is independent of whether the FIFO is a PCI PCI target or a PCI initiator.

10.2.3.1 Add-on FIFO Register Accesses

The FIFO may be accessed from the add-on interface through the Add-on FIFO Port Register (AFIFO) read or write. This is offset 20h in the Add-on Operation Registers. Depending on the device configuration, this register can be accessed either synchronous BPCLK or asynchronous to BPCLK. To access the FIFO as a normal Add-on Operation Register, ADR[6:2], BE[3:0]#, SELECT#, and RD# or WR# are required. The major differences between synchronous and asynchronous modes are when the FIFO pointers advance and the ability to perform burst accesses. The following examples are shown for add-on FIFO reads. FIFO write waveforms are shown in Chapter 12.

Figure 10-7 shows an asynchronous FIFO register read. SELECT# must meet setup and hold times relative to the rising edge of RD#. RD# and SELECT# both asserted enables the DQ outputs, and the first data location in the FIFO is driven onto the bus. The FIFO address and the byte enables must be valid before valid data is driven onto the DQ bus. Data remains valid as long as the address, byte enables, SELECT# and RD# are asserted. Deasserting RD# or SELECT# causes the data bus to float. The rising edge of RD# causes the FIFO pointer to advance. The status outputs are updated to reflect the FIFO condition after it advances.

10

Figure 10-7. Asynchronous FIFO Register Read Access Example
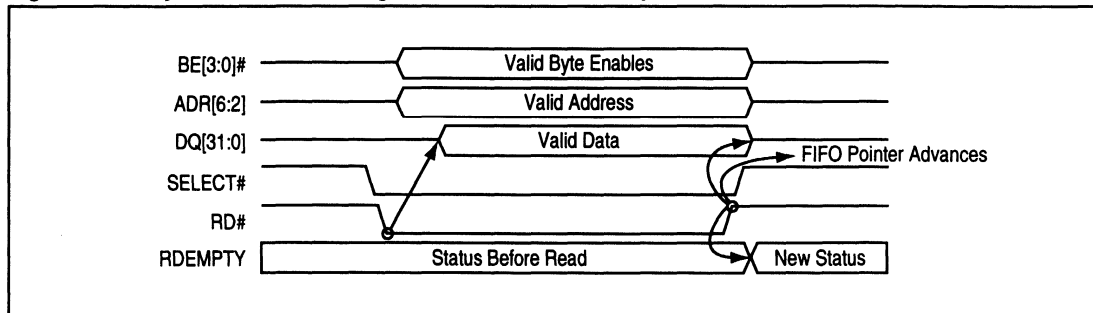
When the last location in the PCI to add-on FIFO is read by the add-on, the FIFO pointer does not change. If another read is performed before more data enters the FIFO, the data is undefined. When a write to a full add-on to PCI FIFO is attempted, nothing happens. No FIFO data is overwritten and the FIFO pointers are not changed.

Figure 10-8 shows a synchronous FIFO register burst access. SELECT# must meet setup and hold times relative to the rising edge of BPCLK. RD# and SELECT# both asserted enables the DQ outputs, and the first data location (data 0) in the FIFO is driven on to the bus. The FIFO address and the byte enables must be valid before valid data is driven onto the DQ bus. Data 0 remains valid until the next rising edge of BPCLK. The rising edge of BPCLK causes the FIFO pointer to advance to the next location (data 1). The next rising edge of BPCLK also advances the FIFO pointer to the next location (data 2). The status outputs reflect the FIFO condition after it advances, and are updated off of the rising edge of BPCLK. When RD# or SELECT# is deasserted, the DQ bus floats. The next time a valid FIFO access occurs and RD# and SELECT# are asserted, data 2 is presented on the DQ bus (as there was no BPCLK edge to advance the FIFO).

10.2.3.2 Add-on FIFO Direct Access Mode

Instead of generating an address, byte enables, SELECT# and a RD# or WR# strobe for every FIFO access, the S5933 allows a simple, direct access mode. Using RDFIFO# and WRFIFO# is functionally identical to performing a standard AFIFO Port Register access, but requires less logic to implement. Accesses to the FIFO register using the direct access signals are always 32-bits wide. The only exception to this is when the MODE pin is configured for 16-bit operation. In this situation, all accesses are 16-bits wide (see Section 10.2.3.5). The RD# and WR# inputs must be inactive when RDFIFO# or WRFIFO# is active. The ADR[6:2] and BE[3:0]# inputs are ignored.

Depending on the device configuration, RDFIFO# and WRFIFO# can act as clocks for data or enables with BPCLK acting as the clock. A Synchronous interface allows higher data rates, and an asynchronous interface is better for slow add-on logic which may require wait states. The major difference between the synchronous and asynchronous modes is when the FIFO advances.

Figure 10-9 shows an asynchronous FIFO register direct access using RDFIFO#. The first location in the FIFO is driven onto the bus when RDFIFO# is asserted. Data remains valid as long as RDFIFO# is asserted. The rising edge of RDFIFO# causes the data bus to float and acts as the clock causing the FIFO pointer to advance. The status outputs reflect the FIFO condition after it advances.

Figure 10-10 shows a synchronous FIFO register direct burst access using RDFIFO#. RDFIFO# acts as an enable and the first data location (data 0) in the FIFO is driven on to the bus when RDFIFO# is asserted. Data 0 remains valid until the next rising edge of BPCLK. The rising edge of BPCLK causes the FIFO pointer to advance to the next location (data 1). The next rising edge of BPCLK advances the FIFO pointer to the next location (data 2). The status outputs reflect the FIFO condition after it advances, and are updated off of the rising edge of BPCLK. When RDFIFO# is deasserted, the DQ bus floats. The next time RDFIFO# is asserted, data 2 is presented on the DQ bus (as there was no BPCLK edge to advance the FIFO).

A synchronous FIFO interface has the advantage of allowing data to be accessed more quickly (in bursts) by the add-on. As a target, if a full S5933 FIFO is written (or an empty FIFO is read) by a PCI initiator, the S5933 requests a retry. The faster the add-on interface can empty (or fill) the FIFO, the less often retries occur. With the S5933 as a PCI initiator, a

Figure 10-8. Synchronous FIFO Register Burst Read Access Example
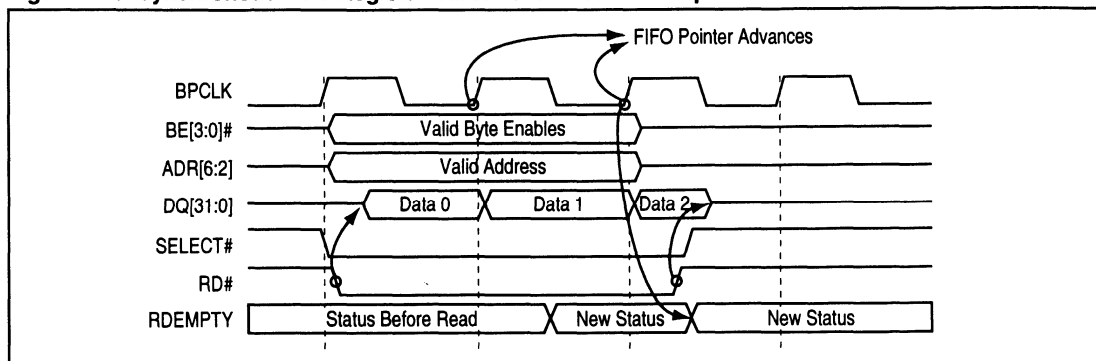
similar situation occurs. Not emptying or filling the FIFO quickly enough results in the S5933 giving up control of the PCI bus. Higher PCI bus data transfer rates are possible through the FIFO with a synchronous interface.

When the last location in the PCI to add-on FIFO is read by the add-on, the FIFO pointer does not change. If another read is performed before more data enters the FIFO, the previous data is driven. When a write to a full add-on to PCI FIFO is attempted, nothing happens. No FIFO data is overwritten and the FIFO pointers are not changed. This behavior is the same whether the FIFO is accessed using the direct access inputs or normal Operation Register accesses.

10.2.3.3   Additional Status/Control Signals for Add-on Initiated Bus Mastering

If a serial non-volatile memory is used to configure the S5933, and the device is configured for add-on initiated bus mastering, two additional FIFO status signals and four additional control signals are available to the add-on interface. The FRF and FWE outputs provide additional FIFO status information. Inputs FRC#, FWC#, AMREN, and AMWEN provide additional FIFO control. Applications may use these signals to monitor/control FIFO flags and PCI bus requests. These new signals are some of the lines that were used for byte-wide nvram interface, but now are reconfigured. The reconfigured lines are as follows:

Outputs:

E_ADDR (15)   FRF
FIFO Read Full: Indicates that the PCI to add-on FIFO is full.

E_ADDR (14)   FWE
FIFO Write Empty: Indicates that the add-on to PCI FIFO is empty.

Inputs:

EQ (7)   AMWEN
Add-on bus Mastering Write ENable: This input is driven high to enable bus master writes.

EQ (6)   AMREN
Add-on bus Mastering Read ENable: This input is driven high to enable bus master reads.

EQ (5)   FRC#
FIFO Read Clear: This line is driven low to clear the PCI to add-on FIFO.

EQ (4)   FWC#
FIFO Write Clear: This line is driven low to clear the add-on to PCI FIFO.

FRF (PCI to add-on FIFO full) and FWE (add-on to PCI FIFO empty) supplement the RDEMPTY and WRFULL status indicators. These additional status outputs provide additional FIFO status information for add-on FIFO control logic.

10

Figure 10-9. Asynchronous FIFO Register RDFIFO# Access Example
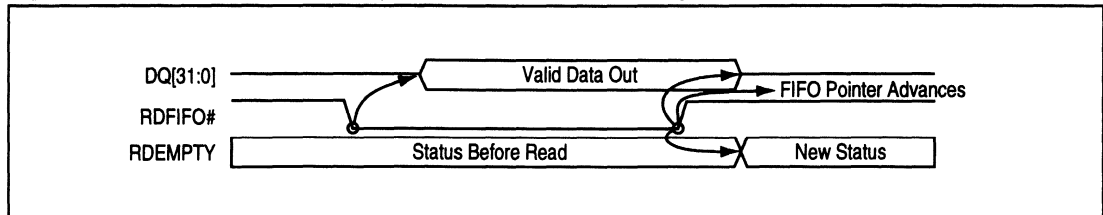
Figure 10-10. Synchronous FIFO Register Burst RDFIFO# Access Example
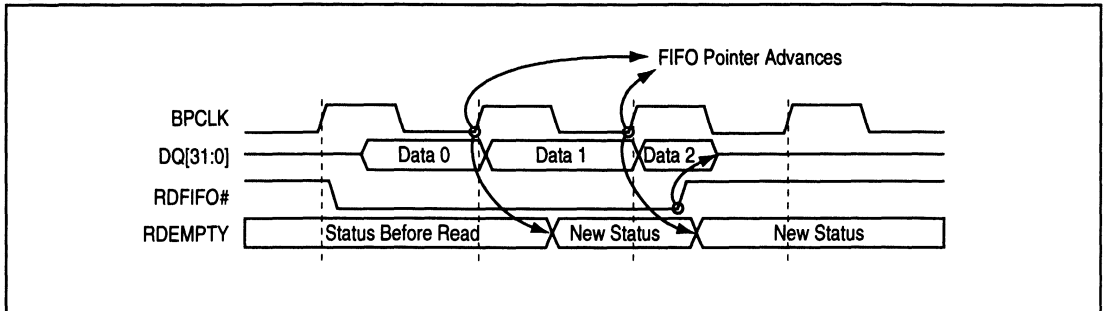
The FRC# and FWC# inputs allow add-on logic to reset the PCI to add-on or add-on to PCI FIFO flags. The FIFO flags can always be reset with software through the Add-on General Control/Status Register (AGCSTS) or the Bus Master Control/Status Register (MCSR), but these hardware inputs are useful for designs which do no implement a CPU on the add-on card. Asserting the FRC# input resets the PCI to add-on FIFO. Asserting the FWC# input resets the add-on to PCI FIFO.

The AMREN and AMWEN inputs allow add-on logic to individually enable and disable bus mastering for the PCI to add-on and add-on to PCI FIFO. These inputs override the Bus Master Control/Status Register (MCSR) bus master enable bits. The S5933 may request the PCI bus for the PCI to add-on FIFO when AMREN is asserted and may request the PCI bus for the add-on to PCI FIFO when AMWEN is asserted. If AMREN or AMWEN is deasserted, the S5933 removes its PCI bus request and gives up control of the bus.

AMREN and AMWEN are useful for add-ons with external FIFOs cascaded into the S5933. For PCI bus master write operations, the entire S5933 add-on to PCI FIFO and the external FIFO may be filled before enabling bus mastering, providing a single long burst write rather than numerous short bursts.

In some applications, the amount of data to be transferred is not known. During read operations, the S5933, attempting to fill its PCI to add-on FIFO, may access up to eight memory locations beyond what is required by the add-on before it stops. In this situation, AMREN can be deasserted to disable PCI reads, and then FRC# can be asserted to flush the unwanted data from the FIFO.

10.2.3.4 FIFO Generated Add-on Interrupts

For add-on initiated bus mastering, the S5933 may be configured to generate interrupts to the add-on interface for the following situations:

- Read transfer count reaches zero

- Write transfer count reaches zero

- An error occurred during the bus master transaction

The interrupt is posted to the add-on interface with the IRQ# output. A high-to-low transition on this output indicates an interrupt condition. Because there is a single interrupt output and multiple interrupt conditions, the Add-on Interrupt Control/Status Register (AINT) must be read to determine the interrupt source. This register is also used to clear the interrupt, returning IRQ# to its high state. If mailbox interrupts are also used, this must be considered in the interrupt service routine.

10.2.3.5 8-Bit and 16-Bit FIFO Add-on Interfaces

The S5933 FIFO may also be used to transfer data between the PCI bus and 8-bit or 16-bit add-on interfaces. This can be done using FIFO advance conditions or the S5933 MODE input pin.

The FIFO may be used as an 8-bit or 16-bit wide FIFO. To use the FIFO as an 8-bit interface, the advance condition should be set for byte 0 (no data is transferred in the upper 3 bytes). To use the FIFO as a 16-bit interface, the advance condition should be set for byte 1 (no data is transferred in the upper 2 bytes). This allows a simple add-on bus interface, but it has the disadvantage of not efficiently utilizing the PCI bus bandwidth because the host is forced to perform 8-bit or 16-bit accesses to the FIFO on the PCI bus. This is the only way to communicate with an 8-bit add-on through the FIFO without additional logic to steer byte lanes on the add-on data bus. Pass-thru mode is more suited to 8-bit add-on interfaces.

Implementing a 16-bit wide FIFO is a reasonable solution, but to avoid wasting PCI bus bandwidth, the best method is to allow the PCI bus and the FIFO to operate with 32-bit data. The S5933 can assemble or disassemble 32-bit quantities for the add-on interface. This is possible through the MODE pin. When MODE is low, the add-on data bus is 32-bits. When MODE is high, the add-on data bus is 16-bits. When MODE is configured for 16-bit operation, BE3# becomes ADR1.

With the FIFO direct access signals (RDFIFO# and WRFIFO#), the MODE pin must reflect the actual add-on data bus width. With MODE = 16-bits, the S5933 automatically takes two consecutive, 16-bit add-on writes to the FIFO and assembles a 32-bit value. FIFO reads operate in the same manner. Two consecutive add-on reads empty the 32-bit FIFO register. The 16-bit data bus is internally steered to the lower and upper words of the 32-bit FIFO register.

One consideration needs to be taken when using the FIFO direct access signals and letting the S5933 do byte lane steering internally. The default condition used to advance the FIFO is byte 0. This must be changed to byte 2 or 3. When MODE is configured for a 16-bit add-on bus, the first 16-bit cycle to the FIFO always accesses the low 16-bits. If the FIFO advance condition is left at byte 0, the FIFO advances after the first 16-bit cycle and the data in the upper 16-bits is directed to the next FIFO location, shifting the data.

Some applications hold the RDFIFO# and WRFIFO# inputs active for a synchronous interface. In 16-bit mode, designs must avoid writing to a full FIFO. The data for the write is lost, but the internal mechanism to direct the 16-bit external data bus to the upper 16-bits of the FIFO register is triggered. This creates a situation where the FIFO is out of step. The next 16-bit FIFO write is directed to the upper 16-bits of the FIFO, and the FIFO advances incorrectly. The WRFULL output should be used to gate the WRFIFO# input to avoid this situation. A similar problem can occur if add-on logic attempts to read an empty FIFO in 16-bit mode. RDEMPTY should be used to gate the RDFIFO# input to avoid problems with the FIFO getting out of step. In 32-bit mode (MODE = low), these situations do not occur.

If FIFO accesses are done without the direct access signals with MODE configured for 16-bits (using ADR, SELECT#, etc.), external hardware must toggle ADR1 between consecutive 16-bit bus cycles. The FIFO advance condition must be set to correspond to the order the application accesses the upper and lower words in the FIFO register.

10.3 CONFIGURATION

The FIFO configuration takes place during initialization and during operation. During initialization, the FIFO hardware interface method and bus master register access rights are defined. During operation, FIFO advance conditions, endian conversion, and bus mastering capabilities are defined. The following section describes the bits and registers which are involved with controlling and monitoring FIFO operation.

10.3.1 FIFO Setup During Initialization

Location 45h in an external non-volatile memory may be used to configure the S5933 FIFO during initialization. If no external non-volatile memory is used, the S5933 defaults to PCI initiated bus master transfers with asynchronous operation for FIFO accesses.

The value of bit 7 in location 45h determines if the address and transfer count registers used in bus mastering are accessible from the PCI bus or from the add-on bus. Once the configuration information is downloaded from non-volatile memory after reset, the bus mastering initialization method can not be changed. Access to the bus master address and transfer count registers cannot be alternated between the PCI bus and the add-on interface during operation.

Bits 6 and 5 in location 45h determine if FIFO register accesses using the RDFIFO#, WRFIFO#, RD# and WR# inputs operate asynchronous or synchronous to BPCLK. For asynchronous operation, RDFIFO#, WRFIFO#, RD# and WR# operate as clocks for data. For synchronous operation, RDFIFO#, WRFIFO#, RD# and WR# operate as enables, using BPCLK to clock data. Synchronous operation allows higher data transfer rates.

Location 45h Configuration Bits

Bit 7   Bus Master Register Access

0   Address and transfer count registers only accessible from the add-on interface

1   Address and transfer count registers only accessible from the PCI interface (default)

Bit 6   RDFIFO#, RD# Operation

0   Synchronous Mode - RDFIFO# and RD# functions as enables

1   Asynchronous Mode - RDFIFO# and RD# functions as clocks (default)

Bit 5   WRFIFO#, WR# Operation

0   Synchronous Mode - WRFIFO# and WR# functions as enables

1   Asynchronous Mode - WRFIFO# and WR# functions as clocks (default)

10.3.2 FIFO Status and Control Bits

The FIFO status can be monitored and the FIFO operation controlled from the PCI Operation Registers and/or the Add-on Operation Registers. The FIFO register resides at offset 20h in the PCI and Add-on Operation Registers.

The Bus Master Control/Status (MCSR) PCI Operation register allows a PCI host to monitor FIFO activity and control FIFO operation. Reset controls allow the PCI to add-on FIFO and add-on to PCI FIFO flags to be reset (individually). Status bits indicate if the PCI to add-on FIFO is empty, has four or more open spaces, or is full. Status bits also indicate if the add-on to PCI FIFO is empty, has four or more full locations or is full. Finally, FIFO PCI bus mastering is monitored/controlled though this register (see Section 10.3.3).

10

The Add-on General Control/Status (AGCSTS) Add-on Operation Register allows an add-on CPU to monitor FIFO activity and control FIFO operation. Reset controls allow the PCI to add-on FIFO and add-on to PCI FIFO flags to be reset (individually). Status bits indicate if the PCI to add-on FIFO is empty, has four or more open spaces, or is full. Status bits also indicate if the add-on to PCI is empty, has four or more full spaces or is full. FIFO bus mastering status may be monitored through this register, but all bus master configuration is through the MCSR PCI Operation Register (see Section 10.3.3).

10.3.3 PCI Initiated FIFO Bus Mastering Setup

For PCI initiated bus mastering, the PCI host sets up the S5933 to perform bus master transfers. The following tasks must be completed to setup FIFO bus mastering:

1) Define interrupt capabilities. The PCI to add-on and/or add-on to PCI FIFO can generate a PCI interrupt to the host when the transfer count reaches zero.

INTCSR	Bit 15	Enable Interrupt on read transfer count equal zero
INTCSR	Bit 14	Enable Interrupt on write transfer count equal zero

2) Reset FIFO flags. This may not be necessary, but if the state of the FIFO flags is not known, they should be initialized.

MCSR	Bit 26	Reset add-on to PCI FIFO flags
MCSR	Bit 25	Reset PCI to add-on FIFO flags

3) Define FIFO management scheme. These bits define what FIFO condition must exist for the PCI bus request (REQ#) to be asserted by the S5933.

MCSR	Bit 13	PCI to add-on FIFO management scheme
MCSR	Bit 9	Add-on to PCI FIFO management scheme

4) Define PCI to add-on and add-on to PCI FIFO priority. These bits determine which FIFO has priority if both meet the defined condition to request the PCI bus. If these bits are the same, priority alternates, with read accesses occurring first.

MCSR	Bit 12	Read vs. write priority
MCSR	Bit 8	Write vs. read priority

5) Define transfer source/destination address. These registers are written with the first address that is to be accessed by the S5933. These address registers are updated after each access to indicate the next address to be accessed. Transfers must start on DWORD boundaries.

MWAR	All	Bus master write address
MRAR	All	Bus master read address

6) Define transfer byte counts. These registers are written with the number of bytes to be transferred. The transfer count does not have to be a multiple of four bytes. These registers are updated after each transfer to reflect the number of bytes remaining to be transferred.

MWTC	All	Write transfer byte count
MRTC	All	Read transfer byte count

7) Enable Bus Mastering. Once steps 1-6 are completed, the FIFO may operate as a PCI bus master. Read and write bus master operation may be independently enabled or disabled.

MCSR	Bit 14	Enable PCI to add-on FIFO bus mastering
MCSR	Bit 10	Enable add-on to PCI FIFO bus mastering

The order of the tasks listed above is not particularly important. It is recommended that bus mastering be enabled as the last step. Some applications may choose to leave bus mastering enabled and start transfers by writing a non-zero value to the transfer count registers. This also works, provided the entire transfer count is written in a single access. As a number of the configuration bits and the two enable bits are all in the MCSR register, it may be most efficient for the FIFO configuration bits to be set with the same register access that enables bus mastering.

If interrupts are enabled, a host interrupt service routine is also required. The service routine determines the source of the interrupt and resets the interrupt. As mailbox registers may also be configured to generate interrupts, the exact source of the interrupt is indicated in the PCI Interrupt Control/Status Register (INTCSR). Typically, the interrupt service routine is used to setup the next transfer by writing new addresses and transfer counts, but some applications may also require other actions. If read transfer or write transfer complete interrupts are enabled, master and target abort interrupts are automatically enabled. These indicate a transfer error has occurred. Writing a one to these bits clears the corresponding interrupt.

INTCSR	Bit 21	Target abort caused interrupt
INTCSR	Bit 20	Master abort caused interrupt
INTCSR	Bit 19	Read transfer complete caused interrupt
INTCSR	Bit 18	Write transfer complete caused interrupt

10.3.4 Add-on Initiated FIFO Bus Mastering Setup

For add-on initiated bus mastering, the add-on sets up the S5933 to perform bus master transfers. The following tasks must be completed to setup FIFO bus mastering:

1) Define transfer count abilities. For add-on initiated bus mastering, transfer counts may be either enabled or disabled. Transfer counts for read and write operations cannot be individually enabled.

| AGCSTS | Bit 28 | Enable transfer count for read and write bus master transfers |

2) Define interrupt capabilities. The PCI to add-on and/or add-on to PCI FIFO can generate an interrupt to the add-on when the transfer count reaches zero (if transfer counts are enabled).

| AINT | Bit 15 | Enable interrupt on read transfer count equal zero |
| AINT | Bit 14 | Enable interrupt on write transfer count equal zero |

3) Reset FIFO flags. This may not be necessary, but if the state of the FIFO flags is not known, they should be initialized.

| AGCSTS | Bit 25 | Reset add-on to PCI FIFO flags |
| AGCSTS | Bit 26 | Reset PCI to add-on FIFO flags |

4) Define FIFO management scheme. These bits define what FIFO condition must exist for the PCI bus request (REQ#) to be asserted by the S5933. This must be programmed through the PCI interface.

| MCSR | Bit 13 | PCI to add-on FIFO management scheme |
| MCSR | Bit 9 | Add-on to PCI FIFO management scheme |

5) Define PCI to add-on and add-on to PCI FIFO priority. These bits determine which FIFO has priority if both meet the defined condition to request the PCI bus. If these bits are the same, priority alternates, with read accesses occurring first. This must be programmed through the PCI interface.

| MCSR | Bit 12 | Read vs. write priority |
| MCSR | Bit 8 | Write vs. read priority |

6) Define transfer source/destination address. These registers are written with the first address that is to be accessed by the S5933. These address registers are updated after each access to indicate the next address to be accessed. Transfers must start on DWORD boundaries.

| MWAR | All | Bus master write address |
| MRAR | All | Bus master read address |

7) Define transfer byte counts. These registers are written with the number of bytes to be transferred. The transfer count does not have to be a multiple of four bytes. These registers are updated after each transfer to reflect the number of bytes remaining to be transferred. If transfer counts are disabled, these registers do not need to be programmed.

| MWTC | All | Write transfer byte count |
| MRTC | All | Read transfer byte count |

10

8) Enable Bus Mastering. Once steps 1-7 are com-
    pleted, the FIFO may operate as a PCI bus mas-
    ter. Read and write bus master operation may be
    independently enabled or disabled. The AMREN
    and AMWEN inputs control bus master enabling
    for add-on initiated bus mastering. The MCSR
    bus master enable bits are ignored for add-on
    initiated bus mastering.

It is recommended that bus mastering be enabled as
the last step. Some applications may choose to leave
bus mastering enabled (AMREN and AMWEN as-
serted) and start transfers by writing a non-zero value
to the transfer count registers (if they are enabled).

If interrupts are enabled, an add-on CPU interrupt
service routine is also required. The service routine
determines the source of the interrupt and resets the
interrupt. As mailbox registers may also be config-
ured to generate interrupts, the exact source of the
interrupt is indicated in the Add-on Interrupt Control
Register (AINT). Typically, the interrupt service rou-
tine is used to setup the next transfer by writing new
addresses and transfer counts (if enabled), but some
applications may also require other actions. If read
transfer or write transfer complete interrupts are en-
abled, the master/target abort interrupt is automati-
cally enabled. These indicate a transfer error has
occurred. Writing a one to these bits clears the corre-
sponding interrupt.

AINT	Bit 21	Master/target abort caused interrupt
AINT	Bit 19	Read transfer complete caused interrupt
AINT	Bit 18	Write transfer complete caused interrupt

Pass-thru Description 11

11.0 PASS-THRU OVERVIEW

The S5933 provides a simple registered access port to the PCI bus. Using a handshaking protocol with add-on card logic, the PCI bus directly accesses resources on the add-on. The Pass-thru data transfer method is very useful for direct add-on memory access, or accessing registers within peripherals on an add-on board. Pass-thru operation requires an external nv memory boot device to define and configure the S5933 pass-thru regions.

The S5933 provides four user-configurable pass-thru regions. Each region corresponds to a PCI Configuration Base Address Register (BADR1-4). A region represents a block of address space (the block size is user-defined). Each block can be mapped into memory or I/O space. Memory mapped regions can request to be located below 1 MByte (Real Mode address space for a PC). Each region also has a configurable bus width for the add-on bus interface. An 8-, 16-, or 32-bit add-on interface may be selected, for use with a variety of add-on memory or peripheral devices.

Pass-thru features can be used only when the S5933 is a PCI target. As a target, the S5933 pass-thru mode supports single data transfers as well as burst transfers. When accessed with burst transfers, the S5933 supports data transfers at the full PCI bandwidth. The data transfer rate is only limited by the PCI initiator performing the access and the speed of the add-on logic.

11.1 FUNCTIONAL DESCRIPTION

To provide the PCI bus add-on with direct access to add-on resources, the S5933 has an internal Pass-thru Address Register (APTA), and a Pass-thru Data Register (APTD). These registers are connected to both the PCI bus interface and the add-on bus interface. This allows a PCI initiator to perform pass-thru writes (data transferred from the PCI bus to the add-on bus) or pass-thru reads (PCI bus requests data from the add-on bus). The S5933 pass-thru interface supports both single cycle (one data phase) and burst accesses (multiple data phases).

11.1.1 Pass-thru Transfers

Data transfers between the PCI bus and the add-on using the pass-thru interface are implemented with a handshaking scheme. If the PCI bus writes to an S5933 pass-thru region, add-on logic must read the data from the S5933 and store it on the add-on. If the PCI bus reads from a pass-thru region, add-on logic must write data to the S5933.

Some applications may require that an address be passed to the add-on for pass-thru accesses. For example, a 4 Kbyte pass-thru region on the PCI bus may correspond to a 4 Kbyte block of SRAM on the add-on card. If a PCI initiator accessed this region, the add-on would need to know the offset within the memory device to access. The Pass-thru Address Register (APTA) allows the add-on to access address information for the current PCI cycle. When the PCI bus performs burst accesses, the APTA register is updated by the S5933 to reflect the address of the current data phase.

For PCI writes to the add-on, the S5933 transfers the data from the PCI bus into the Pass-thru Data Register (APTD). The S5933 captures the data from the PCI bus when TRDY# is asserted. The PCI bus then becomes available for other transfers. When the pass-thru data register becomes full, the S5933 asserts the pass-thru status signals (Section 11.1.2) to indicate to the add-on that data is present. The add-on logic may read the data register and assert PTRDY# to indicate the current access is complete. Until the current access completes, the S5933 responds to further pass-thru accesses with retries.

For PCI reads from the add-on, the S5933 asserts the pass-thru status signals (Section 11.1.2) to indicate to the add-on that data is required. The add-on logic should write to the Pass-thru Data Register and assert PTRDY# to complete the access. The S5933 does not assert TRDY# to the PCI bus until PTRDY# is asserted by add-on logic. If the add-on cannot provide data quickly enough, the S5933 signals a retry to the PCI bus. This allows the PCI bus to perform other tasks, rather than waiting for a slow target. This situation is described in more detail in Section 11.2.1.3.

11

11.1.2 Pass-thru Status/Control Signals

The S5933 pass-thru registers are accessed using the standard add-on register access pins (see Section 8.1.1.2). The Pass-thru Address Register (APTA) can, optionally, be accessed using a single, direct access input, PTADR#. Pass-thru cycle status indicators are provided to control add-on logic based on the type of pass-thru access occurring (single cycle, burst, etc.). The following signals are provided for pass-thru operation:

Signal	Function
PTATN#	This output indicates a pass-thru access is occurring
PTBURST#	This output indicates the pass-thru access is a PCI burst access
PTNUM[1:0]	These outputs indicate which pass-thru region decoded the PCI address
PTBE[3:0]#	These outputs indicate which data bytes are valid (PCI writes), or requested (PCI reads)
PTWR	This output indicates if the pass-thru access is a PCI read or a write
PTADR#	When asserted, this input drives the Pass-thru Address Register contents onto the add-on data bus
PTRDY#	When asserted, this input indicates the current pass-thru transfer has been completed by the add-on
BPCLK	Buffered PCI bus clock output (to synchronize pass-thru data register accesses)

The operation of these signals is described in more detail in Section 11.2.2. Specific timing information is provided in Chapter 12.

11.1.3 Pass-thru Add-on Data Bus Sizing

Many applications require an 8-bit or 16-bit add-on bus interface. Pass-thru regions can be configured to support bus widths other than 32-bits. Each pass-thru region can be defined, during initialization, as 8-, 16-, or 32-bits. All of the regions do not need to be the same. This feature allows a simple interface to 8- and 16-bit add-on devices.

To support alternate add-on bus widths, the S5933 performs internal data bus steering. This allows the add-on interface to assemble and disassemble 32-bit PCI data using multiple add-on accesses to the Pass-thru Data Register (APTD). The add-on byte enable inputs (BE[3:0]#) are used to access the individual bytes or words within APTD.

11.2 BUS INTERFACE

The pass-thru interface on the S5933 is a PCI target-only function. Pass-thru operation allows PCI initiators to read or write resources on the add-on card. A PCI initiator may access the add-on with single data phase cycles or multiple data phase bursts.

The add-on interface implements pass-thru status and control signals used by logic to complete data transfers initiated by the PCI bus. The pass-thru interface is designed to allow add-on logic to function without knowledge of PCI bus activity. Add-on logic only needs to react to the pass-thru status outputs. The S5933 PCI interface independently interacts with the PCI initiator to control data flow between the devices.

The following sections describe the PCI and add-on bus interfaces. The PCI interface description provides a basic overview of how the S5933 interacts with the PCI bus, and may be useful in system debugging. The add-on interface description indicates functions required by add-on logic and details the pass-thru handshaking protocol.

11.2.1 PCI Bus Interface

The S5933 decodes all PCI bus cycle addresses. If the address associated with the current cycle is to one of S5933 pass-thru regions, DEVSEL# is asserted. If the pass-thru logic is currently idle (not busy finishing a previous pass-thru operation), the bus cycle type is decoded and the add-on pass-thru status outputs are set to initiate a transfer on the add-on side. If the pass-thru logic is currently busy completing a previous access, the S5933 signals a retry to PCI initiator.

The following sections describe the behavior of the PCI interface for pass-thru accesses to the S5933. Single cycle accesses, burst accesses, and target-initiated retries are detailed.

11.2.1.1 PCI Pass-thru Single Cycle Accesses

Single cycle transfers are the simplest PCI bus transaction. Single cycle transfers have an address phase and a single data phase. The PCI bus transaction starts when an initiator drives address and command information onto the PCI bus and asserts FRAME#. The initiator always deasserts frame before the last data phase. For single cycle transfers, FRAME# is only asserted during the address phase (indicating the first data phase is also the last).

When the S5933 sees FRAME# asserted, it samples the address and command information to determine if the bus transaction is intended for it. If the address is within one of the defined pass-thru regions, the S5933 accepts the transfer (assert DEVSEL#), and stores the PCI address in the Pass-thru Address Register (APTA).

For pass-thru writes, the S5933 responds immediately (asserting TRDY#) and transfers the data from the PCI bus into the Pass-thru Data Register (APTD). The S5933 then indicates to the add-on interface that a pass-thru write is taking place and waits for add-on logic to read the APTD register and complete the transfer (assert PTRDY#). Once the S5933 has captured the data from the PCI bus, the transfer is finished from the PCI bus perspective, and the PCI bus becomes available for other transfers.

For pass-thru reads, the S5933 indicates to the add-on interface that a pass-thru read is taking place and waits for add-on logic to write the Pass-thru Data Register and complete the transfer (assert PTRDY#). The S5933 completes the cycle when data is written into the data register. If the add-on cannot complete the write quickly enough, the S5933 requests a retry from the initiator. See Section 11.2.1.3 for target-requested disconnect information.

11.2.1.2 PCI Pass-thru Burst Accesses

For PCI pass-thru burst accesses, the S5933 captures the PCI address and determines if it falls into one of the defined pass-thru regions. Accesses that fall into a pass-thru region are accepted by asserting DEVSEL#. The S5933 monitors FRAME# and IRDY# on the PCI bus to identify burst accesses. If the PCI initiator is performing a burst access, the pass-thru status indicators notify add-on logic.

For pass-thru burst writes, the S5933 responds immediately (asserting TRDY#). The S5933 transfers the first data phase of the burst into the Pass-thru Data Register (APTD), and stores the PCI address in the Pass-thru Address Register (APTA). The add-on interface completes the transfer and asserts PTRDY#. Every time PTRDY# is asserted by the add-on, the S5933 begins the next data phase. The next data phase is latched into the data register. For burst accesses, APTA is automatically incremented by the S5933 for each data phase.

For pass-thru burst reads, the S5933 claims the PCI cycle (asserting DEVSEL#). The request for data is passed on to add-on logic and the PCI address is stored in the APTA register. The add-on interface completes the transfer and asserts PTRDY#. The S5933 then drives the requested data on the PCI bus and asserts TRDY# to begin the next data phase. The APTA register is automatically incremented by the S5933 for each data phase.

11.2.1.3 PCI Retry Conditions

In some applications, add-on logic may not be able to respond to pass-thru accesses quickly. In this situation, the S5933 disconnects from the PCI bus, signaling a retry. This indicates that the initiator should try the access again at a later time. This allows other PCI cycles to be run while the logic on the slow target completes the pass-thru access. Ideally, when the initiator retries the access, the target has completed the access and can respond to the initiator.

With many devices, particularly memories, the first access takes longer than subsequent accesses (assuming they are sequential and not random). For this reason, the PCI specification allows 16 clocks to respond to the first data phase of a PCI cycle and 8 clocks for subsequent data phases (in the case of a burst) before a retry must be requested by the S5933.

The S5933 also requests a retry if an initiator attempts to burst past the end of a pass-thru region. The S5933 updates the Pass-thru Address Register (APTA) for each data phase during bursts, and if the updated address is not within the current pass-thru region, a retry is requested.

For example, a PCI system may map a 512 byte pass-thru memory region to 0DC000h to 0DC1FFh. A PCI initiator attempts a four DWORD burst with a starting address of 0DC1F8h. The first and second data phases complete (filling the DWORDs at 0DC1F8h and 0DC1FCh), but the third data phase causes the S5933 to request a retry. This forces the initiator to present the address 0DC200h on the PCI bus. If this address is part of another S5933 pass-thru region, the device accepts the access.

11.2.1.4 PCI Write Retries

When the S5933 requests a retry for a PCI pass-thru write, it indicates that the add-on is still completing a previous pass-thru write access. The Pass-thru Address and Data Register contents (APTA and APTD) are still required for the previous pass-thru operation and cannot be updated by the PCI interface until the access completes (the add-on asserts PTRDY#).

When the add-on is busy completing a pass-thru write, the S5933 requests an **immediate** retry for all pass-thru region accesses, allowing the PCI bus to perform other operations. PCI Operation Registers may be accessed while the add-on is still completing a pass-thru access. Only pass-thru region accesses receive retry requests.

11

11.2.1.5 PCI Read Retries

When the S5933 requests a retry for a PCI pass-thru read, it indicates that the add-on could not complete the read in the required time (Section 11.1.4.1). The pass-thru data cannot be read by the PCI interface until the add-on asserts PTRDY#, indicating the access is complete.

If the retry occurs after the add-on has completed the pass-thru operation by writing the appropriate data into the pass-thru data register and asserting PTRDY#, the S5933 asserts DEVSEL# and TRDY# to complete the PCI read. If the add-on still has not completed the pass-thru read, the S5933 waits for the required 16 clocks. If the add-on completes the access during this time, TRDY# is asserted and the access is finished. If the add-on cannot complete the access within 16 clocks, another retry is requested.

When the add-on is busy completing a pass-thru read, the S5933 requests an **immediate** retry for all pass-thru region accesses, except the region currently completing the previous access. This allows the PCI bus to perform other operations. The next access to the pass-thru region which initiated the retry must be to the **same address** which caused the retry. Another initiator accessing the same pass-thru region causes the S5933 to respond with the original initiator's data (for reads). S5933 PCI Operation Registers may be accessed while the add-on is still completing a pass-thru access. Only other pass-thru region accesses receive retry requests.

11.2.2 Add-on Bus Interface

The pass-thru address and data registers can be accessed as add-on operation registers. The interface to the pass-thru registers is described in Section 8.1.3. The pass-thru data register is updated on the rising edge of BPCLK. For this reason, all pass-thru inputs must be synchronous to BPCLK. In the following sections the add-on pass-thru interface is described for pass-thru single cycle accesses, burst accesses, target-requested retries, and when using 8-bit and 16-bit add-on data buses.

11.2.2.1 Single Cycle Pass-Thru Writes

A single cycle pass-thru write operation occurs when a PCI initiator writes a single value to a pass-thru region. PCI single cycle transfers consists of an address phase and one data phase. During the address phase of the PCI transfer, the S5933 stores the PCI address into the Pass-thru Address Register (APTA). If the S5933 determines that the address is within one of its defined pass-thru regions, it captures the PCI data into the Pass-thru Data Register (APTD).

Figure 11-1 shows a single cycle pass-thru write access (add-on read). The add-on must read the data stored in the APTD register and transfer it to its destination. Note: RD# may be asserted for multiple clocks to allow interfacing with slow add-on devices. Data remains valid until PTRDY# is asserted.

Figure 11-1. Single Cycle Pass-thru Write

PCI Write cycle completed
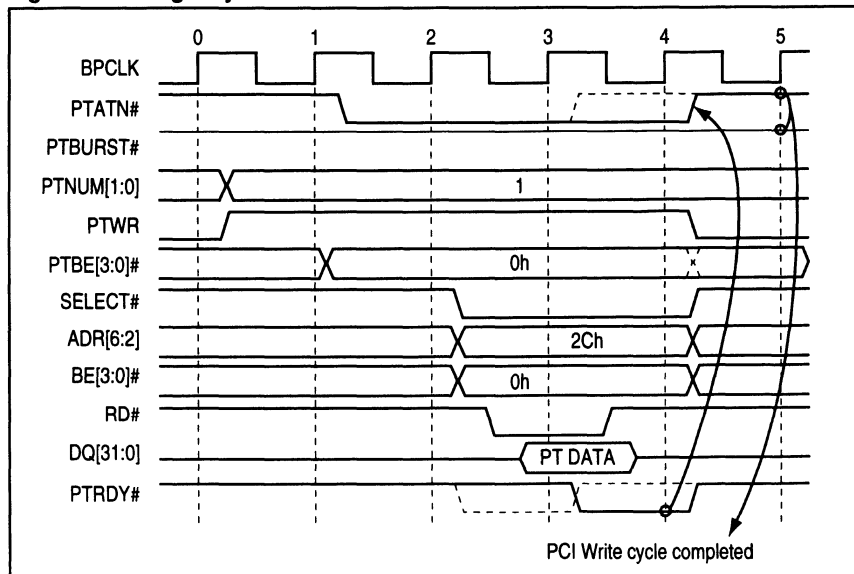
Clock 0: The PCI bus cycle address information is stored in the S5933 Pass-thru Address Register.

Clock 1: The PCI address is recognized as a write to pass-thru region 1. The PCI data is stored in the S5933 Pass-thru Data Register. PTATN# is asserted to indicate a pass-thru access is occurring.

Clock 2: Pass-thru status signals indicate what action is required by add-on logic. Pass-thru status outputs are valid when PTATN# is active and are sampled by the add-on at the rising edge of clock 2.

> PTBURST#   Deasserted. The access has a single data phase.
>
> PTNUM[1:0]   01. Indicates the PCI access is to pass-thru region 1.
>
> PTWR   Asserted. The pass-thru access is a write.
>
> PTBE[3:0]#   0h. Indicates the pass-thru access is 32-bits.

> SELECT#, address and byte enable inputs are driven to read the Pass-thru Data Register at offset 2Ch. DQ[31:0] are driven after RD# and SELECT# are asserted.

Clock 3: If PTRDY# is asserted at the rising edge of clock 3, PTATN# is immediately deasserted and the pass-thru access is completed at clock 4.

Clock 4: If add-on logic requires more time to read the Pass-thru Data Register (slower memory or peripherals), PTRDY# can be delayed, extending the cycle. With PTRDY# asserted at the rising edge of clock 4, PTATN# is deasserted and the pass-thru access is completed at clock 5.

Clock 5: PTATN# and PTBURST# deasserted at the rising edge of clock 5 indicates the pass-thru access is complete. The S5933 can accept new pass-thru accesses from the PCI bus at clock 6.

Figure 11-2 shows a single cycle pass-thru write using the pass-thru address information. This provides PCI cycle address information to select a specific address location within an add-on memory or peripheral. Add-on logic must latch the address for use during the data transfer. Typically, the entire 32-bit address is not required. The add-on may implement a scheme where only the required number of address bits are latched. It may also be useful to use the pass-thru region identifiers, PTNUM[1:0] as address lines. For example, pass-thru region 1 might be a 64K block of SRAM for data, while pass-thru region 2 might be 64K of SRAM for code storage (downloaded from the host during initialization). Using PTNUM0 as address line A16 allows two unique add-on memory regions to be defined.

Figure 11-2. Single Cycle Pass-thru Write with PTADR#
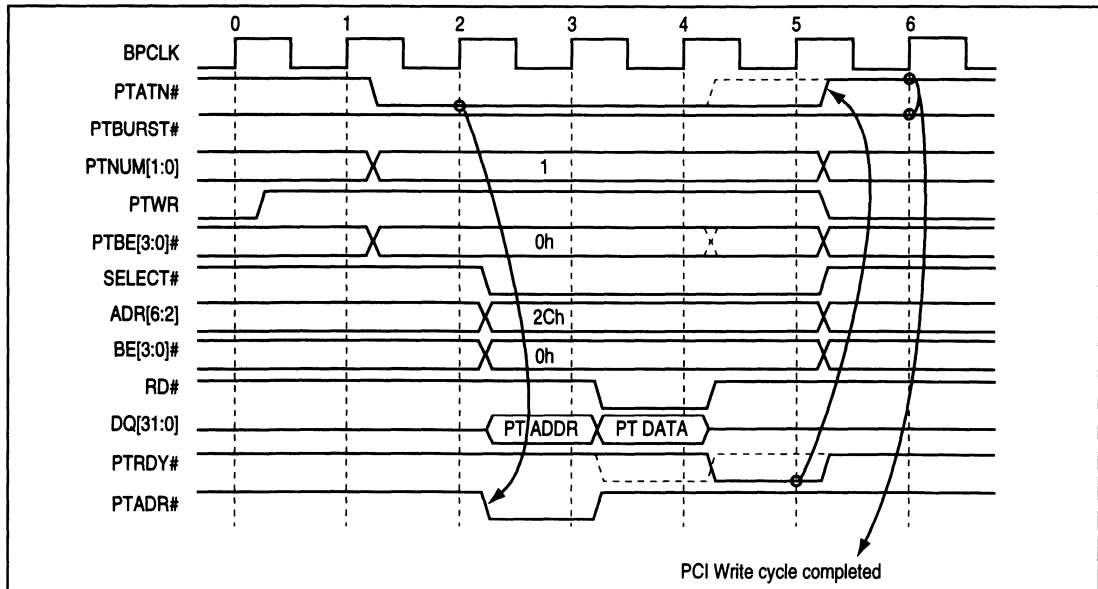
The add-on PTADR# input directly accesses the Pass-thru Address Register and drives the contents onto the data bus (no BPCLK rising edge is required). The byte enables, address, and SELECT# inputs are ignored when PTADR# is asserted. RD# and WR# must not be asserted when PTADR# is asserted.

Clock 0: The PCI bus cycle address is stored in the S5933 Pass-thru Address Register.

Clock 1: The PCI address is recognized as an access to pass-thru region 1. PCI data is stored in the S5933 Pass-thru Data Register. PTATN# is asserted to indicate a pass-thru access is occurring.

Clock 2: Pass-thru status signals indicate what action is required by add-on logic. Pass-thru status outputs are valid when PTATN# is active and are sampled by the add-on at the rising edge of clock 2.

PTBURST#	Deasserted. The access has a single data phase.
PTNUM[1:0]	01. Indicates the PCI access is to pass-thru region 1.
PTWR	Asserted. The pass-thru access is a write.
PTBE[3:0]#	0h. Indicate the pass-thru access is 32-bits.

The PTADR# input is asserted to read the Pass-thru Address Register. The byte enable, address, and SELECT# inputs are changed during this clock to select the Pass-thru Data Register during clock cycle 3.

Clock 3: SELECT#, byte enable, and the address inputs remain valid to read the Pass-thru Data Register at offset 2Ch. RD# is asserted to drive data register contents onto the DQ bus.

Clock 4: If PTRDY# is asserted at the rising edge of clock 4, PTATN# is immediately deasserted and the pass-thru access is completed at clock 5.

Clock 5: If add-on logic requires more time to read the Pass-thru Data Register (slower memory or peripherals), PTRDY# can be delayed, extending the cycle. PTRDY# asserted at the rising edge of clock 5 causes PTATN# to be immediately deasserted.

Clock 6: PTATN# and PTBURST# deasserted at the rising edge of clock 6 indicates the pass-thru access is complete. The S5933 can accept new pass-thru accesses from the PCI bus at clock 7.

11.2.2.2 Single Cycle Pass-Thru Reads

A single cycle pass-thru read operation occurs when a PCI initiator reads a single value from a pass-thru region. PCI single cycle transfers consists of an address phase and a one data phase. During the address phase of the PCI transfer, the S5933 stores the PCI address into the Pass-thru Address Register (APTA). If the S5933 determines that the address is within one of its defined pass-thru regions, it indicates to the add-on that a write to the Pass-thru Data Register (APTD) is required.

Figure 11-3 shows a single cycle pass-thru read access (add-on write) using PTADR#. The add-on reads data from a source on the add-on and writes it to the APTD register.

Clock 0: PCI address information is stored in the S5933 Pass-thru Address Register. The PCI cycle is recognized as an access to pass-thru region 1. PTATN# is asserted by the S5933 to indicate a pass-thru access is occurring.

Clock 1: Pass-thru status signals indicate what action is required by add-on logic. Pass-thru status outputs are valid when PTATN# is active and are sampled by the add-on at the rising edge of clock 1.

PTBURST#	Deasserted. The access has a single data phase.
PTNUM[1:0]	01. Indicates the PCI access was to pass-thru region 1.
PTWR	Deasserted. The pass-thru access is a read.
PTBE[3:0]#	0h. Indicate the pass-thru access is 32-bits.

The PTADR# input is asserted to read the Pass-thru Address Register. The byte enable, address, and SELECT# inputs are changed during this clock to select the Pass-thru Data Register during clock cycle 3.

Clock 2: This clock is required to avoid contention on the DQ bus. Time must be allowed after PTADR# is deasserted for the DQ outputs to float before add-on logic attempts to write to the Pass-thru Data Register.

Clock 3: SELECT#, byte enables, and the address inputs remain valid to write the Pass-thru Data Register at offset 2Ch. If WR# is asserted at the rising edge of clock 3, data on the DQ bus is latched into APTD.

If PTRDY# is asserted at the rising edge of clock 3, PTATN# is immediately deasserted and the pass-thru access is completed at clock 4.

Figure 11-3. Single Cycle Pass-thru Read with PTADR#

Data stored in pass-thru data register

PCI Read cycle completed
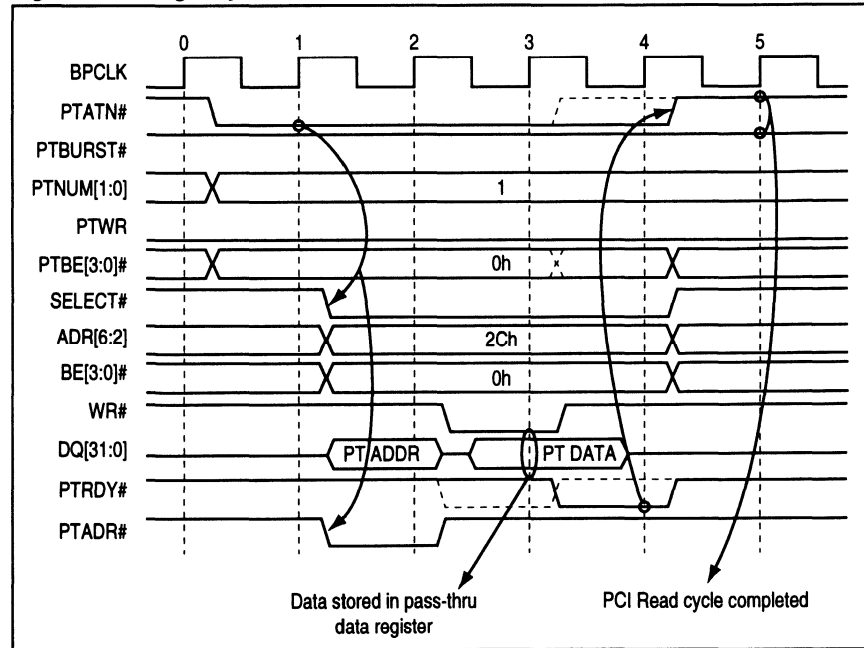
Clock 4: If add-on logic requires more time to write the pass-thru data register (slower memory or peripherals), PTRDY# can be delayed, extending the cycle. PTRDY# asserted at the rising edge of clock 4 causes PTATN# to be immediately deasserted and the pass-thru access is completed at clock 5.

Clock 5: PTATN# and PTBURST# deasserted at the rising edge of clock 5 indicates the pass-thru access is complete. The S5933 can accept new pass-thru accesses from the PCI bus at clock 6.

11.2.2.3 Pass-Thru Burst Writes

A pass-thru burst write operation occurs when a PCI initiator writes multiple values to a pass-thru region. A PCI burst cycle consists of an address phase and multiple data phases. During the address phase of the PCI transfer, the S5933 stores the PCI address into the Pass-thru Address Register (APTA). If the S5933 determines that the address is within one of its defined pass-thru regions, it captures the PCI data into the Pass-thru Data Register (APTD). After the add-on completes each read from the pass-thru data register (asserts PTRDY#), the next data phase is initiated.

Figure 11-4 shows a 6 data phase pass-thru burst write (add-on read). In this case, the add-on asserts PTADR# and then reads multiple data phases from the S5933. This works well for add-on logic which supports burst cycles. If the add-on logic does not support burst accesses, PTADR# may be pulsed before each data phase. The S5933 automatically increments the address in the APTA register during PCI burst cycles. In this example PTRDY# is always asserted, indicating add-on logic is capable of accepting data at a rate of one DWORD per clock cycle.

Clock 0: PCI address information is stored in the S5933 Pass-thru Address Register.

Clock 1: The PCI address is recognized as an access to pass-thru region 1. PCI data for the first data phase is stored in the S5933 Pass-thru Data Register. PTATN# is asserted by the S5933 to indicate a pass-thru access is occurring.

Clock 2: Pass-thru status signals indicate what action is required by add-on logic. Pass-thru status outputs are valid when PTATN# is active and are sampled by the add-on at the rising edge of clock 2.

11

Figure 11-4. Pass-thru Burst Write
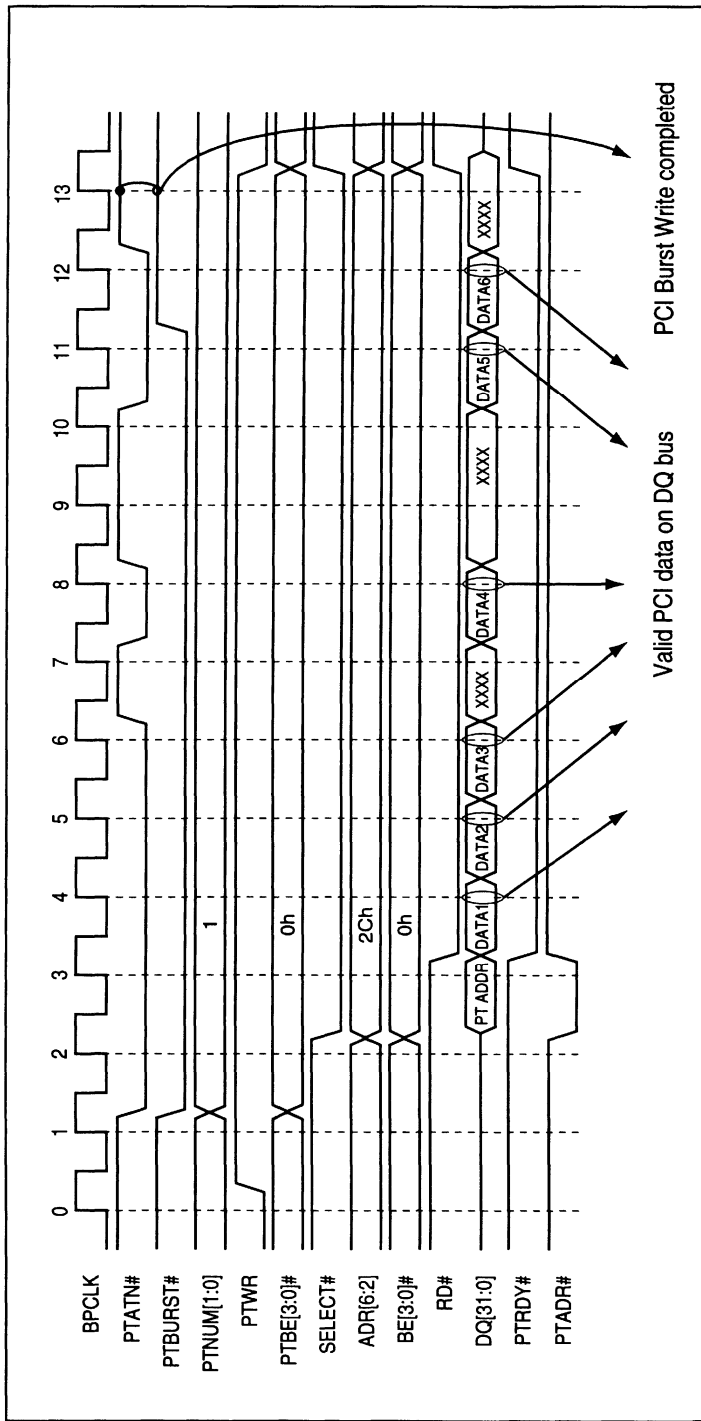
PTBURST#   Asserted. The access has a multiple data phases.

PTNUM[1:0]   01. Indicates the PCI access was to pass-thru region 1.

PTWR   Asserted. The pass-thru access is a write.

PTBE[3:0]#   0h. Indicate the pass-thru access is 32-bits.

The PTADR# input is asserted to read the Pass-thru Address Register. The byte enable, address, and SELECT# inputs are changed during this clock to select the Pass-thru Data Register during clock cycle 3.

Clock 3:   SELECT#, byte enables, and the address inputs remain driven to read the Pass-thru Data Register at offset 2Ch. RD# is asserted to drive data register contents onto the DQ bus.

Clock 4:   Add-on logic uses the rising edge of clock 4 to store DATA 1 from the S5933. PTRDY# asserted at the rising edge of clock 4 completes the current data phase. DATA 2 is driven on the add-on bus.

Clock 5:   Add-on logic uses the rising edge of clock 5 to store DATA 2 from the S5933. PTRDY# asserted at the rising edge of clock 5 completes the current data phase. DATA 3 is driven on the add-on bus.

Clock 6:   Add-on logic uses the rising edge of clock 6 to store DATA 3 from the S5933. PTRDY# asserted at the rising edge of clock 6 completes the current data phase. On the PCI bus, IRDY# has been deasserted, causing PTATN# to be deasserted. This is how a PCI initiator adds wait states, if it cannot provide data quickly enough. Data on the add-on bus is not valid.

Clock 7:   Because PTATN# remains deasserted, add-on logic cannot store data at the rising edge of clock 7. PTATN# is reasserted, indicating the PCI initiator is no longer adding wait states. DATA 4 is driven on the add-on bus.

Clock 8:   Add-on logic uses the rising edge of clock 8 to store DATA 4 from the S5933. PTRDY# asserted at the rising edge of clock 8 completes the current data phase. On the PCI bus, IRDY# has been deasserted again, causing PTATN# to be deasserted. Data on the add-on bus is not valid.

Clock 9:   The PCI initiator is still adding wait states. Add-on logic cannot store data while PTATN# is deasserted.

Clock 10:   Because PTATN# remains deasserted, add-on logic cannot read data at the rising edge of clock 10. PTATN# is reasserted, indicating the PCI initiator is no longer adding wait states. DATA 5 is driven on the add-on bus.

Clock 11:   Add-on logic uses the rising edge of clock 11 to store DATA 5 from the S5933. PTRDY# asserted at the rising edge of clock 11 completes the current data phase. DATA 6 is driven on the add-on bus.

Clock 12:   Add-on logic uses the rising edge of clock 12 to store DATA 6 from the S5933. PTRDY# asserted at the rising edge of clock 12 completes the final data phase

Clock 13:   PTATN# and PTBURST# deasserted at the rising edge of clock 13 indicates the pass-thru access is complete. The S5933 can accept new pass-thru accesses from the PCI bus at clock 15.

Figure 11-5 also shows a 5 data phase pass-thru burst write, but the add-on logic uses PTRDY# to control the rate at which data is transferred. In many applications, add-on logic is not fast enough to accept data at every BPCLK rising edge (every 30 ns in a 33 MHz PCI system). In this example, the add-on interface accepts data every other clock. In the example, RD# is asserted during the entire add-on burst, but it can be deasserted when PTRDY# is deasserted, the S5933 functions the same under both conditions.

Clock 0:   PCI address information is stored in the S5933 Pass-thru Address Register.

Clock 1:   The PCI address is recognized as an access to pass-thru region 1. PCI data for the first data phase is stored in the S5933 Pass-thru Data Register. PTATN# is asserted by the S5933 to indicate a pass-thru access is occurring.

Clock 2:   Pass-thru status signals indicate what action is required by add-on logic. Pass-thru status outputs are valid when PTATN# is active and are sampled by the add-on at the rising edge of clock 2.

PTBURST#   Asserted. The access has multiple data phases.

PTNUM[1:0]   01. Indicates the PCI access is to pass-thru region 1.

PTWR   Asserted. The pass-thru access is a write.

PTBE[3:0]#   0h. Indicate the pass-thru access is 32-bits.

11

Figure 11-5. Pass-thru Burst Writes Controlled by PTRDY#
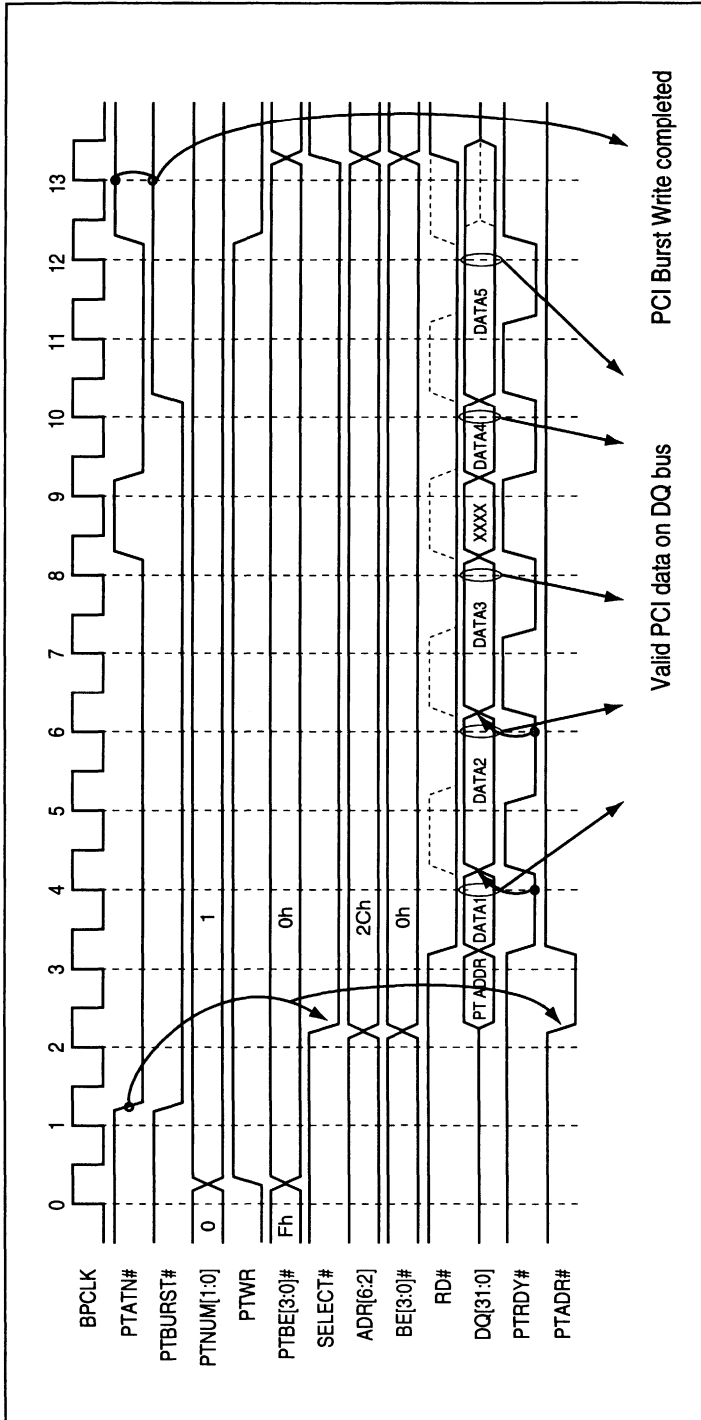
The PTADR# input is asserted to read the Pass-thru Address Register. The byte enable, address, and SELECT# inputs are changed during this clock to select the Pass-thru Data Register during clock cycle 3.

Clock 3: SELECT#, byte enable, and the address inputs remain driven to read the Pass-thru Data Register at offset 2Ch. RD# is asserted to drive data register contents onto the add-on data bus.

Clock 4: Add-on logic uses the rising edge of clock 4 to store DATA 1 from the S5933. PTRDY# asserted at the rising edge of clock 4 completes the current data phase. DATA 2 is driven on the add-on bus.

Clock 5: Add-on logic is not fast enough to store DATA 2 by the rising edge of clock 5. PTRDY# deasserted at the rising edge of clock 5 extends the current data phase and DATA 2 remains driven on the add-on bus.

Clock 6: Add-on logic uses the rising edge of clock 6 to store DATA 2 from the S5933. PTRDY# asserted at the rising edge of clock 6 completes the current data phase. DATA 3 is driven on the add-on bus.

Clock 7: Add-on logic is not fast enough to store DATA 3 by the rising edge of clock 7. PTRDY# deasserted at the rising edge of clock 7 extends the current data phase is and DATA 3 remains driven on the add-on bus.

Clock 8: Add-on logic uses the rising edge of clock 8 to store DATA 3 from the S5933. PTRDY# asserted at the rising edge of clock 8 completes the current data phase. On the PCI bus, IRDY# has been deasserted, causing PTATN# to be deasserted. Data on the add-on bus is not valid.

Clock 9: Because PTATN# remains deasserted, add-on logic cannot store data at the rising edge of clock 9. PTATN# is reasserted, indicating the PCI initiator is no longer adding wait states. DATA 4 is driven on the add-on bus

Clock 10: Add-on logic uses the rising edge of clock 10 to store DATA 4 from the S5933. PTRDY# asserted at the rising edge of clock 10 completes the current data phase. DATA 5 is driven on the add-on bus. PTBURST# is deasserted, indicating that on the PCI bus, the burst is complete except for the last data phase. Since the data is double buffered, there may be one or two pieces of data available to the add-on when PTBURST# becomes inactive.

This example shows the single data available case. If another piece of data was available, then PTATN# would remain active instead of going inactive at clock 12.

Clock 11: Add-on logic is not fast enough to store DATA 5 by the rising edge of clock 11. PTRDY# deasserted at the rising edge of clock 11 extends the data phase and DATA 5 remains driven on the add-on bus.

Clock 12: Add-on logic uses the rising edge of clock 12 to store DATA 5 from the S5933. PTRDY# asserted at the rising edge of clock 12 completes the final data phase.

Clock 13: PTATN# deasserted at the rising edge of clock 13 indicates the pass-thru access is complete. The S5933 can accept new pass-thru accesses from the PCI bus at clock 14.

11.2.2.4 Pass-Thru Burst Reads

A pass-thru burst read operation occurs when a PCI initiator reads multiple DWORDs from a pass-thru region. A burst transfer consists of a single address and a multiple data phases. During the address phase of the PCI transfer, the S5933 stores the PCI address into the Pass-thru Address Register (APTA). If the S5933 determines that the address is within one of its defined pass-thru regions, it indicates to the add-on that a write to the Pass-thru Data Register (APTD) is required. Figure 11-6 shows a 6 data phase pass-thru burst read access (add-on write) using PTADR#.

Clock 0: PCI address information is stored in the S5933 Pass-thru Address Register. The PCI address is recognized as an access to pass-thru region 1. PTATN# is asserted by the S5933 to indicate a pass-thru access is occurring.

Clock 1: Pass-thru status signals indicate what action is required by add-on logic. Pass-thru status outputs are valid when PTATN# is active and are sampled by the add-on at the rising edge of clock 2.

PTBURST#	Deasserted, the S5933 does not yet recognize a PCI burst.
PTNUM[1:0]	01. Indicates the PCI access is to pass-thru region 1.
PTWR	Deasserted. The pass-thru access is a read.
PTBE[3:0]#	0h. Indicate the pass-thru access is 32-bits.

11

AMCC

Figure 11-6. Pass-thru Burst Read
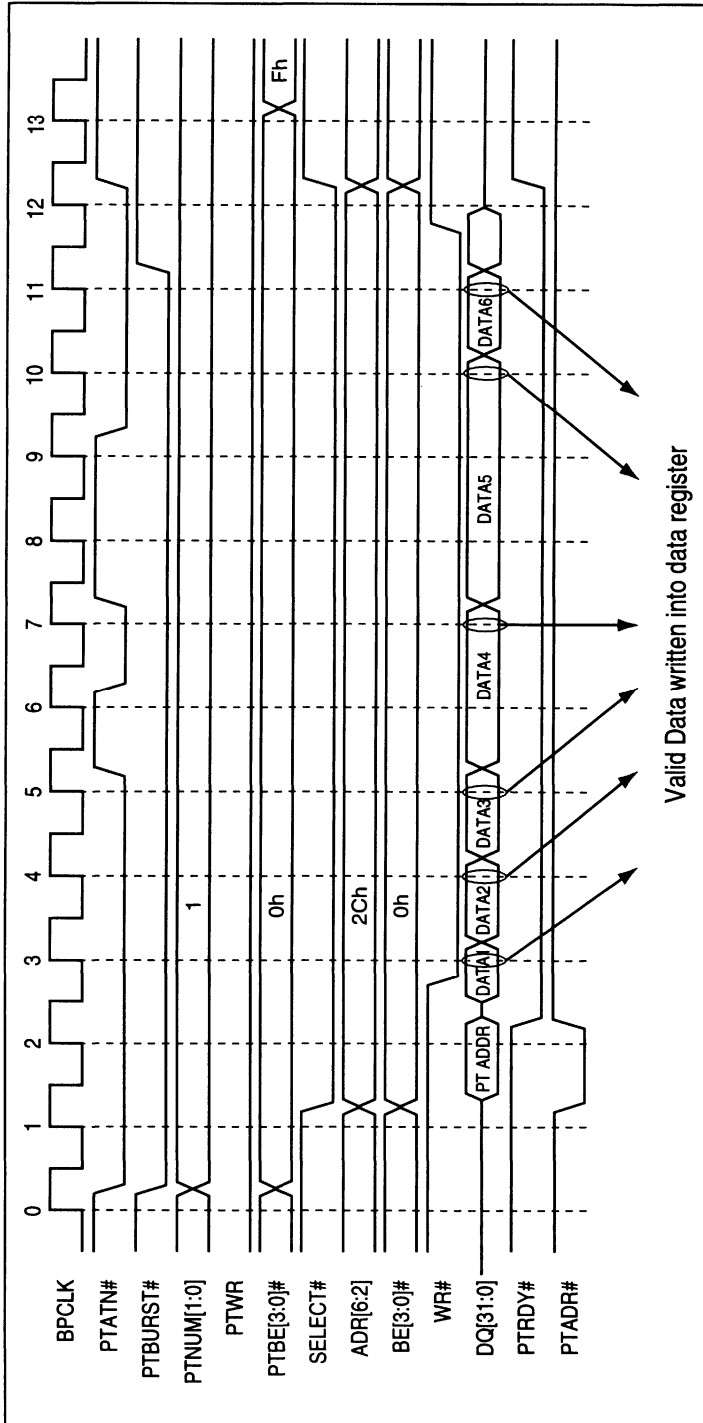
The PTADR# input is asserted to read the Pass-thru Address Register. The byte enable, address, and SELECT# inputs are changed during this clock to select the Pass-thru Data Register during clock cycle 3.

Clock 2: SELECT#, byte enables, and the address inputs remain driven to read the Pass-thru Data Register at offset 2Ch. PTBURST# is asserted by the S5933, indicating the current pass-thru read is a burst.

Clock 3: WR# asserted at the rising edge of clock 3 writes DATA 1 into the S5933. PTRDY# asserted at the rising edge of clock 3 completes the current data phase.

Clock 4: WR# asserted at the rising edge of clock 4 writes DATA 2 into the S5933. PTRDY# asserted at the rising edge of clock 4 completes the current data phase.

Clock 5: WR# asserted at the rising edge of clock 5 writes DATA 3 into the S5933. PTRDY# asserted at the rising edge of clock 5 completes the current data phase. On the PCI bus, IRDY# has been deasserted, causing PTATN# to be deasserted. This is how a PCI initiator adds wait states, if it cannot read data quickly enough.

Clock 6: PTATN# remains deasserted at the rising edge of clock 6. The add-on cannot write DATA 4 until PTATN# is asserted. PTATN# is reasserted during the cycle, indicating the PCI initiator is no longer adding wait states. Add-on logic continues to drive DATA 4 on the add-on bus.

Clock 7: WR# asserted at the rising edge of clock 7 writes DATA 4 into the S5933. PTRDY# asserted at the rising edge of clock 7 completes the current data phase. On the PCI bus, IRDY# has been deasserted, causing PTATN# to be deasserted. The PCI initiator is adding wait states.

Clock 8: PTATN# remains deasserted at the rising edge of clock 8. The add-on cannot write DATA 5 until PTATN# is asserted. Add-on logic continues to drive DATA 5 on the add-on bus.

Clock 9: PTATN# remains deasserted at the rising edge of clock 9. The add-on cannot write DATA 5 until PTATN# is asserted. Add-on logic continues to drive DATA 5 on the add-on bus. PTATN# is reasserted during the cycle, indicating the PCI initiator is done adding wait states.

Clock 10: WR# asserted at the rising edge of clock 10 writes DATA 5 into the S5933. PTRDY# asserted at the rising edge of clock 10 completes the current data phase.

Clock 11: WR# asserted at the rising edge of clock 11 writes DATA 6 into the S5933. PTRDY# asserted at the rising edge of clock 11 completes the final data phase.

Clock 12: PTATN# and PTBURST# deasserted at the rising edge of clock 12 indicates the pass-thru access is complete. The S5933 can accept new pass-thru accesses from the PCI bus at clock 14. Any data written into the pass-thru data register is not required and is never passed to the PCI interface (as PTRDY# is not asserted at the rising edge of clock 13).

Figure 11-7 also shows a 5 data phase pass-thru burst read, but the add-on logic uses PTRDY# to control the rate at which data is transferred. In many applications, add-on logic is not fast enough to provide data every BPCLK (every 30 ns in a 33 MHz PCI system). In this example, the add-on interface writes data every other clock cycle. WR# is shown asserted during the entire add-on burst, but WR# can be deasserted when PTRDY# is deasserted, the S5933 functions the same under both conditions.

Clock 0: PCI address information is stored in the S5933 Pass-thru Address Register. The PCI address is recognized as an access to pass-thru region 1. PTATN# is asserted by the S5933 to indicate a pass-thru access is occurring.

Clock 1: Pass-thru status signals indicate what action is required by add-on logic. Pass-thru status outputs are valid when PTATN# is active and are sampled by the add-on at the rising edge of clock 2.

PTBURST#	Deasserted, the S5933 does not yet recognize a PCI burst.
PTNUM[1:0]	01. Indicates the PCI access is to pass-thru region 1.
PTWR	Deasserted. The pass-thru access is a read.
PTBE[3:0]#	0h. Indicate the pass-thru access is 32-bits.

The PTADR# input is asserted to read the Pass-thru Address Register. The byte enable, address, and SELECT# inputs are changed during this clock to select the Pass-thru Data Register during clock cycle 3.

11

![AMCC logo]

Figure 11-7 : PCI Burst Read Controlled by PTRDY#
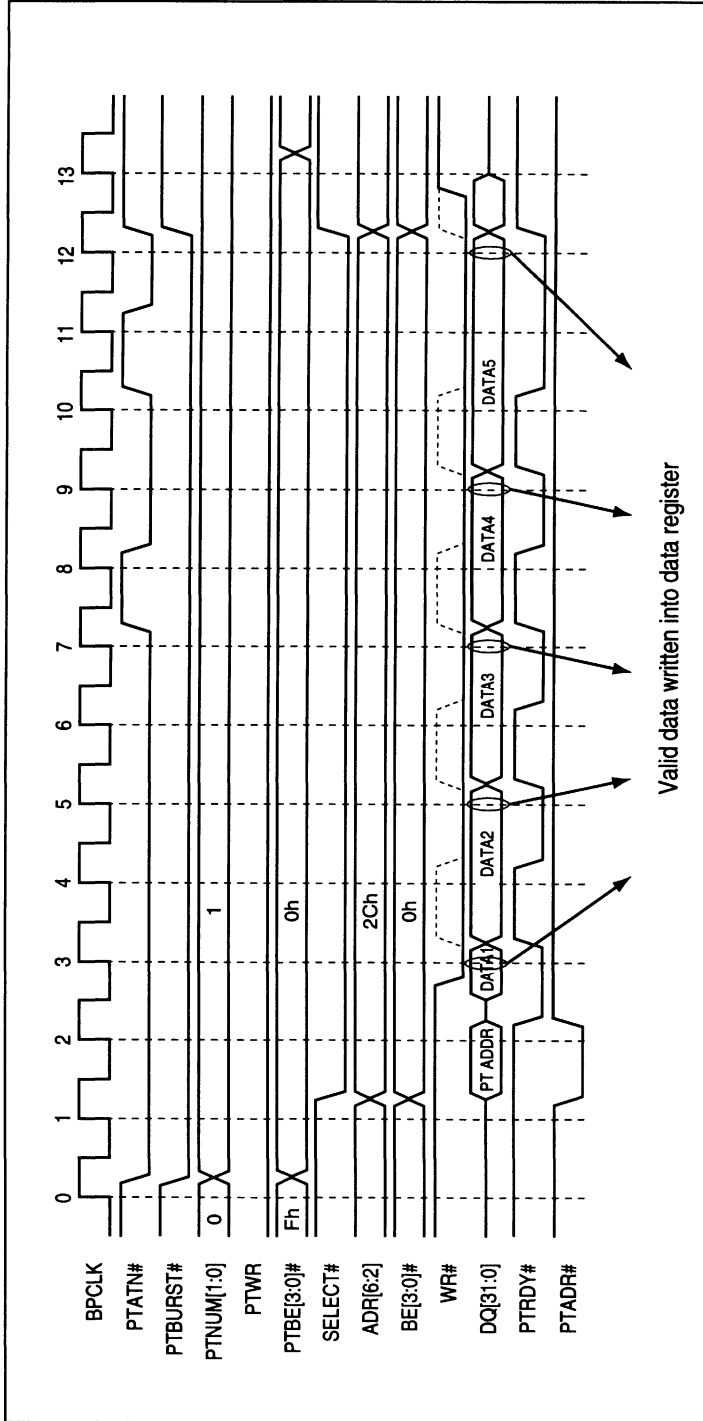
Clock 2: SELECT#, byte enable, and the address inputs remain driven to read the Pass-thru Data Register at offset 2Ch. PTBURST# is asserted by the S5933, indicating the current pass-thru read is a burst.

Clock 3: WR# asserted at the rising edge of clock 3 writes DATA 1 into the S5933. PTRDY# asserted at the rising edge of clock 3 completes the current data phase.

Clock 4: Add-on logic drives DATA 2 on the add-on bus, but PTRDY# deasserted at the rising edge of clock 4 extends the current data phase.

Clock 5: WR# asserted at the rising edge of clock 5 writes DATA 2 into the S5933. PTRDY# asserted at the rising edge of clock 5 completes the current data phase.

Clock 6: Add-on logic drives DATA 3 on the add-on bus, but PTRDY# deasserted at the rising edge of clock 6 extends the current data phase.

Clock 7: WR# asserted at the rising edge of clock 7 writes DATA 3 into the S5933. PTRDY# asserted at the rising edge of clock 7 completes the current data phase. On the PCI bus, IRDY# has been deasserted, causing PTATN# to be deasserted. This is how a PCI initiator adds wait states, if it cannot read data quickly enough.

Clock 8: PTATN# remains deasserted at the rising edge of clock 8. The add-on cannot write DATA 4 until PTATN# is asserted. Add-on logic continues to drive DATA 4 on the add-on bus. PTATN# is reasserted during the cycle, indicating the PCI initiator is done adding wait states.

Clock 9: WR# asserted at the rising edge of clock 9 writes DATA 4 into the S5933. PTRDY# asserted at the rising edge of clock 9 completes the current data phase.

Clock 10: Add-on logic drives DATA 5 on the add-on bus, but PTRDY# deasserted at the rising edge of clock 10 extends the current data phase.

Clock 11: PTATN# remains deasserted at the rising edge of clock 11. The add-on does not have to write DATA 5 until PTATN# is asserted. Add-on logic continues to drive DATA 5 on the add-on bus. PTATN# is reasserted during the cycle, indicating the PCI initiator is done adding wait states.

Clock 12: PTRDY# asserted at the rising edge of clock 12 completes the final data phase. Any data written into the pass-thru data register is not required and is never passed to the PCI interface (as PTRDY# is not asserted at the rising edge of clock 13).

Clock 13: PTATN# and PTBURST# deasserted at the rising edge of clock 13 indicates the pass-thru access is complete. The S5933 can accept new pass-thru accesses from the PCI bus at clock 14.

11.2.2.5 Add-on Pass-Thru Disconnect Operation

As discussed in Section 11.2.1.3, slow PCI targets are prevented from degrading PCI bus performance. The PCI specification allows only 16 clocks for a target to respond before it must request a retry on single data phase accesses. For burst accesses, the first data phase is allowed 16 clocks to complete, all subsequent data phases are allowed 8 clocks each. This requirement allows other PCI initiators to use the bus while the target requesting the retry completes the original access.

Figure 11-8 shows the conditions that cause the S5933 to request a retry from a PCI initiator on the first data phase of a PCI read operation. FRAME# is asserted during the rising edge of PCI clock 1. From this point, the S5933 has 16 clock cycles to respond

11

Figure 11-8. Target Requested Retry on the First PCI Data Phase

PTRDY# must be asserted by this time to present disconnecting

PTRDY# asserted too late so S593X disconnects (asserts STOP#)
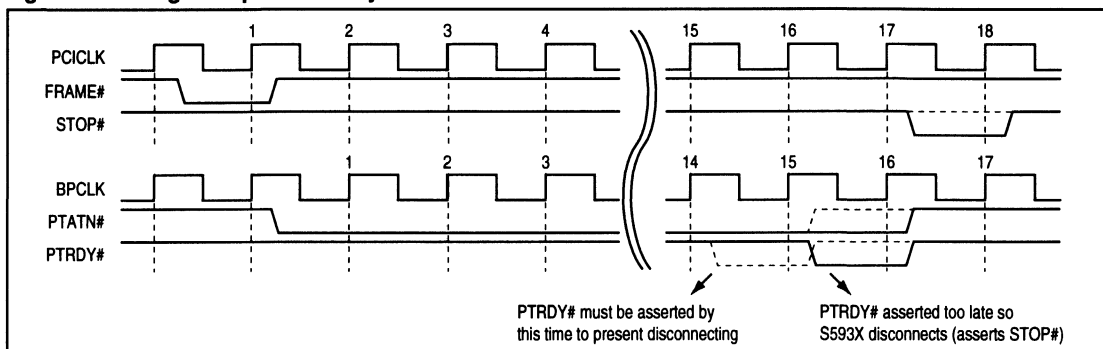
to the initiator with TRDY# (completing the cycle). FRAME# could remain asserted, indicating a burst read, but the retry request conditions are identical for a single data phase read and the first data phase of a burst read.

BPCLK is identical to PCICLK, lagging by a propagation delay of a few nanoseconds (see Chapter 12). PTATN# is asserted on the add-on interface as soon as FRAME# is sampled active at a PCICLK rising edge.

After PTATN# is asserted, PTRDY# must be asserted by the 15th BPCLK rising edge to prevent the S5933 from requesting a retry. TRDY# is asserted on the PCI interface one clock cycle after PTRDY# is asserted on the add-on interface. If add-on logic does not return PTRDY# by the 15th BPCLK rising edge, the S5933 asserts STOP#, requesting a retry from the PCI initiator.

For pass-thru write operations, the S5933 never disconnects on the first or second PCI data phases of a burst. The first data and second phases are always accepted immediately by the S5933. No further action is required by the PCI initiator. The only situation where the S5933 may respond to a pass-thru write with a retry request is after the second data phase of a pass-thru burst write.

Figure 11-9 shows the conditions required for the S5933 to request a retry after the second data phase of a burst transfer. This figure applies to both pass-thru burst read and write operations.

The previous data phase is completed with the assertion of PTRDY# at the rising edge of BPCLK 0. Add-on logic must assert PTRDY# by the rising edge of BPCLK 8 to prevent the S5933 from asserting STOP#, requesting a retry. Meeting this condition allows the S5933 to assert TRDY# by the rising edge of PCICLK 8, completing the data phase with requiring a retry.

When the S5933 requests a retry, the pass-thru status indicators remain valid (allowing the add-on logic to complete the access). PTBURST# is the exception to this. PTBURST# is deasserted to indicate that there is currently no burst in progress on the PCI bus. The other pass-thru status indicators remain valid until PTATN# is deasserted. Figure 11-10 shows the add-on bus interface signals after the S5933 requests a retry.

As long as PTATN# remains asserted, add-on logic should continue to transfer data. For PCI read operations, one add-on write operation is required after a retry request. After the add-on write, asserting PTRDY# deasserts PTATN#.

For pass-thru write operations, one or two data transfers may remain after the S5933 signals a retry. Two data transfers are possible because the S5933 has a double buffered pass-thru data register used for writes. A PCI burst may have filled both registers before the S5933 requested a retry. PTATN# remains asserted until both are emptied. PTRDY# must be asserted after each read from the pass-thru data register. If both registers are full, PTATN# is deasserted only after PTRDY# is asserted the second time. The S5933 only accepts further PCI accesses after both registers are emptied.

11.2.2.6 8-Bit and 16-Bit Pass-Thru Add-on Bus Interface

The S5933 allows a simple interface to devices with 8-bit or 16-bit data buses. Each pass-thru region may be defined as 8-, 16-, or 32-bits, depending on the contents of the nv memory boot device loaded into the PCI Base Address Configuration Registers during initialization (Section 11.3.2). The pass-thru add-on interface internally controls byte lane steering to allow access to the 32-bit Pass-thru Data Register (APTD) from 8-bit or 16-bit add-on buses.

Figure 11-9. Target Requested Retry after the First Data Phase of a Burst Operation

PCI Data Transfer   Latest assertion of PTRDY#   PTRDY# asserted too late,
                           to prevent disconnect   results in disconnect
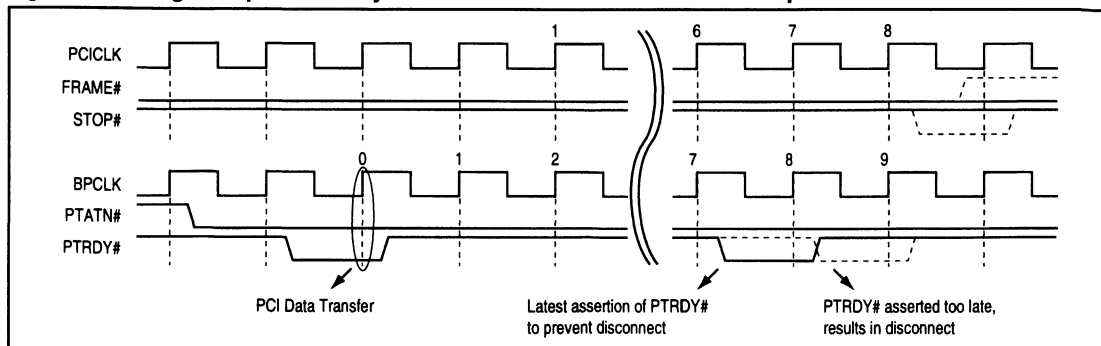
Figure 11-10. Pass-thru Signals after a Target Requested Retry
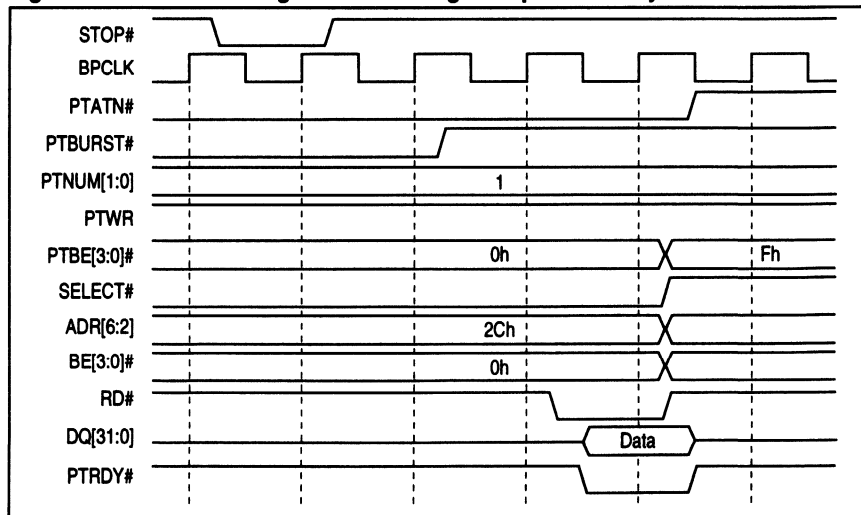
Internal byte lane steering may be used whether the MODE input defines a 16-bit or 32-bit add-on interface. When a 16-bit add-on interface is used, the ADR1 input is used in conjunction with the byte enables to steer data into the proper APTD register byte locations.

If MODE defines a 16-bit interface, only 16-bits of address are driven when PTADR# is asserted. If more than 16-bits of address are required, the Pass-thru Address Register (APTA) must be read with SELECT#, RD#, byte enable and address inputs. Two consecutive reads are required to latch all of the address information (one with ADR1=0, one with ADR1=1).

Regardless of MODE, various data widths may be used. For pass-thru writes (add-on APTD reads), add-on logic must read the APTD register one byte or one word at a time (depending on the add-on bus width). The internal data bus is steered to the correct portion of APTD using the BE[3:0]# inputs. Table 11-1 shows the byte lane steering mechanism used by the S5933. The BYTEn symbols indicate data bytes in the Pass-thru Data Register.

When a read is performed with a BEn# input asserted, the corresponding PTBEn# output is deasserted. Add-on logic cycles through the byte enables to read the entire APTD register. Once all data is read (PTBE[3:0]# are deasserted), PTRDY# is asserted by the add-on, completing the access.

For pass-thru reads (add-on APTD writes), the bytes requested by the PCI initiator are indicated by the PTBE[3:0]# outputs. Add-on logic uses the PTBE[3:0]# signals to determine which bytes must be written (and which bytes have already been written). For example, a

Table 11-1. Byte Lane Steering for Pass-thru Data Register Read

Byte Enables				APTD Register Read Byte Lane Steering			
3	2	1	0	DQ[31:24]	DQ[23:16]	DQ[15:8]	DQ[7:0]
x	x	x	0	BYTE3	BYTE2	BYTE1	BYTE0
x	x	0	1	BYTE3	BYTE2	BYTE1	BYTE1
x	0	1	1	BYTE3	BYTE2	BYTE3	BYTE2
0	1	1	1	BYTE3	BYTE3	BYTE3	BYTE3

PCI initiator performs a byte pass-thru read from an 8-bit pass-thru region with PCI BE2# asserted. On the add-on interface, PTBE2# is asserted, indicating that the PCI initiator requires data in this byte. Once the add-on writes APTD, byte 2, PTBE2# is deasserted, and the add-on may assert PTRDY#, completing the cycle.

Table 11-2 shows how the external add-on data bus is steered to the Pass-thru Data Register bytes. This mechanism is determined by the pass-thru region bus width defined during initialization (see Section 11.3). The BYTEn symbols indicate data bytes in the Pass-thru Data Register. For example, an 8-bit add-on write with BE1# asserted results in the data on DQ[7:0] being steered into BYTE1 of the APTD register.

Table 11-2. Byte Lane Steering for Pass-thru Data Register Write

Defined PT-Bus Width	APTD Register Write Byte Lane Steering			
	BYTE3	BYTE2	BYTE1	BYTE0
32-Bit Data Bus	DQ[31:24]	DQ[23:16]	DQ[15:8]	DQ[7:0]
16-Bit Data Bus	DQ[15:8]	DQ[7:0]	DQ[15:8]	DQ[7:0]
8-Bit Data Bus	DQ[7:0]	DQ[7:0]	DQ[7:0]	DQ[7:0]

11

To write data into the APTD Register, the PTBEn# output and the BEn# input must both be asserted. The following describes how APTD Register writes are controlled:

Write BYTE3 if PTBE3# AND BE3# are asserted

Write BYTE2 if PTBE2# AND BE2# are asserted

Write BYTE1 if PTBE1# AND BE1# are asserted

Write BYTE0 if PTBE0# AND BE0# are asserted

After each byte is written into the pass-thru data register, its corresponding PTBE[3:0]# output is deasserted. This allows add-on logic to monitor which bytes have been written, and which bytes remain to be written. When all bytes requested by the PCI initiator have been written, the PTBE[3:0]# are all be deasserted, and the add-on asserts PTRDY#.

Figure 11-11 shows pass-thru operation for a region defined for an 8-bit add-on bus interface. As the 8-bit device is connected only to DQ[7:0], the device must access APTD one byte at a time.

The PCI initiator has performed a 32-bit write of 08D49A30h to pass-thru region zero. PTBE[3:0]# are all asserted. At clock 1, the add-on begins reading the APTD Register (asserting SELECT#, ADR[6:2], and RD#). Add-on logic asserts BE0#, and BYTE0 of APTD is driven on DQ[7:0]. At the rising edge of clock 2, BE0# is sampled by the S5933 and PTBE0# is deasserted. PTBE[3:1]# are still asserted.

During clock 2, only BE1# is activated, and BYTE1 of APTD is driven on DQ[7:0]. At the rising edge of clock 3, BE1# is sampled by the S5933 and PTBE1# is deasserted. PTBE[3:2]# are still asserted.

This process continues until all bytes have been read from the APTD Register. During clock 5, RD# is deasserted and PTRDY# is asserted. PTRDY# is sampled by the S5933 at the rising edge of clock 6, and the current data phase is completed. PTATN# is deasserted and new data can be written from the PCI bus. In this example, the byte enables are asserted, sequentially, from BE0# to BE3#. This is not required, bytes may be accessed in any order.

New data is written by the PCI initiator and is available in the APTD Register during clock 7. RD# is asserted and the byte enables are cycled again. With each new data from the PCI bus, the add-on sequences through the byte enables to access APTD via DQ[7:0].

For 16-bit peripheral devices, the byte steering works in the same way. Because the add-on data bus is 16-bits wide, only two 16-bit cycles are required to access the entire APTD Register. Two byte enables can be asserted during each access.

In Figure 11-11, RD# is held low and the byte enables are changed each clock. This assumes the add-on can accept data at one byte per clock. This is the fastest transfer possible. For slower devices, wait states may be added.

As long as the byte enables remain in a given state, the corresponding byte of the APTD Register is connected to the DQ bus (the RD# or WR# pulse may also be lengthened). Each access may be extended for slower add-on devices, but extending individual data phases for pass-thru cycles may result in the S5933 requesting retries by the initiator (Section 11.2.5).

Figure 11-11. Pass-Thru Write to an 8-bit Add-on Device
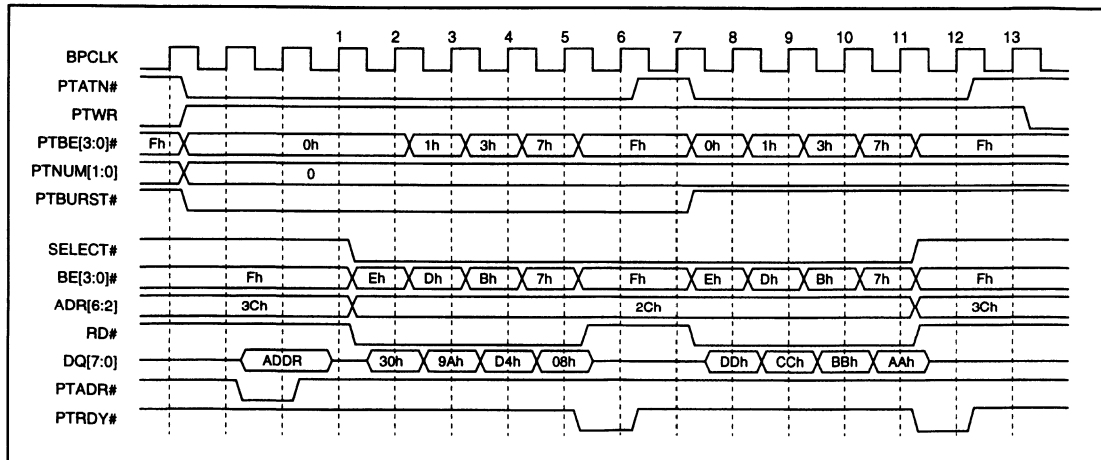
11.3 CONFIGURATION

The S5933 pass-thru interface utilizes four Base Address Registers (BADR1:4). Each Base Address Register corresponds to a pass-thru region. The contents of these registers during initialization determine the characteristics of that particular pass-thru region. Each region can be mapped to memory or I/O space. Memory mapped devices can, optionally, be mapped below 1 Mbyte and can be identified as prefetchable. Both memory and I/O regions can be configured as 8-, 16-, or 32-bits wide.

The designer has the option to use 1, 2, 3, 4 or none of the pass-thru regions. Base Address Registers are loaded during initialization from the external non-volatile boot device. Without an external boot device, the default value for the BADR registers is zero (region disabled). The Base Address Registers are the only registers that define pass-thru operation.

11.3.1 S5933 Base Address Register Definition

Some bits in the Base Address Registers have specific functions. The following bits have special functions:

D0   Memory or I/O mapping. If this bit is clear, the region should be memory mapped. If this bit is set, the region should be I/O mapped.

D2:1   Location of a memory region. These bits request that the region be mapped in a particular part of memory. These bit definitions are only used for memory mapped regions.

D2	D1	Location
0	0	Anywhere in 32-bit memory space
0	1	Below 1 Mbyte in memory space (Real Mode address space)
1	0	Anywhere in 64-bit memory space (not valid for the S5933)
1	1	Reserved

D3   Prefetchable. For memory mapped regions, the region can be defined as cacheable. If set, the region is cacheable. If this bit is clear, the region is not.

D31:30   Pass-thru region bus width. These two bits are used by the S5933 to define the data bus width for a pass-thru region. Regardless of the programming of other bits in the BADR register, if D31:30 are zeros, the pass-thru region is disabled.

D31	D30	Add-on Bus Width
0	0	Region disabled
0	1	8-bits
1	0	16-bits
1	1	32-bits

BADR1:4 bits D31:30 are used only by the S5933. When the host reads the Base Address Registers during configuration cycles, they always return the same value as D29. If D29 is zero, D31:30 return zero, indicating the region is disabled. If D29 is one, D31:30 return one. This operation limits each pass-thru region to a maximum size of 512 Mbytes of memory.

For I/O mapped regions, the PCI specification allows no more than 256 bytes per region. The S5933 allows larger regions to be requested by the add-on, but a PCI BIOS will not allocate the I/O space and will probably disable the region.

11.3.2 Creating a Pass-thru Region

Section 3.11 describes the values that must be programmed into the non-volatile boot device to request various block sizes and characteristics for pass-thru regions (also see Section 11.3.1). After reset, the S5933 downloads the contents of the boot device locations 54h, 58h, 5Ch, and 60h into "masks" for the corresponding Base Address Registers. The following are some examples for various pass-thru region definitions:

NV Memory Contents	Pass-thru Region Definition
54h = BFFFF002h	Pass-thru region 1 is a 4Kbyte region, mapped below 1 Mbyte in memory space with a 16-bit add-on data bus. This memory region is not cacheable.
58h = 3xxxxxxxh	Pass-thru region 2 is disabled. (D31:30 = 00.)
60h = FFFFFF81h	Pass-thru region 3 is a 32-bit, 128 byte I/O-mapped region.
64h = 00000000h	Pass-thru region 4 is disabled.

During the PCI bus configuration, the host CPU writes all ones to each Base Address Register, and then reads the contents of the registers back. The mask downloaded from the boot device determines which bits are read back as zeros and which are read back as ones. The number of zeros read back indicates the amount of memory or I/O space a particular S5933 pass-thru region is requesting (see Section 3.11).

11

After the host reads all Base Address Registers in the system (as every PCI device implements from one to six), the PCI BIOS allocates memory and I/O space to each Base Address region. The host then writes the start address of each region back into the Base Address Registers. The start address of a region is always an integer multiple of the region size. For example, a 64 Kbyte memory region is always mapped to begin on a 64K boundary in memory. It is important to note that no PCI device can xbe absolutely located in system memory or I/O space. All mapping is determined by the system, not the application.

11.3.3 Accessing a Pass-thru Region

After the system is finished defining all Base Address Regions within a system, each Base Address Register contains a physical address. The application software must now find the location in memory or I/O space of its hardware. PCI systems provide BIOS or operating system function calls for application software to find particular devices on the PCI bus based on Vendor ID and Device ID values. This allows application software to access the device's Configuration Registers.

The Base Address Register values in the S5933's Configuration Space may then be read and stored for use by the program to access application hardware. The value in the Base Address Registers is the physical address of the first location of that pass-thru region. Some processor architectures allow this address to be used directly to access the PCI device. For Intel Architecture systems, the physical address must be changed into a Segment/Offset combination.

For Real Mode operation in an Intel Architecture system (device mapped below 1 Mbyte in memory), creating a Segment/Offset pair is relatively simple. To calculate a physical address, the CPU shifts the segment register 4 bits to the left and adds the offset (resulting in a 20 bit physical address). The value in the Base Address Register must be read and shifted 4 bits to the right. This is the segment value and should be stored in one of the Segment registers. An offset of zero (stored in SI, DI or another offset register) accesses the first location in the pass-thru region.

Electrical Characteristics 12

12.0   Electrical Characteristics

12.1   Absolute Maximum Ratings

Parameter	Min	Max	Units
Storage Temperature	−55	125	°C
Supply Voltage (VDD)	−0.3	7.0	Volts
Input Pin Voltage	−0.5	VDD+0.5	Volts

12.2   D.C. Characteristics

The following table summarizes the required parameters defined by the PCI specification as they apply to the S5933 family of controllers.

Input/Output Electrical Characteristics

Symbol	Parameter	Min	Max	Units	Test Conditions	Notes
V_{CC}	Supply Voltage	4.75	5.25	V		
V_{ih}	Input High Voltage	2.0		V		
V_{il}	Input Low Voltage	−0.5	0.8	V		
I_{ih}	Input High Leakage Current		70	μA	$V_{in} = 2.7$	1
I_{il}	Input Low Leakage Current		−70	μA	$V_{in} = 0.5$	1
V_{oh}	Output High Voltage	2.4		V	$I_{out} = -2mA$	
V_{ol}	Output Low Voltage		0.55	V	$I_{out} = 3mA, 6mA$	2
C_{in}	Input Pin Capacitance		10	pF		3
C_{clk}	CLK Pin Capacitance	5	12	pF		
C_{IDSEL}	IDSEL Pin Capaticance		8	pF		

Notes:

1. Input leakage applies to all inputs and bidirectional buffers.

2. PCI bus signals without pull-up resistors will provide the 3 mA output current. Signals which require a pull-up resistor (FRAME#, TRDY#, IRDY#, DEVSEL#, STOP#, SERR#, and PERR#) will provide 6 mA output current.

3. The PCI specification limits all PCI inputs not located on the motherboard to 10 pF (the PCI clock is allowed to be 12 pF).

12.2.1 PCI Bus Signals

AD[31:0] (t/s), C/BE[3:0]#(t/s), FRAME#(s/t/s), IRDY#(s/t/s), TRDY#(s/t/s), STOP#(s/t/s), LOCK#(in), IDSEL(in), DEVSEL#(s/t/s), GNT#(in), RST#(in), CLK(in).

12.2.2 Add-On Bus Signals

DQ[31:0] (t/s), ADR[5:2] (in), BE[3:0]#(in), SELECT#(in), RD#(in), WR#(in) PTATN#(out), PTBE[3:0]#(out), PTNUM[1:0] (out), PTWR(out), PTBURST#(out), PTADR#(in), PTRDY#(in), WRFULL(out), RDEMPTY(out), WRFIFO#(in), RDFIFO#(in), SYSRST#(out), IRQ#(out), BPCLK(out).

All add-on bus signal outputs are capable of sourcing or sinking 8mA.

12.2.3 nv Memory Interface Signals

ERD#/SCL 5mA (out), EWR#/SDA 2mA (t/s), EQ[7:0] 1mA (t/s), EA[15:9] 2mA (out), EA[7:0] 1mA (t/s).

These signal pins are capable of sourcing or sinking 1mA when operated as outputs.

12.3   A.C. Characteristics

12.3.1 PCI Bus Timings

Functional Operation Range (V_{CC}=5.0V ±5%, 0°C to 70°C, 50 pF load on outputs)

Symbol	Parameter	Min	Max	Units	Notes
TCL	Cycle Time	30		ns	
t1	High Time	12		ns	
t2	Low Time	12		ns	
t3	Rise Time (0.8V to 2.0V)		3	ns	
t4	Fall Time (2.0V to 0.8V)		3	ns	
t5	Output Valid Delay (Bussed Signals) Output Valid Delay (Point-to-Point Signals)	2 2	11 12	ns	Note 1
t6	Float to Active Delay	2		ns	
t7	Active to Float Delay		28	ns	
t8	Rising Edge Setup (Bussed Signals) Rising Edge Setup (GNT#) Rising Edge Setup (REQ#)	7 10 12		ns	
t9	Hold from PCI Clock Rising Edge	0		ns	

Notes:
1. Minimum times are for unloaded outputs, maximum times are for 50 pF equivalent loads.

Figure 12-1. PCI Clock Timing
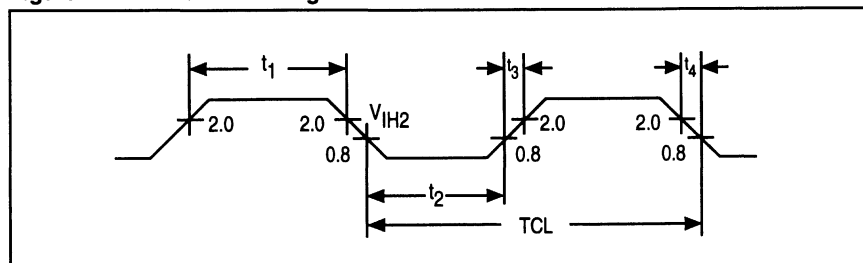
Figure 12-2. PCI Output Timing
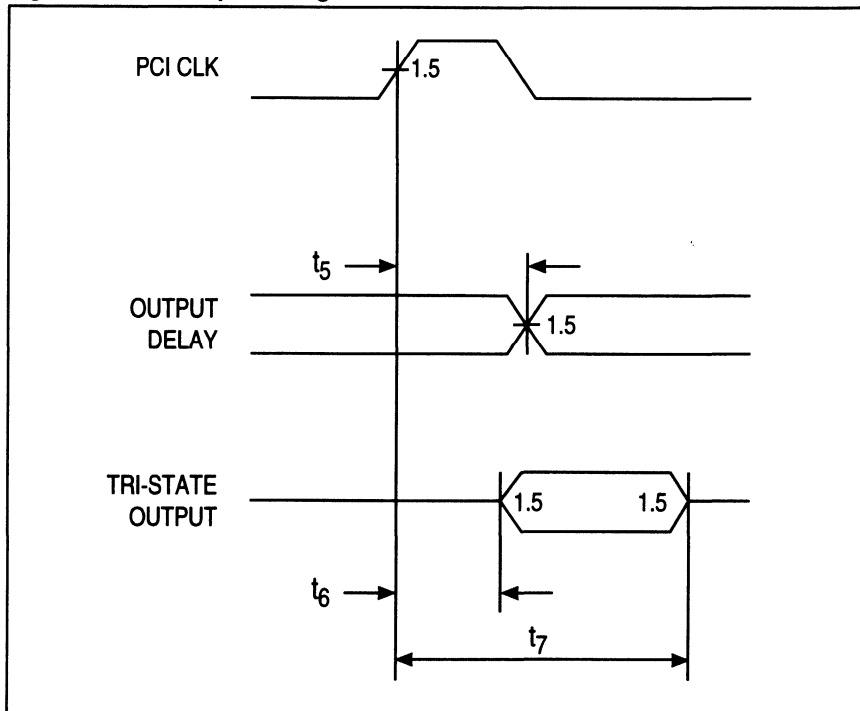
Figure 12-3. PCI Input Timing

12
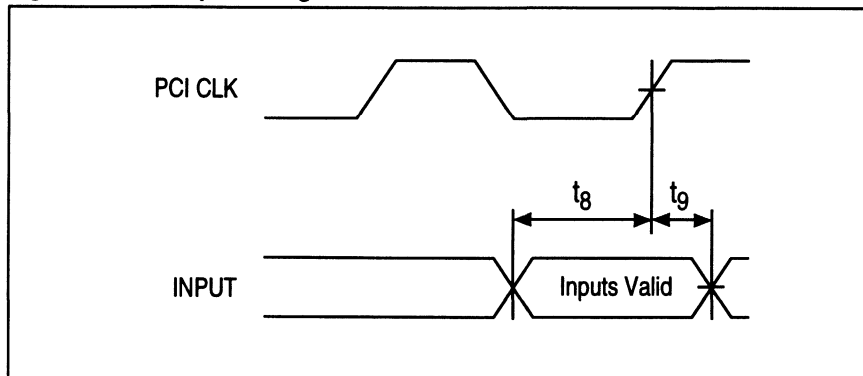
12.3.2 Target S5933 Asynchronous Operation Register Access Timings

Functional Operation Range (V$_{CC}$=5.0V ±5%, 0˚C to 70˚C, 50 pF load on outputs)

Symbol	Parameter	Min	Max	Units	Notes
t10	SELECT# Setup to RD# or WR# Rising Edge	3		ns	
t11	SELECT# Hold from RD# or WR# Rising Edge	2		ns	
t12	ADR[6:2], BE[3:0]# to DQ[31:0] Valid		16	ns	
t13	ADR[6:2], BE[3:0]# Setup to RD#, WR# Rising Edge	5		ns	
t14	ADR[6:2], BE[3:0]# Hold from RD#, WR# Rising Edge	2		ns	
t15	RD#, WR#, RDFIFO#, WRFIFO# or PTADR# High Time	12		ns	
t16	RD#, WR#, RDFIFO#, WRFIFO# or PTADR# Low Time	12		ns	
t17	RD# (or SELECT#), RDFIFO# or PTADR# Low to DQ[31:0] Driven		13	ns	Note 1, 2
t18	RD# (or SELECT#), RDFIFO# or PTADR# High to DQ[31:0] Float		12	ns	Note 1, 2
t19	DQ[31:0] Hold from RD#, RDFIFO# or PTADR# Rising Edge	2		ns	
t20	DQ[31:0] Setup to WR# or WRFIFO# Rising Edge	5		ns	
t21	DQ[31:0] Hold from WR# or WRFIFO# Rising Edge	2		ns	
t22	Add-on to PCI FIFO Status Valid from WR# or WRFIFO# Rising Edge		16	ns	
t23	PCI to Add-on FIFO Status Valid from RD# or RDFIFO# Rising Edge		16	ns	

Notes:

1. This timing also applies to the use of BE[3:0]# to control DQ[31:0] drive.

2. RD# and SELECT# must both be asserted to drive DQ[31:0] — delay is from the last one asserted.

Figure 12-4. Add-on Operation Register Read Timing
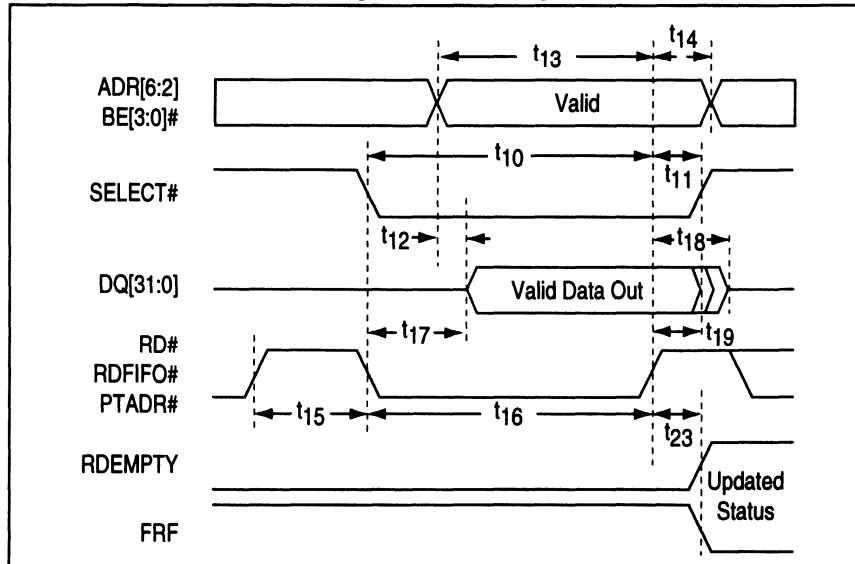
Figure 12-5. Add-on Operation Register Write Timing

12
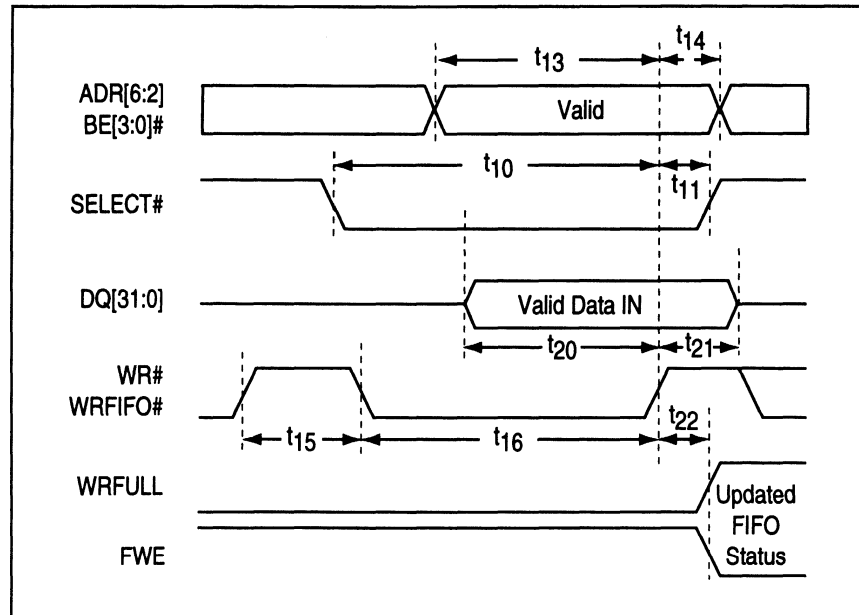
12.3.3 Target S5933 Synchronous FIFO Interface Timings

Functional Operation Range (V_{CC}=5.0V ±5%, 0˚C to 70˚C, 50 pF load on outputs)

Symbol	Parameter	Min	Max	Units	Notes
t10a	SELECT# Setup to BPCLK Rising Edge	3		ns	
t11a	SELECT# Hold from BPCLK Rising Edge	2		ns	
t13	ADR[6:2], BE[3:0]# Setup to BPCLK Rising Edge	5		ns	
t14	ADR[6:2], BE[3:0]# Hold from BPCLK Rising Edge	2		ns	
t17	RD# (or SELECT#) or RDFIFO# Low to DQ[31:0] Driven		13	ns	Note 1, 2
t18	RD# (or SELECT#) or RDFIFO# High to DQ[31:0] Float		12	ns	Note 1, 2
t22a	Add-on to PCI FIFO Status Valid from BPCLK Rising Edge		10.5	ns	
t23a	PCI to Add-on FIFO Status Valid from BPCLK Rising Edge		10.5	ns	
t29	RD#, WR#, RDFIFO# or WRFIFO# Setup to BPCLK Rising Edge	5		ns	
t30	RD#, WR#, RDFIFO# or WRFIFO# Hold from BPCLK Rising Edge	2		ns	
t31	DQ[31:0] Setup to BPCLK Rising Edge	5		ns	
t32	DQ[31:0] Hold from BPCLK Rising Edge	2		ns	
t33	DQ[31:0] Valid from BPCLK Rising Edge		15	ns	

Notes:

1. This timing also applies to the use of BE[3:0]# to control DQ[31:0] drive.

2. RD# and SELECT# must both be asserted to drive DQ[31:0] — delay is from the last one asserted.

Figure 12-6. Synchronous FIFO Read Timing
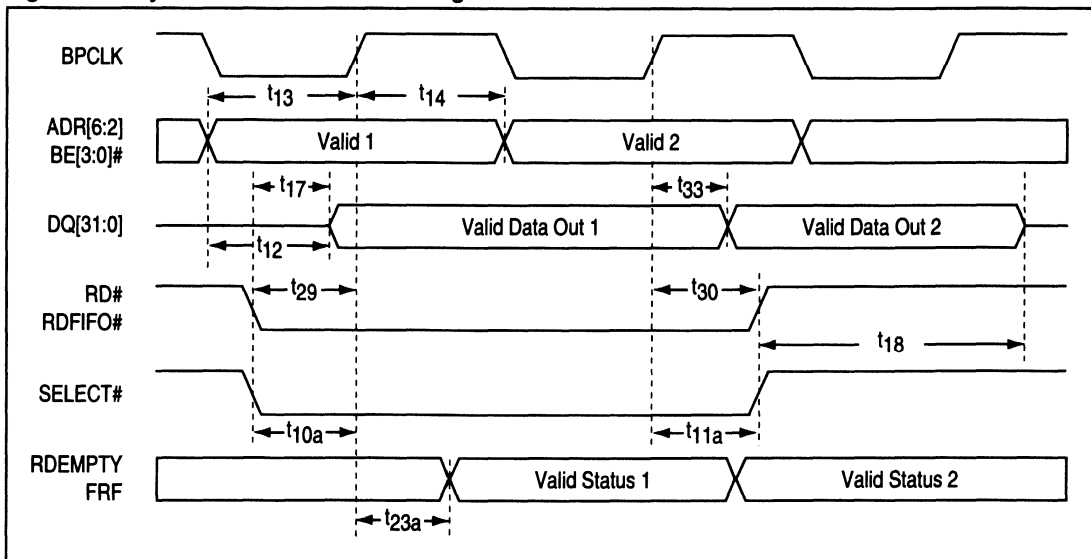
Figure 12-7. Synchronous FIFO Write Timing
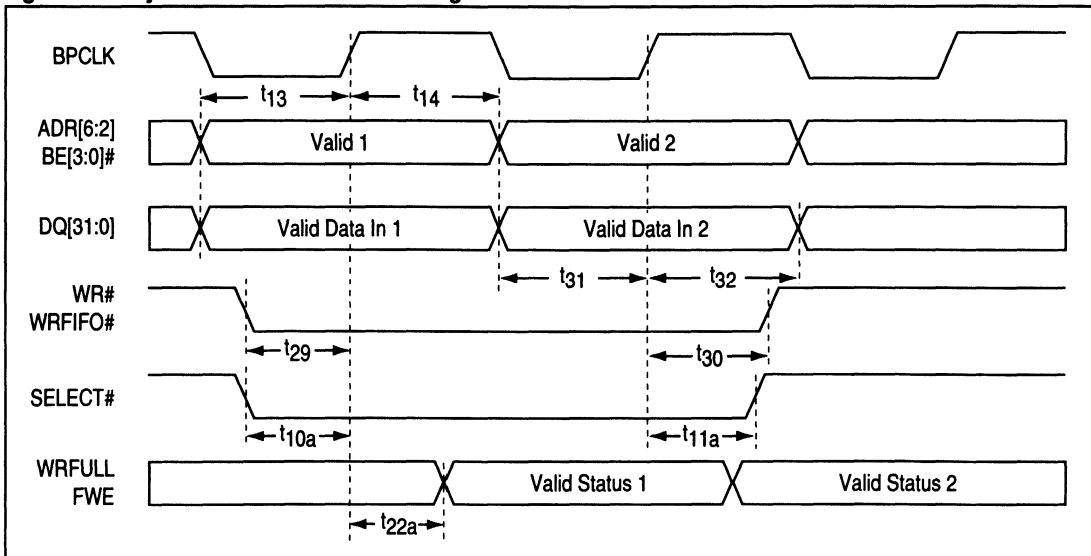
12.3.4 Target FIFO Control Timings

Functional Operation Range (V_{CC}=5.0V ±5%, 0˚C to 70˚C, 50 pF load on outputs)

Symbol	Parameter	Min	Max	Units	Notes
t47	FIFO Reset Low Time	10		ns	
t48	FIFO Reset Low to FIFO Status Valid		17	ns	

Figure 12-8. FIFO Reset Timing
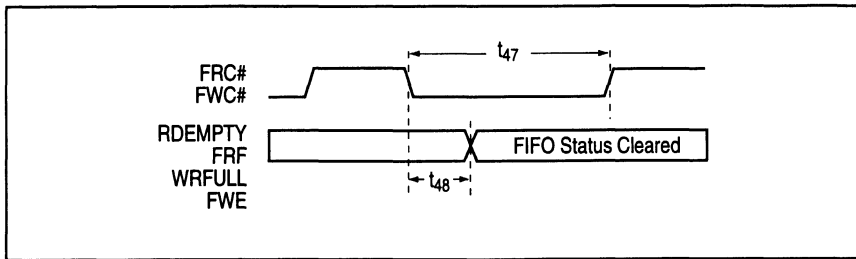
Figure 12-9. Pass Thru Clock Relationship to PCI Bus Clock
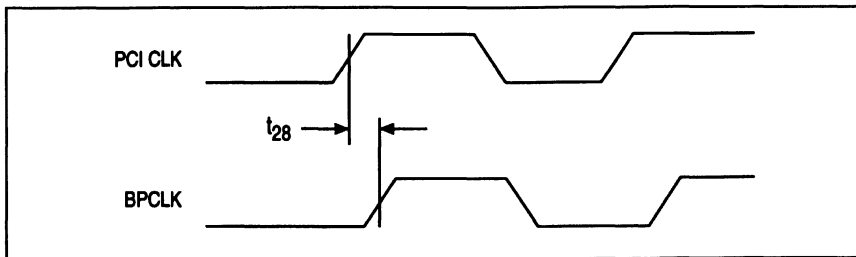
12.3.5 Target S5933 Pass-thru Interface Timings

Functional Operation Range (V_{CC}=5.0V ±5%, 0°C to 70°C, 50 pF load on outputs)

Symbol	Parameter	Min	Max	Units	Notes
t10a	SELECT# Setup to BPCLK Rising Edge	3		ns	
t11a	SELECT# Hold from BPCLK Rising Edge	2		ns	
t13	ADR[6:2], BE[3:0]# Setup to BPCLK Rising Edge	5		ns	
t14	ADR[6:2], BE[3:0]# Hold from BPCLK Rising Edge	2		ns	
t17	RD# Low to DQ[31:0] Driven		13	ns	Note 1
t24	Pass-thru Status Valid from BPCLK Rising Edge		5	ns	
t25	Pass-thru Status Hold from BPCLK Rising Edge	0		ns	
t26	PTRDY# Setup to BPCLK Rising Edge	5		ns	
t27	PTRDY# Hold from BPCLK Rising Edge	3		ns	
t28	PCICLK to BPCLK delay	2	6.5	ns	
t29	RD#, WR# Setup to BPCLK Rising Edge	5		ns	
t30	RD#, WR# Hold from BPCLK Rising Edge	2		ns	
t31	DQ[31:0] Setup to BPCLK Rising Edge	5		ns	
t32	DQ[31:0] Hold from BPCLK Rising Edge	2		ns	
t33	DQ[31:0] Valid from BPCLK Rising Edge		15	ns	
t34	DQ[31:0] Float from RD# Rising Edge		12	ns	

Notes:

1. This timing also applies to the use of BE[3:0]# to control DQ[31:0] drive.

12

Figure 12-10. Pass-thru Data Register Read Timing
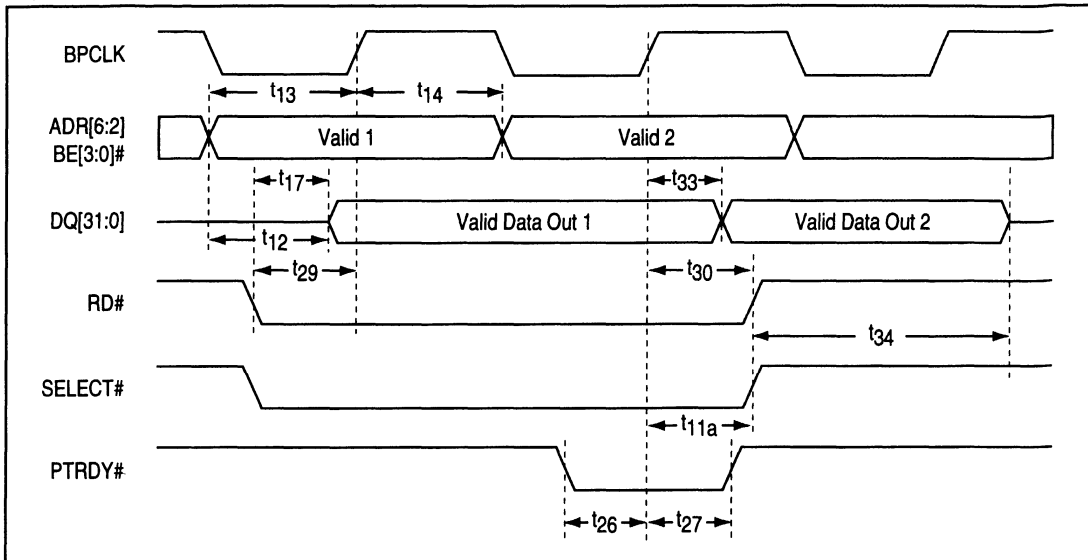
Figure 12-11. Pass-thru Data Register Write Timing
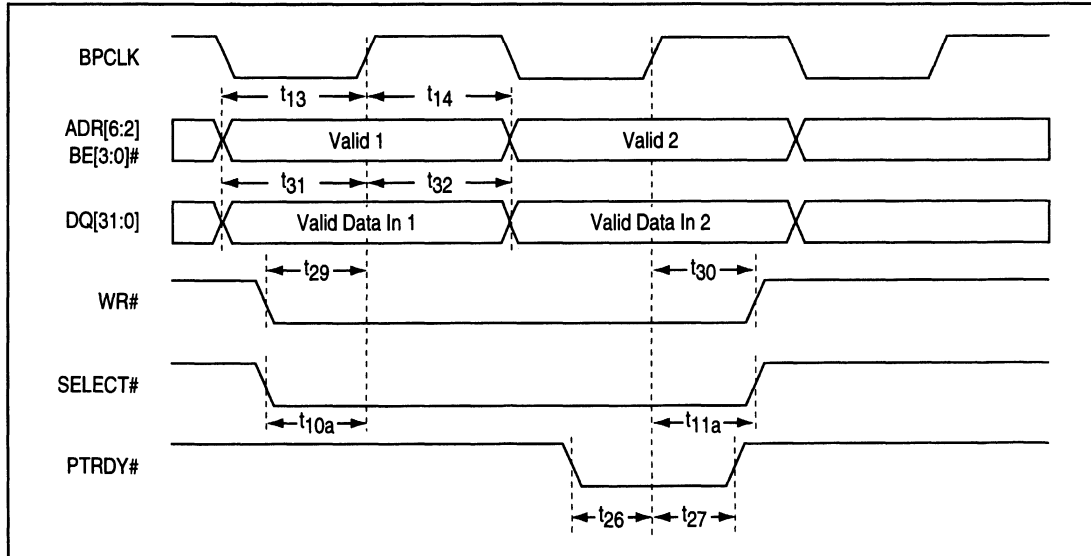
Figure 12-12. Pass-thru Status Indicator Timing
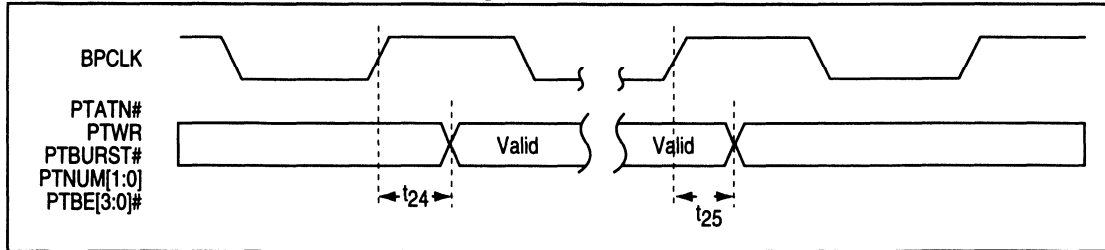
12.3.6 Target Byte-Wide nv Memory Interface Timings

Functional Operation Range (V_{CC}=5.0V ±5%, 0°C to 70°C, 50 pF load on outputs)

Symbol	Parameter	Min	Max	Units	Notes
t35	ERD# Cycle Time	8T		ns	Note 1
t36	ERD# Low Time	6T		ns	Note 1
t37	ERD# High Time	2T		ns	Note 1
t38	EA[15:0] Setup to ERD# or EWR# Low	T		ns	Note 1
t39	EA[15:0] Hold from ERD# or EWR# High	T		ns	Note 1
t40	EQ[7:0] Setup to ERD# Rising Edge	10		ns	Note 1
t41	EQ[7:0] Hold from ERD# Rising Edge	2		ns	Note 1
t42	EWR# Cycle Time			ns	Note 1,2
t43	EWR# Low Time	6T		ns	Note 1
t44	EWR# High Time	2T		ns	Note 1
t45	EQ[7:0] Setup to EWR# Low	-10	0	ns	Note 1
t46	EQ[7:0] Hold from EWR# High	T		ns	Note 1

Notes:

1. T represents the clock period for the PCI bus clock (30ns @ 33 MHz).
2. The write cycle time is controlled by both the PCI bus clock and software operations to initiate the write operation of nv memory. This parameter is the result of several software operations to the Bus Master Control/Status Register (MCSR) — Section 4.10.

12

Figure 12-13. nv Memory Read Timing
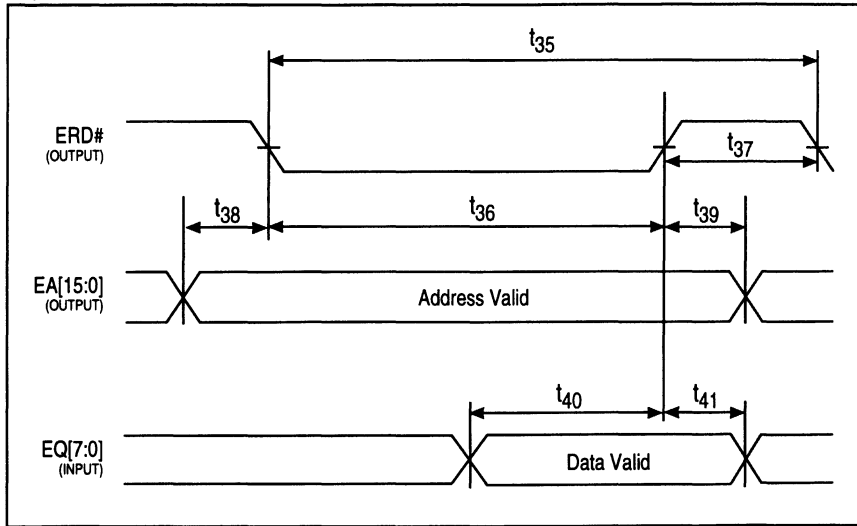
Figure 12-14. nv Memory Write Timing
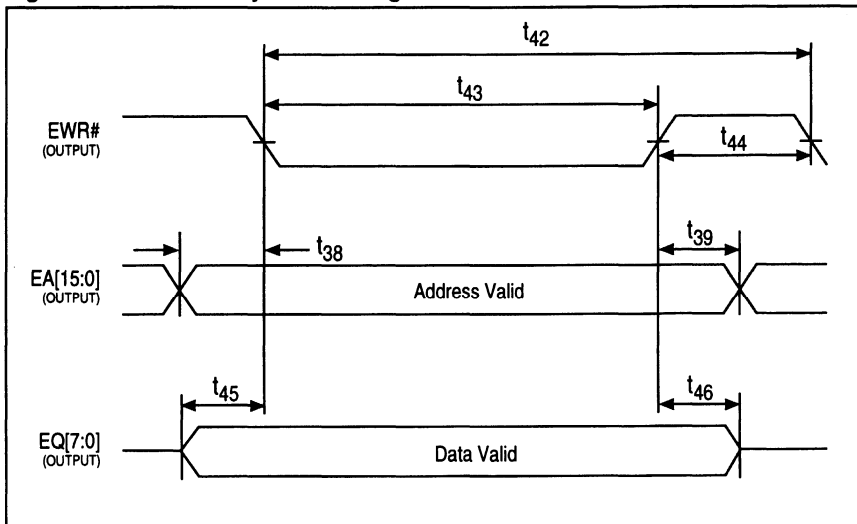
12.3.7 Target Interrupt Timings

Functional Operation Range (V$_{CC}$=5.0V ±5%, 0˚C to 70˚C, 50 pF load on outputs)

Symbol	Parameter	Min	Max	Units	Notes
t49	IRQ# Low from BPCLK Rising Edge		TBD	ns	Note 1
t50	IRQ# High from BPCLK Rising Edge		TBD	ns	Note 1

Notes:

1. This timing applies to interrupts generated and cleared from the PCI interface.

Figure 12-15. IRQ# Interrupt Output Timing
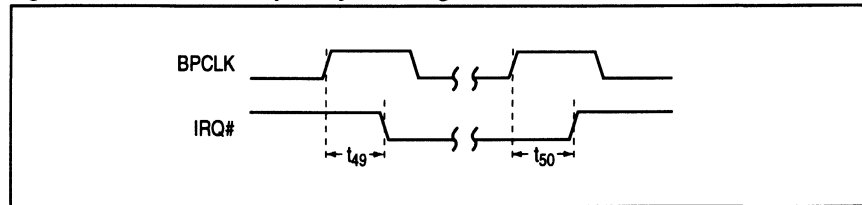
Functional Operation Range (V$_{CC}$=5.0V ±5%, 0˚C to 70˚C, 50 pF load on outputs)

Symbol	Parameter	Min	Max	Units	Notes
t51	EMBCLK Low Time	12		ns	
t52	EMBLK High Time	12		ns	
t53	EMB[7:0] Setup to EMBCLK Rising Edge	5		ns	
t54	EMB[7:0] Hold from EMBCLK Rising Edge	2		ns	

Figure 12-16. Mailbox 4, Byte 3 Direct Input Timing

12
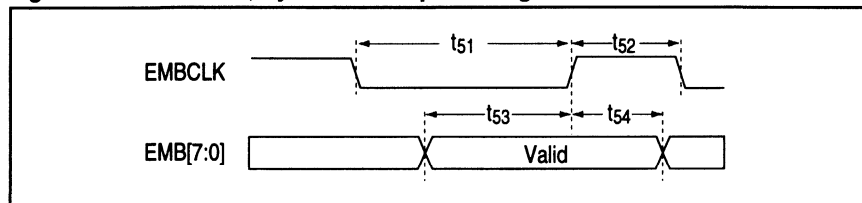
13.0 PACKAGE AND ORDERING INFORMATION

13.1 S5933 Pinout and Pin Assignment

Top edge pins (120 → 81):
PTBE3#, PTBE2#, PTBE1#, DQ20, PTBE0#, PTRDY#, PTATN#, EA9, PTBURST#, VCC, VSS, EA8, PTWR, PTADR#, RDEMPTY, DQ21, RDFIFO#, WRFULL, WRFIFO#, EA7, DQ0, DQ1, DQ2, EA6, DQ3, DQ4, DQ5, DQ22, DQ6, VCC, VSS, EA5, DQ7, BE0#, DQ8, DQ23, DQ9, DQ10, DQ11, EA4

Left side (121 → 160):

Pin	Signal
121	EA10
122	PTNUM1
123	PTNUM0
124	IRQ#
125	DQ19
126	SYSRST#
127	EWR#/SDA
128	ERD#/SCL
129	EA11
130	VSS
131	VCC
132	ADR6
133	DQ18
134	NC
135	SNV
136	NC
137	EA12
138	FLT#
139	RST#
140	BPCLK
141	EA13
142	CLK
143	GNT#
144	REQ#
145	DQ17
146	AD31
147	AD30
148	AD29
149	EA14/FWE
150	VSS
151	VCC
152	AD28
153	EA15/FRF
154	AD27
155	AD26
156	AD25
157	DQ16
158	AD24
159	C/BE3#
160	IDSEL

S5933
(160 PQFP)

Right side (80 → 41):

Pin	Signal
80	DQ12
79	DQ13
78	DQ14
77	DQ24
76	DQ15
75	SELECT#
74	WR#
73	EA3
72	RD#
71	VCC
70	VSS
69	EA2
68	ADR2
67	ADR3
66	ADR4
65	DQ25
64	ADR5
63	BE1#
62	BE2#
61	EA1
60	BE3#
59	MODE
58	INTA#
57	EA0
56	AD0
55	AD1
54	AD2
53	DQ26
52	AD3
51	VCC
50	VSS
49	EQ7/AMWEN
48	AD4
47	AD5
46	AD6
45	DQ27
44	AD7
43	C/BE0#
42	AD8
41	EQ6/AMREN

Bottom edge pins (1 → 40):
EQ0, AD23, AD22, AD21, DQ31, AD20, AD19, AD18, EQ1, VSS, VCC, AD17, DQ30, AD16, C/BE2#, FRAME#, EQ2, IRDY#, TRDY#, DEVSEL#, EQ3, STOP#, LOCK#, PERR#, DQ29, SERR#, PAR, C/BE1#, EQ4/FWC#, VSS, VCC, AD15, EQ5/FRC#, AD14, AD13, AD12, DQ28, AD11, AD10, AD9

13

AMCC

Table 13-1. S5933 Alphabetical Pin Assignment

Signal	Pin#	Type	Signal	Pin#	Type	Signal	Pin#	Type
AD0	56	t/s	ADR2	68	in	DQ16	157	t/s
AD1	55	t/s	ADR3	67	in	DQ17	145	t/s
AD2	54	t/s	ADR4	66	in	DQ18	133	t/s
AD3	52	t/s	ADR5	64	in	DQ19	125	t/s
AD4	48	t/s	ADR6	132	in	DQ20	117	t/s
AD5	47	t/s	BE0#	87	in	DQ21	105	t/s
AD6	46	t/s	BE1#	63	in	DQ22	93	t/s
AD7	44	t/s	BE2#	62	in	DQ23	85	t/s
AD8	42	t/s	BE3#	60	in	DQ24	77	t/s
AD9	40	t/s	BPCLK	140	out	DQ25	65	t/s
AD10	39	t/s	C/BE0#	43	t/s	DQ26	53	t/s
AD11	38	t/s	C/BE1#	28	t/s	DQ27	45	t/s
AD12	36	t/s	C/BE2#	15	t/s	DQ28	37	t/s
AD13	35	t/s	C/BE3#	159	t/s	DQ29	25	t/s
AD14	34	t/s	CLK	142	in	DQ30	13	t/s
AD15	32	t/s	DEVSEL#	20	t/s	DQ31	5	t/s
AD16	14	t/s	DQ0	100	t/s	EA0	57	t/s
AD17	12	t/s	DQ1	99	t/s	EA1	61	t/s
AD18	8	t/s	DQ2	98	t/s	EA2	69	t/s
AD19	7	t/s	DQ3	96	t/s	EA3	73	t/s
AD20	6	t/s	DQ4	95	t/s	EA4	81	t/s
AD21	4	t/s	DQ5	94	t/s	EA5	89	t/s
AD22	3	t/s	DQ6	92	t/s	EA6	97	t/s
AD23	2	t/s	DQ7	88	t/s	EA7	101	t/s
AD24	158	t/s	DQ8	86	t/s	EA8	109	t/s
AD25	156	t/s	DQ9	84	t/s	EA9	113	out
AD26	155	t/s	DQ10	83	t/s	EA10	121	out
AD27	154	t/s	DQ11	82	t/s	EA11	129	out
AD28	152	t/s	DQ12	80	t/s	EA12	137	out
AD29	148	t/s	DQ13	79	t/s	EA13	141	out
AD30	147	t/s	DQ14	78	t/s	EA14/FWE	149	t/s
AD31	146	t/s	DQ15	76	t/s	EA15/FRF	153	t/s

Table 13-1. S5933 Alphabetical Pin Assignment (Continued)

Signal	Pin#	Type		Signal	Pin#	Type
EQ0	1	t/s		PTRDY#	115	in
EQ1	9	t/s		PTWR	108	out
EQ2	17	t/s		RD#	72	in
EQ3	21	t/s		RDEMPTY	106	out
EQ4/FWC#	29	t/s		RDFIFO#	104	in
EQ5/FRC#	33	t/s		REQ#	144	out
EQ6/AMREN	41	t/s		RST#	139	in
EQ7/AMWEN	49	t/s		SELECT#	75	in
ERD#/SCL	128	out		SERR#	26	o/d
EWR#/SDA	127	t/s		SNV	135	in
FLT#	138	in		STOP#	22	t/s
FRAME#	16	t/s		SYSRST#	126	out
GNT	143	in		TRDY#	19	t/s
IDSEL	160	in		VCC	11	V
INTA#	58	o/d		VCC	31	V
IRDY#	18	t/s		VCC	51	V
IRQ#	124	out		VCC	71	V
LOCK#	23	in		VCC	91	V
MODE	59	in		VCC	111	V
NC	136	—		VCC	131	V
NC	134	—		VCC	151	V
PAR	27	t/s		VSS	10	V
PERR#	24	t/s		VSS	30	V
PTADR#	107	in		VSS	50	V
PTATN#	114	out		VSS	70	V
PTBE0#	116	out		VSS	90	V
PTBE1#	118	out		VSS	110	V
PTBE2#	119	out		VSS	130	V
PTBE3#	120	out		VSS	150	V
PTBURST#	112	out		WR#	74	in
PTNUM0	123	out		WRFIFO#	102	in
PTNUM1	122	out		WRFULL	103	out

13

Table 13-2. S5933 Numerical Pin Assignment

Pin#	Signal	Type	Pin#	Signal	Type	Pin#	Signal	Type
1	EQ0	t/s	33	EQ5/FRC#	t/s	65	DQ25	t/s
2	AD23	t/s	34	AD14	t/s	66	ADR4	in
3	AD22	t/s	35	AD13	t/s	67	ADR3	in
4	AD21	t/s	36	AD12	t/s	68	ADR2	in
5	DQ31	t/s	37	DQ28	t/s	69	EA2	t/s
6	AD20	t/s	38	AD11	t/s	70	VSS	V
7	AD19	t/s	39	AD10	t/s	71	VCC	V
8	AD18	t/s	40	AD9	t/s	72	RD#	in
9	EQ1	t/s	41	EQ6/AMREN	t/s	73	EA3	t/s
10	VSS	V	42	AD8	t/s	74	WR#	in
11	VCC	V	43	C/BE0#	t/s	75	SELECT#	in
12	AD17	t/s	44	AD7	t/s	76	DQ15	t/s
13	DQ30	t/s	45	DQ27	t/s	77	DQ24	t/s
14	AD16	t/s	46	AD6	t/s	78	DQ14	t/s
15	C/BE2#	t/s	47	AD5	t/s	79	DQ13	t/s
16	FRAME#	t/s	48	AD4	t/s	80	DQ12	t/s
17	EQ2	t/s	49	EQ7/AMWEN	t/s	81	EA4	t/s
18	IRDY#	t/s	50	VSS	V	82	DQ11	t/s
19	TRDY#	t/s	51	VCC	V	83	DQ10	t/s
20	DEVSEL#	t/s	52	AD3	t/s	84	DQ9	t/s
21	EQ3	t/s	53	DQ26	t/s	85	DQ23	t/s
22	STOP#	t/s	54	AD2	t/s	86	DQ8	t/s
23	LOCK#	in	55	AD1	t/s	87	BE0#	in
24	PERR#	t/s	56	AD0	t/s	88	DQ7	t/s
25	DQ29	t/s	57	EA0	t/s	89	EA5	t/s
26	SERR#	o/d	58	INTA#	o/d	90	VSS	V
27	PAR	t/s	59	MODE	in	91	VCC	V
28	C/BE1#	t/s	60	BE3#	in	92	DQ6	t/s
29	EQ4/FWC#	t/s	61	EA1	t/s	93	DQ22	t/s
30	VSS	V	62	BE2#	in	94	DQ5	t/s
31	VCC	V	63	BE1#	in	95	DQ4	t/s
32	AD15	t/s	64	ADR5	in	96	DQ3	t/s

Table 13-2. S5933 Numerical Pin Assignment (Continued)

Pin#	Signal	Type	Pin#	Signal	Type
97	EA6	t/s	129	EA11	out
98	DQ2	t/s	130	VSS	V
99	DQ1	t/s	131	VCC	V
100	DQ0	t/s	132	ADR6	in
101	EA7	t/s	133	DQ18	t/s
102	WRFIFO#	in	134	NC	—
103	WRFULL	out	135	SNV	in
104	RDFIFO#	in	136	NC	—
105	DQ21	t/s	137	EA12	out
106	RDEMPTY	out	138	FLT#	in
107	PTADR#	in	139	RST#	in
108	PTWR	out	140	BPCLK	out
109	EA8	t/s	141	EA13	out
110	VSS	V	142	CLK	in
111	VCC	V	143	GNT	in
112	PTBURST#	out	144	REQ#	out
113	EA9	out	145	DQ17	t/s
114	PTATN#	out	146	AD31	t/s
115	PTRDY#	in	147	AD30	t/s
116	PTBE0#	out	148	AD29	t/s
117	DQ20	t/s	149	EA14/FWE	t/s
118	PTBE1#	out	150	VSS	V
119	PTBE2#	out	151	VCC	V
120	PTBE3#	out	152	AD28	t/s
121	EA10	out	153	EA15/FRF	t/s
122	PTNUM1	out	154	AD27	t/s
123	PTNUM0	out	155	AD26	t/s
124	IRQ#	out	156	AD25	t/s
125	DQ19	t/s	157	DQ16	t/s
126	SYSRST#	out	158	AD24	t/s
127	EWR#/SDA	t/s	159	C/BE3#	t/s
128	ERD#/SCL	out	160	IDSEL	in

13

13.2 Package Physical Dimensions
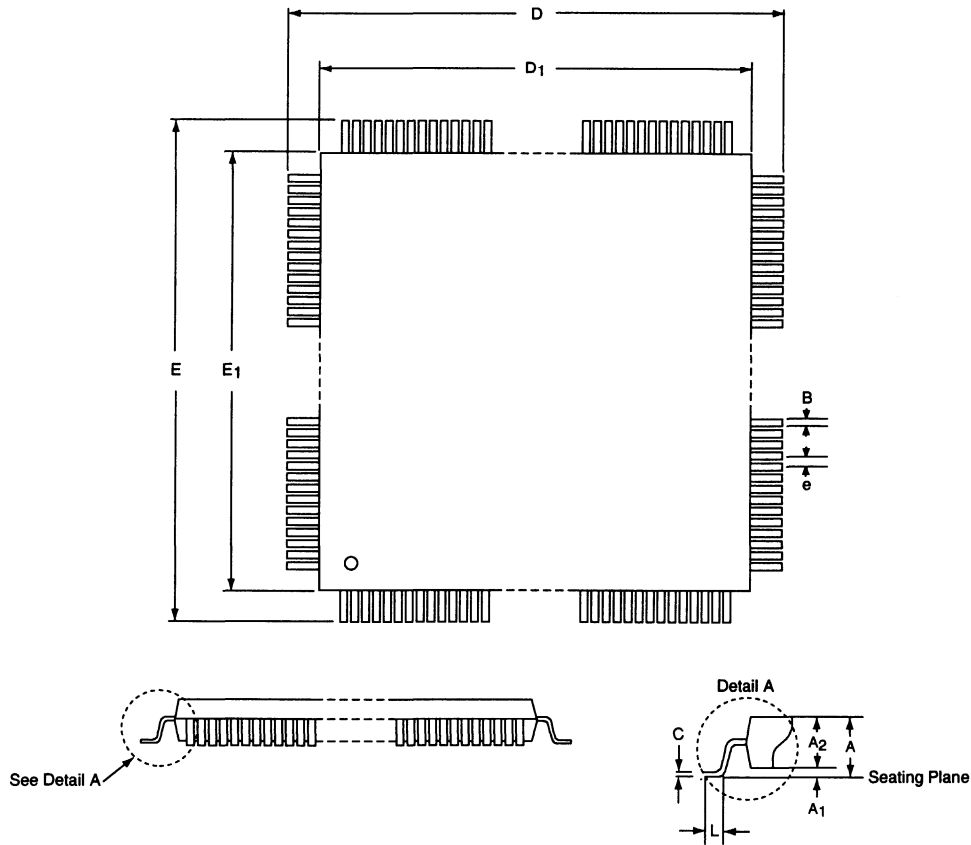
PQFP IN MILLIMETERS		
LEAD#	160	
SYMBOL	MIN	MAX
A	—	4.07
A1	0.25	—
A2	3.17	3.87
B	0.22	0.38
C	0.15	0.20
D1	27.90	28.10
E1	27.90	28.10
e	0.65	BSC
D	31.65	32.15
E	31.65	32.15
L	0.65	0.95

13.3 Ordering Information

```
    S5933            Q          B


                                      ┌──────  Revision Level
                                      │        Contact AMCC for
                                      │        latest Revision Letter

                                      │        Package Option
                             └────────┘        Q = 160-pin PQFP

                  └────────────────────────────  Device Number
```
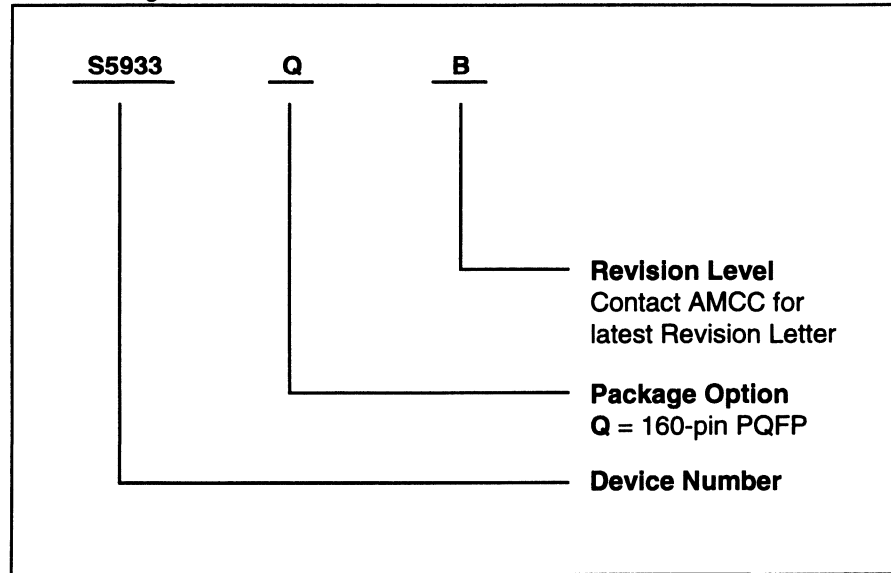
Application Notes A

Careful attention must always be exercised when beginning the design and layout phase of any new printed circuit board. It is only through careful planning and the usage of good electrical design practices that long term product reliability may be achieved. Careful attention to layout issues of EMI, cross-talk and decoupling will avoid engineering prototype development problems and future production failures due to an "on the edge" design.

Designs incorporating PCI devices have specific printed circuit board layout requirements in order to comply with industry standard PCI local bus specifications. In addition to these requirements, AMCC's S5933 PCI controller has specific requirements to ensure proper function and long life. This applications note is supplied by AMCC as factory recommended PCB design layout guidelines for printed circuit boards incorporating the S5933 PCI controller. The following sections detail important design areas concerning PCB design, power decoupling, ferrite beads and critical trace layout.

PCB DESIGN

AMCC highly recommends the use of a four layer printed circuit board. This is due to the expected high trace density in most PCI designs incorporating PLCC and PQFP devices. Four layer designs significantly overcome ground noise problems associated with most two layer PCBs. Should a two layer design be implemented, leave as much copper on the PCB as possible for all power distribution traces. This is extremely important in ground traces to avoid ground loops and ground potential problems. Also utilize multiple feed-thrus for large traces. A 100 mil trace with a single 0.030 feed thru has it's current carrying capability significantly reduced.

DECOUPLING

The AMCC PCI controller is an application specific standard product (ASSP) utilizing CMOS technology. Although CMOS technology is commonly known for it's noise immunity properties, special consideration must still be given to electrical noise generated due to the high speed signals utilized in a PCI design.

The first design issue of concern is decoupling. By "decoupling", the designer provides a means to dissociate circuit functions from the power bus serving that circuit. This "means" is provided through the usage of decoupling capacitors, ferrite beads and proper printed circuit board trace layout. The lack of correct decoupling increases both radiated and conducted emissions which increases electrical noise and susceptibility to circuit failure (i.e. erratic and intermittent circuit functions or "FLAKYNESS").

To reduce these problems, AMCC recommends PCI designs incorporate a low speed PCB decoupling capacitor. Select a 10uF (minimum) tantalum or metalized polycarbonate (not aluminum) capacitor for this purpose. Specify a low ESR type at a working voltage slightly above the circuit's operating voltage. Locate the capacitor no more than 300 mils from the PCB's power entry point as shown in figure 1. This point is likely the PCB edge fingers or a wire connector interfacing the PCB to the system power supply.

AMCC also recommends the use of one high speed 0.1uF decoupling capacitor per PCB IC package. Specify an X7R or BX type dielectric ceramic chip capacitor also rated slightly above required working voltage for this purpose. Locate each capacitor next to and on the same side of the PCB as each IC device. Locate capacitors no more than 150 mils from each IC package's ground pin. High speed decoupling for the S5933 PCI controller IC requires one 0.1uF per power and ground pin pair. IMPORTANT: Locate these capacitors on the same side of the PCB as the S5933 at a distance of less than 150 mils as shown in Figure 1.

The use of the above capacitors will sufficiently decouple the Vcc and ground planes from high and low speed circuit functions. One last decoupling issue of concern involves any unused PCI edge connector power fingers. Industry PCI specifications provides for power sources of +3.3V and +5V within the PCI edge connector. PCI specifications require any unused power and V I/O pins on the PCI edge connector be decoupled to the ground plane with an average of 0.01uF.

A

FERRITE BEADS

In many applications the requirement for ferrite beads to reduce high frequency noise is unnecessary once the above indicated board layout and by-pass practices have been followed. However, in some applications high frequency noise may persist. In these cases the addition of a ferrite bead to the input +V power entry point as shown in figure 1 may be necessary. The size and type of material used will vary depending on the particular design and electrical noise to be eliminated. Consult the bead manufacturer data books to determine the best fit. Expect to try a small selection of beads to achieve best results. Use a single pass thru or one turn to increase effectiveness of the bead. **DO NOT** series beads to increase impedance. Use a cracked air gap bead to reduce saturation effects as necessary.

CRITICAL TRACE LAYOUT

AMCC's S5933 PCI controller is very powerful and flexible. It operates at clock and data bus speeds of up to 33 megahertz. Data bus transfer operations can occur in 30 nanoseconds with signal rise and fall times of 3 nanoseconds. At these speeds, signal traces look more like transmission lines where trace impedance becomes an important factor. Carefull attention to trace lengths and routing will prevent impedance caused ringing and will preserve signal rise and fall times. As a last design issue, standard PCI specifications require the following design criteria be adhered to for all PCI bus controller devices.

1. Trace lengths for each 32 bit interface data signal from the S5933 to the PCI bus is limited to a maximum of 1.5 inches for all 32 bit and 64 bit cards.

2. Trace lengths for the balance of the PCI bus signals, used in the 64 bit extension, is limited to a maximum of 2.0 inches from the S5933 to the PCI bus.

3. The trace connecting the S5933 PCI clock signal (S5933 pin 142) to the PCI bus connector must be 2.5 inches +/- .1 inches in length and can be routed to only one load. It may be necessary to "snake" this trace, as shown in figure 1, depending on the physical location of the PCI controller IC on the PCB to ensure this requirement is met. Ensure all corners of this trace are rounded. Do not use 90 degree sharp corners.

4. The unloaded impedance of a shared PCI signal trace on the expansion card must be held within a 60 to 100 ohm range. On a typical 4 layer glass epoxy PCB, maintaining a signal trace width of 20 to 30 mils will satisfy this requirement.

Figure 1. PCI Evaluation Board Signal Layer #1
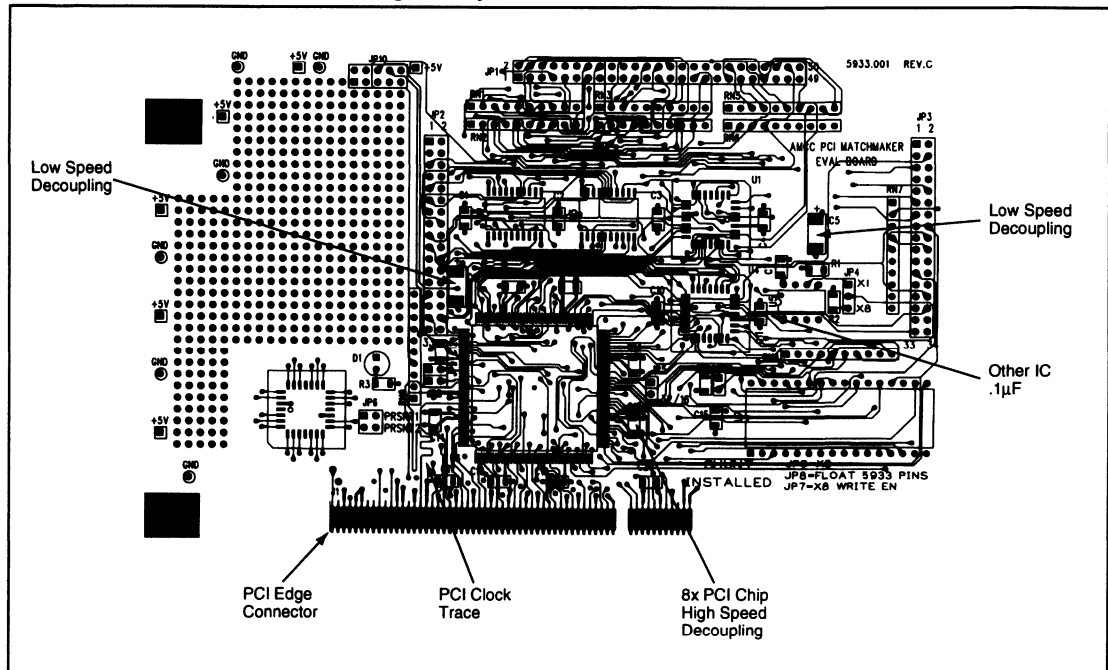
AMCC ®

1.0 OVERVIEW

The AMCC S5933 PCI Controller provides add-in cards with bus mastering capabilities on the PCI bus. The S5933 internal FIFO is used to transfer data between the add-on and the PCI bus as a PCI initiator (bus master). The S5933 allows burst transfers at the full PCI bandwidth.

This application note discusses how the S5933 may be used in a PCI bus mastering (DMA) application. A brief background of DMA as it relates to the ISA bus is provided in Section 2. The remaining sections describe the bus master capabilities of the S5933 including add-on hardware support and required software support. Performance for bus mastering applications on the PCI bus is also discussed. An example C-Language program is provided in Appendix A to set up PCI DMA transfers with the S5933 controller.

2.0 DMA BACKGROUND

In ISA bus-based personal computers, there were two potential bus masters: the host CPU and the system DMA controller. There was no protocol defined for alternate bus masters to gain control of the bus and transfer data to other cards or platform memory. Most ISA add-in cards were designed as ISA bus slaves. The DMA controller or the CPU were used for data transfers to and from ISA cards.

For performance reasons, some ISA add-in cards required bus mastering capabilities. To become an ISA bus master, a free DMA channel was utilized. The ISA card asserted a DMA request to the 8237 DMA controller, the 82C37 asserted the HOLD input to the CPU. When the 82C37 received the HLDA acknowledge back from the CPU, it asserted an acknowledge to the add-in card. The add-in card then had control of the ISA bus to perform data transfers. For this reason, bus mastering is also referred to as DMA.

The PCI bus protocol has specific provisions to allow bus mastering (DMA) by any device connected to the PCI bus. If add-in cards perform their own data transfers, the host CPU is freed to perform other tasks (code execution, etc.). The PCI bus provides a dedicated request (REQ#)/grant(GNT#) pair to each PCI bus device (or card slot). These are all routed to a central bus arbiter that determines which device controls the PCI bus, based on a predefined priority scheme.

3.0 S5933 ARCHITECTURE

The S5933 performs DMA (bus master) transfers on the PCI bus through its FIFO interface. It has two, independent FIFOs. Each FIFO is 8-deep by 32-bits wide. One is used to transfer data from the PCI bus to the add-on, and the other is used to transfer data from the add-on to the PCI bus. The S5933 only performs DMA transfers to and from memory-mapped targets.

Each FIFO has an associated address and transfer count register. These registers may be defined by either the host CPU or add-on logic (configurable at reset). Each FIFO has a programmable priority and management scheme. Each FIFO also has the ability to generate interrupts when the transfer count expires or error conditions occur during a PCI bus transfer.

3.1 Configuring the S5933 for DMA Transfers

A DMA (bus master) transfer may be set up by either the host CPU, or add-on logic. This is defined for the S5933 at reset and cannot change during operation. Initiating a DMA transfer involves setting up the source/destination addresses and transfer counts as well as enabling the transfer. The FIFO relative priorities and FIFO management schemes are always programmed by the host CPU.

If an external, non-volatile boot memory is used with the S5933, the contents of offset 45h define FIFO operation. The following bits functions are defined:

Bit 7 Bus Master Register Access	
0	Address and transfer count registers only accessible from the add-on interface
1	Address and transfer count registers only accessible from the PCI interface (default)
Bit 6 RDFIFO# or RD# Operation	
0	Synchronous Mode — RDFIFO# and RD# functions as enables
1	Asynchronous Mode — RDFIFO# and RD# functions as clocks (default)
Bit 5 WRFIFO# or WR# Operation	
0	Synchronous Mode — WRFIFO# and WR# functions as enables
1	Asynchronous Mode — WRFIFO# and WR# functions as clocks (default)

A

If no external non-volatile boot memory is used with the S5933, the default configuration for bus mastering is for transfers to be set up by the host CPU. Bits 6 and 5 define how the add-on FIFO interface operates. The default configuration for FIFO read and write strobes is to be asynchronous to the PCI clock (provided to the add-on by the S5933 BPCLK output). For more information on the FIFO add-on bus interface, refer to the S5933 PCI Controller Data Book.

3.1.1 PCI Initiated DMA Transfers

If no external non-volatile boot device is used, or if location 45h, bit 7 is 1, the address and transfer count registers are only accessible from the PCI bus interface. This requires that the PCI host write these register to initiate a DMA transfer. In this configuration, the S5933 can be programmed to generate a PCI interrupt (INTA#) when the DMA transfer count reaches zero or when an error occurs during a DMA transfer (target or master abort conditions).

PCI initiated DMA transfers must be enabled through the Bus Master Control/Status Register (MCSR). The register is at offset 3Ch in the S5933 PCI Operation Registers. S5933 Read DMA transfers and Write DMA transfers have separate control bits and can be individually enabled. The enable bits are not automatically cleared upon completion of a DMA transfer. DMA transfers remain enabled until these bits are cleared by the host CPU.

3.1.2 Add-on Initiated DMA Transfers

If an external, **serial** non-volatile boot device is used and location 45h, bit 7 is 0, the address and transfer count registers are only accessible from the add-on bus interface. This requires that add-on logic write these register to initiate a DMA transfer. In this configuration, the S5933 can be programmed to generate an add-on interrupt (IRQ#) when the DMA transfer count reaches zero or when an error occurs during a DMA transfer. A serial boot device is required because the transfer enable inputs used for add-on initiated DMA are multiplexed with the byte-wide nv-memory interface.

Add-on initiated DMA transfers are enabled using the AMREN and AMWEN inputs. The bus master enable bits in the Bus Master Control/Status Register (MCSR) are ignored. Asserting AMREN enables the S5933 to request control of the PCI bus for a PCI read transfer when the appropriate FIFO conditions are met. Asserting AMWEN enables the S5933 to request control of the PCI bus for a PCI write transfer when the appropriate FIFO conditions are met (see section 3.4 on FIFO management schemes). If an

enable is deasserted during a DMA transfer, the current PCI bus transaction completes and the S5933 deasserts REQ#, giving up control of the PCI bus.

3.2 DMA Address Registers

There are two DMA address registers: the Master Write Address Register (MWAR) and the Master Read Address Register (MRAR). These registers are located at offsets 24h and 2Ch, respectively, in the S5933 PCI Operation Registers. These are written with the beginning memory address of the DMA transfer. The S5933 requires that DMA transfers start on double-word boundaries (A1 and A0 = 0).

During a DMA transfer, the address registers are incremented by four after each completed data phase. If a PCI target disconnects and requests a retry from the S5933, the correct address is maintained to allow the transfer to begin from where it was disconnected.

3.3 DMA Transfer Count Registers

There are two DMA transfer count registers: the Master Write Transfer Count Register (MWTC) and the Master Read Transfer Count Register (MRTC). These registers are located at offsets 28h and 30h, respectively, in the S5933 PCI Operation Registers. These are written with the a DMA transfer byte count of up to 64 Mbytes. The transfer count registers are decremented by four after each completed data phase. The transfer count registers do not reset to their initial value after reaching zero, the PCI host must reprogram them.

Although S5933 DMA transfer must begin on double-word boundaries, the transfer count does not have to be a multiple of four bytes. When the transfer count decrements below four, only the byte enable corresponding to the number of bytes left are asserted. For example, if a transfer count of 10 was programmed into one of the transfer count registers, two data transfers would complete with all byte enables asserted (8 bytes). The final data transfer would have only BE0# and BE1# asserted, transferring the final two bytes.

For add-on initiated DMA transfers, the transfer counts are enabled or disabled through the Add-on General Control/Status Register (AGCSTS), bit 28. The transfer counts for read and write transfers cannot be individually enabled. This may be useful in applications where the amount of data to be transferred is not known. If transfer counts are disabled, the S5933 continues to transfer data according to the conditions listed above, but transfer counts are ignored.

3.4 FIFO Management Schemes

The S5933 provides flexibility in how the FIFO is managed for DMA transfers. The FIFO management scheme determines when the S5933 requests the PCI bus (asserts REQ#). The most efficient way to utilize the capabilities of the PCI bus is with burst transfers. Requesting the PCI bus every time the FIFO contains a single double-word is an ineffecient use of the bus, and limits the performance of other PCI devices within a system. It is more desirable request the bus when multiple operations are required, allowing the S5933 to perform a burst transfer.

The management scheme is configurable for the PCI to add-on and add-on to PCI FIFOs (and may be different for each). Bus mastering must be enabled for the management scheme to apply (via the MCSR enable bits or AMREN/AMWEN). The FIFO management option is programmed through the Bus Master Control/Status Register (MCSR).

For the PCI to add-on FIFO (DMA reads), there are two options. The FIFO can be programmed to request the bus when any FIFO location is empty or only when four or more locations are empty. After the S5933 is granted control of the PCI bus, the management scheme does not apply. The device continues to read as long as there is an open FIFO location. For DMA read transfers, the S5933 maintains control of the PCI bus until one of the following events:

- The read transfer count (MRTC) reaches zero
- Bus mastering is disabled (with the MSCR enable bit or AMREN)
- Another master requests the bus and the Latency Timer is expired
- The PCI target aborts the transfer
- The PCI to add-on FIFO becomes full

For the add-on to PCI FIFO (DMA writes), there are two management options. The FIFO can be programmed to request the bus when any FIFO location is full or only when four or more locations are full. After the S5933 is granted control of the PCI bus, the management scheme does not apply. The device continues to write as long as there is data in the FIFO. For DMA write transfers, the S5933 maintains control of the PCI bus until one of the following events:

- The write transfer count (MWTC) reaches zero
- Bus mastering is disabled (with the MSCR enable bit or AMWEN)

- Another master requests the bus and the Latency Timer is expired
- The PCI target aborts the transfer
- The add-on to PCI FIFO becomes empty

There are two special cases for the add-on to PCI FIFO management scheme. The first case is when the FIFO is programmed to request the PCI bus only when four or more locations (16 bytes) are full, but the transfer count is less than 16 bytes. In this situation, the FIFO ignores the management scheme and finishes transferring the data. The second case is when the S5933 is configured for add-on initiated bus mastering. In this situation, the FIFO management scheme must be set to request the PCI bus when one or more locations are full.

3.5 S5933 DMA Channel Priority

In many applications, the S5933 performs both DMA read and write transfers. This requires a priority scheme be implemented between the two FIFOs. If the FIFO management condition for initiating a PCI read and a PCI write are both met, a method must exist to determine which transfer is performed first.

Bits D12 and D8 in the Bus Master Control/Status Register (MCSR) control the read and write DMA channel priority, respectively. If these bits are both set or both clear, priority alternates, beginning with read transfers. If the read priority is set and the write priority is clear, read cycles take priority. If the write priority is set and the read priority is clear, write cycles take priority. Priority arbitration is only done when neither FIFO has control of the PCI bus (the PCI to add-on FIFO never interrupts an add-on to PCI FIFO transfer in progress and vice-versa).

3.6 S5933 DMA Interrupts

The S5933 can generate interrupts under the following conditions: the read transfer count reaches zero, the write transfer count reaches zero, or an error occurs on the PCI bus during a DMA transfer.

Which interface (PCI bus or add-on bus) receives the interrupt is determined by which side initiated the DMA transfer. If PCI initiated DMA transfers are used, a PCI INTA# interrupt is generated when an interrupt condition is met. If add-on initiated DMA transfers are used, IRQ# is generated to the add-on interface. Interrupts are optional and may be disabled.

A

3.6.1 Transfer Count Interrupts

Transfer count interrupts may come from two sources: read transfer count and/or write transfer count. One or both interrupts may be enabled, or both may be disabled. A read transfer count interrupt is generated when the Master Read Transfer Count Register (MRTC) decrements to zero. A write transfer count interrupt is generated when the Master Write Transfer Count Register (MWTC) decrements to zero. These registers decrement when a PCI transfer completes successfully.

If add-on initiated DMA transfers are used with transfer counts disabled, these interrupt sources are disabled.

3.6.2 PCI Bus Error Condition Interrupts

In some situations, the PCI bus may signal an error condition during an S5933 DMA transfer. These error conditions include a target abort or master abort on the PCI bus. If one of these conditions exists, it is important to notify the device which initiated the transfer that it cannot complete successfully. Interrupts on PCI error conditions are only enabled if one or both of the transfer count interrupts are enabled. There is no individual enable bit for PCI error interrupts.

A PCI target abort indicates an error condition where no number of retries to the target will result in a successful data transfer. A PCI master abort occurs when a PCI bus master (the S5933, in this case) attempts to access a PCI target which is either non-existent or disabled. In either of these situations, the device which set up the DMA transfer is notified with an interrupt (either INTA# or IRQ#).

3.6.3 Controlling S5933 DMA Interrupts

For PCI initiated DMA transfers, interrupts are enabled through the S5933 Interrupt Control Status Register (INTCSR). This register is located at offset 38h in the S5933 PCI Operation Registers. INTCSR is also accessed during interrupt service routines to determine the interrupt source and clear the interrupt. The following INTCSR bits relate to DMA Operations:

Bit 14	**Enable Interrupt on Write Transfer Complete.** If set, INTA# is generated when MWTC decrements to zero during a DMA transfer.
Bit 15	**Enable Interrupt on Read Transfer Complete.** If set, INTA# is generated when MRTC decrements to zero during a DMA transfer.

Bit 18	**Write Transfer Complete Interrupt.** When set, this bit indicates MWTC has decremented to zero and INTA# has been asserted. Writing a one to this bit clears the interrupt source and deasserts INTA#. Writing a zero to this bit has no effect.
Bit 19	**Read Transfer Complete Interrupt.** When set, this bit indicates MRTC has decremented to zero and INTA# has been asserted. Writing a one to this bit clears the interrupt source and deasserts INTA#. Writing a zero to this bit has no effect.
Bit 20	**Master Abort Interrupt.** When set, this bit indicates that the S5933 had to perform a master abort and INTA# has been asserted. Writing a one to this bit clears the interrupt source and deasserts INTA#. Writing a zero to this bit has no effect.
Bit 21	**Target Abort Interrupt.** When set, this bit indicates that the S5933 received a target abort and INTA# has been asserted. Writing a one to this bit clears the interrupt source and deasserts INTA#. Writing a zero to this bit has no effect.

For add-on initiated DMA transfers, interrupts are enabled through the S5933 Add-on Interrupt Control/Status Register (AINT). This register is located at offset 38h in the S5933 Add-on Operation Registers. AINT is also accessed during interrupt service routines to determine the interrupt source and clear the interrupt. The following AINT bits relate to DMA Operations:

Bit 14	**Enable Interrupt on Write Transfer Complete.** If set, IRQ# is generated when MWTC decrements to zero during a DMA transfer.
Bit 15	**Enable Interrupt on Read Transfer Complete.** If set, IRQ# is generated when MRTC decrements to zero during a DMA transfer.
Bit 18	**Write Transfer Complete Interrupt.** When set, this bit indicates MWTC has decremented to zero and IRQ# has been asserted. Writing a one to this bit clears the interrupt source and deasserts IRQ#. Writing a zero to this bit has no effect.

Bit 19   Read Transfer Complete Interrupt.
When set, this bit indicates MRTC has decremented to zero and IRQ# has been asserted. Writing a one to this bit clears the interrupt source and deasserts IRQ#. Writing a zero to this bit has no effect.

Bit 21   Bus Master Error Interrupt. When set, this bit indicates that the S5933 had to perform a master abort or received a target abort and IRQ# has been asserted. Writing a one to this bit clears the interrupt source and deasserts IRQ#. Writing a zero to this bit has no effect.

3.6.4   PCI Interrupt Considerations

If the S5933 is configured to generate PCI bus interrupts (INTA#) for DMA transfer counts and PCI bus error conditions, considerations must be made for the interrupt handler. The interrupt vector must be obtained, and because hardware interrupts may be shared, interrupt handlers may have to be "chained."

3.6.4.1   Enabling PCI Interrupts

The Interrupt Pin (INTFIN) PCI Configuration Register may be loaded out of non-volatile memory offset 7Dh by the S5933 at reset. The value loaded into this register identifies which PCI interrupt: INTA#, INTB#, INTC#, or INTD# is used. The default value is 01h, identifying INTA#. For the S5933, the INTA# output should be connected to the PCI bus INTA# pin. If PCI interrupts are not required, this register may be initialized to 00h (interrupts disabled) or ignored.

The four PCI interrupts are mapped within the system chipset to standard PC IRQ numbers (0-15). The 82C59A compatible interrupt controllers (two cascaded controllers) each have an interrupt mask register. The master mask register, located at system I/O location 21h controls the masking of IRQ0-7, and the slave mask register, located at system I/O location 1Ah controls the masking of IRQ8-15. Application software must make sure the S5933 interrupt line (indicated by the PCI Interrupt Line Register described in Section 3.6.4.2) is unmasked. If the interrupt is masked, the handler never executes.

3.6.4.2   PCI Host Interrupt Handlers

The S5933 Interrupt Line (INTLN) PCI Configuration Register is loaded out of non-volatile memory offset 7Ch by the S5933 at reset. The value loaded into the register is a hardware interrupt number for the host interrupt controller (IRQ0 to 15). This value may be used by the host, or the BIOS may overwrite it with its own value.

Multiple PCI devices may be assigned to a single hardware interrupt by the host. PCI device drivers are, therefore, required to determine if the current interrupt was generated by the device it services. If not, it must pass control, or "chain" the previous interrupt handler to service the interrupt.

Each application's code must install its own interrupt vector. To do this, the Interrupt Line Configuration Register is read. A value between 0 and 15 is returned, corresponding to a PC hardware IRQ number. This must be translated into a software interrupt number. The following table shows the conversions for a PC:

Hardware Interrupt	Software Interrupt
IRQ0	08h
IRQ1	09h
IRQ2	0Ah
IRQ3	0Bh
IRQ4	0Ch
IRQ5	0Dh
IRQ6	0Eh
IRQ7	0Fh
IRQ8	70h
IRQ9	71h
IRQ10	72h
IRQ11	73h
IRQ12	74h
IRQ13	75h
IRQ14	76h
IRQ15	77h

Once the software interrupt number is calculated, the previous interrupt handler's vector can be read and stored. The new interrupt vector is then installed. In this manner, numerous devices within a system can share a single hardware interrupt.

A

Each application's interrupt handler must first check the source of the interrupt. For S5933 applications, this is done by reading the S5933 Interrupt Control/Status Register. The status of all possible S5933 PCI interrupt sources is indicated by this register. If the status bits indicate that no interrupts were generated by the S5933, the handler must call the previous interrupt handler whose vector it replaced. The previous interrupt handler then performs a similar task for the device it services, and this process continues until the device which generated the interrupt is found.

If the interrupt handler determines that the source of the interrupt was the S5933, the interrupt source must be cleared through the Interrupt Control/Status Register (INTCSR). The handler then performs whatever tasks are necessary to service the interrupt (such as rewriting address or transfer count registers). Finally, the handler must clear the 83C59A interrupt controller 'in-service' bit. This bit is set when the host processor acknowledges the interrupt and jumps to the interrupt service routine. A specific end-of-interrupt (EOI) is used to clear this bit. If a PCI device is mapped to hardware interrupts IRQ8 to IRQ15, two EOI commands must be issued. One EOI must be issued for the slave 82C59A which supports IRQ8-15. A second EOI is required for the master 82C59A. The second EOI is a specific EOI for IRQ2 because the slave interrupt controller in a PC is cascaded into the IRQ2 input of the master interrupt controller. Without the end-of-interrupt sequence, the interrupt controllers will not recognize further interrupts from that source.

4.0 PCI DMA PERFORMANCE FACTORS

There are a number of factors which determine DMA performance on the PCI bus. The clock speed and data bus width are important in determining maximum bus bandwidth. The most important factor in DMA performance is traffic on the PCI bus. As the number of PCI devices which require access to the bus increases, the bandwidth available to each individual device decreases.

4.1 PCI Bus Arbitration

Each device on the PCI bus has a dedicated REQ#/GNT# pair which are connected to the system bus arbiter. When a device asserts REQ#, it indicates a data transfer is required. The transfer may be a single data phase or a burst. The bus arbiter asserts GNT# to the device to indicate that it may now perform the transfer.

The bus arbiter may remove GNT# from a PCI bus master on any PCI clock. The current transaction completes, and the PCI master gives up control of the bus. GNT# may already be asserted to the next master, but it is not allowed to drive the PCI bus until IRDY# and FRAME# are deasserted, indicating the bus is idle. If the original bus master has more data to be transferred, it may keep REQ# asserted, but must wait for GNT# again.

The priority scheme used to determine which PCI device controls the bus at a given time is determined by the system. The PCI specification requires a few extra Configuration Registers within PCI bus master devices. During system initialization, these registers are read to determine each device's requirements (if any). Based on these, a priority scheme can be defined which is unique to that system. Ideally, the arbitration scheme gives priority to devices with higher bandwidth requirements, but does not prevent other PCI bus masters from gaining control of the bus.

4.2 PCI Bus Access Latency

There are three components to latency on the PCI bus. The total latency is measured from the time a bus master requests the bus (asserts REQ#) to when the target of the transfer completes the transfer (asserts TRDY#). Each of these components is described in the following sections.

4.2.1 Arbitration Latency

Once a PCI device asserts REQ#, it must wait for the bus arbiter to assert GNT#. This delay is called arbitration latency. This is determined by the priority scheme used by the bus arbiter, the relative priority of the device requesting the bus, and the amount of activity on the PCI bus.

4.2.2 Bus Acquisition Latency

Once a PCI device receives GNT# from the arbiter, it must wait for the current bus master to complete it's transaction (indicated by FRAME# and IRDY# deasserted) before driving the PCI bus. Bus acquisition latency is the time from when GNT# is received to when FRAME# is asserted by a particular master. This is determined by the length of the current bus master's transfer.

4.2.3 Target Latency

After a PCI device begins a bus cycle, it must wait for the target of the transfer to complete the cycle (by asserting TRDY#). Target latency is the time from when FRAME# is asserted to when TRDY# is asserted. This is unique for each target and depends

on the function of the target (main memory, VGA controller, network interface, etc.).

4.3 PCI Configuration Registers

A PCI device with bus mastering capabilities may implement up to three additional configuration registers. These registers indicate the requirements of a particular PCI device. These are read during system initialization and may be used to define a bus master priority scheme. The S5933 implements all of these registers. They can be boot loaded from an external, non-volatile memory device at reset.

4.3.1 Latency Timer Register

If a bus master is capable of PCI bursts of more than two data phases, this register is required. The Latency Timer Register defines the number of PCI clocks that the S5933 is guaranteed control of the bus. The Latency Timer Register is shown in Figure 1. The value programmed in this register is decremented once every 8 PCI clocks after the S5933 asserts FRAME#.

The default value for this register is 00h, indicating that the S5933 has no minimum transfer length requirement. The system can overwrite the Latency Timer Register with any value. This prevents an individual PCI master from controlling the bus for an extremely long period.

If no other PCI master has requested the bus, the Latency Timer value does not matter (even if it has expired). The S5933 transfers data until the transaction is complete. If another bus master requests the bus and receives GNT# while the S5933 is transferring data, three situations can occur:

1) If the Latency Timer has not expired, and the S5933 completes its transaction before it expires, the transaction completes normally.

2) If the Latency Timer has not expired, the S5933 transfers data until the latency timer expires where it completes the current data phase, terminates the transaction and gives up control of the bus

3) If the Latency Timer has already expired, the S5933 completes the current data phase, terminates the transaction, and gives up control of the bus.

4.3.2 Minimum Grant Register

This register defines how long of a burst period the device typically requires (in units of 250 ns). This read-only register is for information only. It may be used by the system, in conjunction with the Maximum Latency register, to define a PCI bus arbitration scheme (if the bus arbiter is programmable).

A value of zero (default for the S5933) indicates there is no strict requirement for burst length. The Minimum Grant Register is shown in Figure 2.

Figure 1. Latency Timer Register Definition
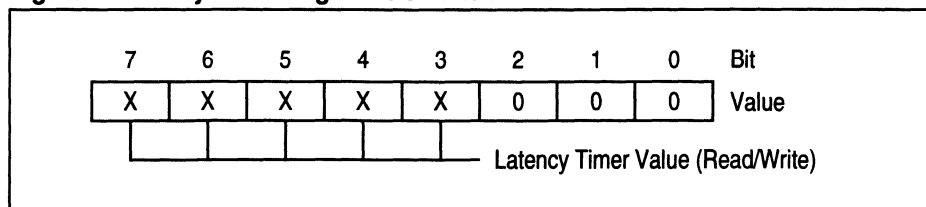
Figure 2. Minimum Grant Register Definition
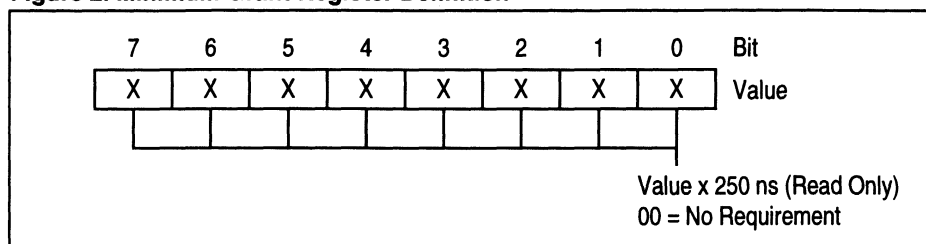
A

Figure 3. Maximum Latency Register Definition

7	6	5	4	3	2	1	0	Bit
X	X	X	X	X	X	X	X	Value

Value x 250 ns (Read Only)
00 = No Requirement
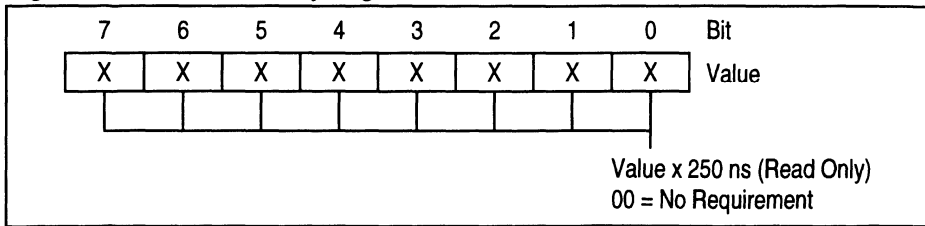
4.3.3   Maximum Latency Register

This register defines how often the device typically needs PCI bus access (in units of 250 ns). This read-only register is for information only. It may be used by the system, in conjunction with the Minimum Grant register, to define a PCI bus arbitration scheme (if the bus arbiter is programmable).

A value of zero (default for the S5933) indicates there is no strict requirement for access to the PCI bus. The Maximum Latency Register is shown in Figure 3.

4.4   Sample PCI Performance Calculation

The maximum theoretical bandwidth for the PCI bus at 33 MHz is 132 Mbytes per second. The actual bandwidth is less. Achievable bandwidth depends of factors such as bus utilization by other masters, the bus arbitration scheme, and burst length limitations of both the PCI initiator and target. The following examples show bandwidth calculations for two situations: a DMA transfer from the S5933 to main DRAM memory, and a DMA transfer to another S5933 add-in board. These performance calculations are based on the current PCI systems. As chipsets, memory controllers and other PCI devices evolve, performance will increase, accordingly.

4.4.1   S5933 Burst to Main DRAM Memory

Table 1 shows a situation where an add-on device can write data to the S5933 FIFO at a rate of one double-word (32-bits) every 60 ns. The target for the DMA transfer is main DRAM memory. The PCI memory controller allows 4 data phase write bursts. The fifth data phase receives a target requested re-try. The PCI Specification requires that a PCI initiator deassert REQ# for two clocks. For this example, a 4 clock latency period is used from the reassertion of REQ# by the S5933 to the reassertion of GNT# by the PCI bus arbiter.

The sequence shown assumes the following initial conditions:

- Master Write Address Register (MWAR) = 100000h

- Master Write Transfer Count Register (MWTC) is disabled

- Bus mastering for the S5933 is already enabled

- The Add-on to PCI FIFO is full (8 dwords)

- The PCI bus arbiter has just asserted GNT# to the S5933

Table 1. Sample S5933 Burst to Main Memory

Time	PCI Bus Activity	Add-on Bus Activity	FIFO Status	REQ#	GNT#
30 ns	Address = 100000h	Idle	8 dwords	0	0
60 ns	Data Transfer 1	Wait State	7 dwords	0	0
90 ns	Data Transter 2	Write FIFO	7 dwords	0	0
120 ns	Data Transfer 3	Wait State	6 dwords	0	0
150 ns	Data Transfer 4	Write FIFO	6 dwords	0	0
180 ns	Target Disconnect	Wait State	6 dwords	0	0
210 ns	Idle (or other master)	Write FIFO	7 dwords	1	1
240 ns	Idle (or other master)	Wait State	7 dwords	1	1
270 ns	Idle (or other master)	Write FIFO	8 dwords	0	1
300 ns	Idle (or other master)	Idle	8 dwords	0	1
330 ns	Idle (or other master)	Idle	8 dwords	0	1
360 ns	Idle (or other master)	Idle	8 dwords	0	1
390 ns	Idle (or other master)	Idle	8 dwords	0	0
420 ns	Address = 100010h	Idle	8 dwords	0	0
450 ns	Data Transfer 5	Wait State	7 dwords	0	0

The sequence would repeat every 390 ns with a 33 MHz PCI bus clock. Four Dwords, or 16 bytes have been transferred to main memory in during this time. This would average out to about 40 MBytes/sec.

The major factor in the calculation is how long it takes for the S5933 to regain control of the PCI bus after the main memory controller disconnects after the fourth data phase. This depends on the number of PCI devices using the bus. Because the transfer is to main memory, the S5933 must compete for bus bandwidth with numerous other devices that are usually on the Primary PCI bus. A typical system may have the host CPU, a VGA controller, an ethernet interface, and a SCSI or IDE drive sharing the Primary PCI bus. If a rotational priority scheme were implemented by the bus arbiter, it could be significantly more than 4 PCI clocks before the S5933 regains control of the bus.

4.4.2 S5933 Burst to Another S5933 PCI Card

Table 2 shows a situation where an add-on device can write data to the S5933 FIFO at a rate of one double-word (32-bits) every 60 ns. The target for the DMA transfer is the pass-thru interface of another S5933 device which can accept data at a rate of one double-word every 30 ns. The S5933 pass-thru interface does not disconnect from a burst write unless the add-on has not read the pass-thru data register within 16 PCI clocks for the first data phase or 8 PCI clocks on successive data phases. In this situation, the initiator S5933 deasserts request because it runs out of data. The target S5933 never disconnects in this situation.

The sequence shown assumes the following initial conditions:

- Master Write Address Register (MWAR) = 100000h

- Master Write Transfer Count Register (MWTC) is disabled

- Bus mastering for the S5933 is already enabled

- The Add-on to PCI FIFO is full (8 dwords)

- The PCI bus arbiter has just asserted GNT# to the S5933

In this example, it is assumed that the AMWEN signal has been deasserted when the FIFO goes empty at 480 ns. This allows the FIFO to refill before another transfer is initiated. This is the most efficient way to utilize the PCI bus. Sending a signal, long burst is better than sending numerous short bursts because less overall time is spent arbitrating for PCI bus control.

The FIFO would be full again at 690 ns. If AMWEN is reasserted when the FIFO is full, and a 4 clock latency is assumed (as with the previous example), the next address phase begins at 810 ns. The process would repeat from there. This results in 15 dwords (60 bytes) being transferred every 810 ns. The results in an average 74 MByte/sec. transfer rate. Again, this is heavily dependent on PCI bus utilization by other devices. Unless the two cards in question share an isolated PCI bus on the secondary side of a PCI to PCI bridge, bandwidth would likely be less than this.

Table 2. Sample S5933 Burst to Another S5933 Device

Time	PCI Bus Activity	Add-on Bus Activity	FIFO Status	REQ#	GNT#
30 ns	Address = 100000h	Idle	8 dwords	0	0
60 ns	Data Transfer 1	Wait State	7 dwords	0	0
90 ns	Data Transfer 2	Write FIFO	7 dwords	0	0
120 ns	Data Transfer 3	Wait State	6 dwords	0	0
150 ns	Data Transfer 4	Write FIFO	6 dwords	0	0
180 ns	Data Transfer 5	Wait State	5 dwords	0	0
210 ns	Data Transfer 6	Write FIFO	5 dwords	0	0
240 ns	Data Transfer 7	Wait State	4 dwords	0	0
270 ns	Data Transfer 8	Write FIFO	4 dwords	0	0
300 ns	Data Transfer 9	Wait State	3 dwords	0	0
330 ns	Data Transfer 10	Write FIFO	3 dwords	0	0
360 ns	Data Transfer 11	Wait State	2 dwords	0	0
390 ns	Data Transfer 12	Write FIFO	2 dwords	0	0
420 ns	Data Transfer 13	Wait State	1 dword	0	0
450 ns	Data Transfer 14	Write FIFO	1 dword	0	0
480 ns	Data Transfer 15	Wait State	0 dwords	1	1
510 ns	Idle (or other Master)	Write FIFO	1 dword	1	1

A

AMCC

BUS MASTERING WITH THE S5933 PCI MATCHMAKER

5.0 DMA SOFTWARE SUPPORT

DMA transfers with the S5933 depend mostly on hardware. Most of the design is the interface on the add-on card which fills and empties the FIFO. There is some software support required for DMA transfers. Address and transfer count registers must be loaded, the FIFOs must be configured, and interrupts (if used) must be enabled and serviced. The following sections provide an overview of what actions are required by software for S5933 DMA operations.

5.1 PCI Initiated DMA Transfers

For PCI initiated DMA transfers, the PCI host CPU (Pentium™, Alpha™, etc.) sets up the S5933 to perform bus master transfers. The following tasks must be completed to setup FIFO bus mastering:

1) **Define interrupt capabilities**. The PCI to add-on and/or add-on to PCI FIFO can generate a PCI interrupt to the host when the transfer count reaches zero.

   INTCSR Bit 15 Enable Interrupt on read transfer count equal zero

   INTCSR Bit 14 Enable Interrupt on write transfer count equal zero

2) **Reset FIFO flags**. This may not be necessary, but if the state of the FIFO flags is not known, they should be initialized.

   MCSR Bit 26 Reset add-on to PCI FIFO flags

   MCSR Bit 25 Reset PCI to add-on FIFO flags

3) **Define FIFO management scheme**. These bits define what FIFO condition must exist for the PCI bus request (REQ#) to be asserted by the S5933.

   MCSR Bit 13 PCI to add-on FIFO management scheme

   MCSR Bit 9 Add-on to PCI FIFO management scheme

4) **Define FIFO priority scheme**. These bits determine which FIFO has priority if both meet the defined condition to request the PCI bus. If these bits are the same, priority alternates, with read accesses occurring first.

   MCSR Bit 12 Read vs. write priority

   MCSR Bit 8 Write vs. read priority

5) **Define transfer source/destination address**. These registers are written with the first address that is to be accessed by the S5933. These address registers are updated after each access to indicate the next address to be accessed. Transfers must start on DWORD boundaries.

   MWAR All Bus master write address

   MRAR All Bus master read address

6) **Define transfer byte counts**. These registers are written with the number of bytes to be transferred. The transfer count does not have to be a multiple of four bytes. These registers are updated after each transfer to reflect the number of bytes remaining to be transferred.

   MWTC All Write transfer byte count

   MRTC All Read transfer byte count

7) **Enable Bus Mastering**. Once steps 1-6 are completed, the FIFO may operate as a PCI bus master. Read and write bus master operations may be independently enabled or disabled.

   MCSR Bit 14 Enable PCI to add-on FIFO bus mastering

   MCSR Bit 10 Enable add-on to PCI FIFO bus mastering

It is recommended that bus mastering be enabled as the last step. Some applications may choose to leave bus mastering enabled and start transfers by writing a non-zero value to the transfer count registers. This also works, provided the entire 26-bit transfer count is written at once. If transfer count interrupts are enabled, they must be enabled after the transfer count(s) are written. If interrupts are enabled and the transfer count is zero, an interrupt occurs immediately.

5.2 Servicing a PCI Initiated DMA Transfer Interrupt

If interrupts are enabled, a host interrupt service routine is also required. The service routine determines the source of the interrupt and resets the interrupt. The source of the interrupt is indicated in the PCI Interrupt Control/Status Register (INTCSR). Typically, the interrupt service routine is used to set up the next transfer by writing a new address and transfer count value, but some applications may also require other actions. If read transfer or write transfer complete interrupts are enabled, master and target abort interrupts are automatically enabled. Writing a one to these bits clears the corresponding interrupt.

INTCSR	Bit 21	Target abort caused interrupt
INTCSR	Bit 20	Master abort caused interrupt
INTCSR	Bit 19	Read transfer complete caused interrupt
INTCSR	Bit 18	Write transfer complete caused interrupt

5.3 Add-on Initiated DMA Transfers

For add-on initiated DMA transfers, the add-on sets up the S5933 to perform bus master transfers. The following tasks must be completed to setup FIFO bus mastering:

1) **Define transfer count abilities.** For add-on initiated bus mastering, transfer counts may be either enabled or disabled. Transfer counts for read and write operations cannot be individually enabled.

AGCSTS	Bit 28	Enable transfer count for read and write bus master transfers

2) **Define interrupt capabilities.** The PCI to add-on and/or add-on to PCI FIFO can generate an interrupt to the add-on when the transfer count reaches zero (if transfer counts are enabled).

AINT	Bit 15	Enable interrupt on read transfer count equal zero
AINT	Bit 14	Enable interrupt on write transfer count equal zero

3) **Reset FIFO flags.** This may not be necessary, but if the state of the FIFO flags is not known, they should be initialized.

AGCSTS	Bit 26	Reset add-on to PCI FIFO flags
AGCSTS	Bit 25	Reset PCI to add-on FIFO flags

4) **Define FIFO management scheme.** These bits define what FIFO condition must exist for the PCI bus request (REQ#) to be asserted by the S5933. This must be programmed through the PCI interface.

MCSR	Bit 13	PCI to add-on FIFO management scheme
MCSR	Bit 9	Add-on to PCI FIFO management scheme

5) **Define FIFO priority scheme.** These bits determine which FIFO has priority if both meet the defined condition to request the PCI bus. If these bits are the same, priority alternates, with read accesses occurring first. This must be programmed through the PCI interface.

MCSR	Bit 12	Read vs. write priority
MCSR	Bit 8	Write vs. read priority

6) **Define transfer source/destination address.** These registers are written with the first address that is to be accessed by the S5933. These address registers are updated after each access to indicate the next address to be accessed. Transfers must start on DWORD boundaries.

MWAR	All	Bus master write address
MRAR	All	Bus master read address

7) **Define transfer byte counts.** These registers are written with the number of bytes to be transferred. The transfer count does not have to be a multiple of four bytes. These registers are updated after each transfer to reflect the number of bytes remaining to be transferred. If transfer counts are disabled, these registers do not need to be programmed.

MWTC	All	Write transfer byte count
MRTC	All	Read transfer byte count

A

8) **Enable Bus Mastering**. Once steps 1-7 are completed, the FIFO may operate as a PCI bus master. Read and write bus master operation may be independently enabled or disabled. The AMREN and AMWEN inputs control bus master enabling for add-on initiated bus mastering. The MCSR bus master enable bits are ignored for add-on initiated bus mastering.

It is recommended that bus mastering be enabled as the last step. Some applications may choose to leave bus mastering enabled (AMREN and AMWEN asserted) and start transfers by writing a non-zero value to the transfer count registers (if they are enabled). This works, provided the entire 26-bit transfer count is written at once. If transfer count interrupts are enabled, they must be enabled after the transfer count(s) are written. If interrupts are enabled and the transfer count is zero, an interrupt occurs immediately.

5.4 Servicing an Add-on Initiated DMA Transfer Interrupt

If interrupts are enabled, an add-on CPU interrupt service routine is also required. The service routine determines the source of the interrupt and resets the interrupt. The source of the interrupt is indicated in the Add-on Interrupt Control Register (AINT). Typically, the interrupt service routine is used to set up the next transfer by writing a new address and transfer count value (if enabled), but some applications may also require other actions. If read transfer or write transfer complete interrupts are enabled, the master/target abort interrupt is automatically enabled. Writing a one to these clears the corresponding interrupt.

AINT	Bit 21	Master/target abort caused interrupt
AINT	Bit 19	Read transfer complete caused interrupt
AINT	Bit 18	Write transfer complete caused interrupt

6.0 SAMPLE S5933 DMA SUPPORT CODE

The following section is a sample program written in C-language to setup the S5933 for DMA operations. The code is written for PCI initiated bus mastering using transfer count interrupts. The code is written for an x86 compatible PCI platform, but could be modified to support other host processors.

The code has been compiled using Borland C/C++ Version 4.5. Because 32-bit registers are used within the code, code must be compiled to generate 386 code (otherwise, 32-bit register mneumonics such as _EAX are not recognized).

```c
#include <dos.h>
#include <stddef.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include "amcc.h"


/*****************************************************************/
/*  A2P = Add-on to PCI FIFO                                     */
/*  P2A = PCI to Add-on FIFO                                     */
/*****************************************************************/

/* Transfer Count Interrupt Enables */
#define EN_READ_TC_INT        0x00008000L
#define EN_WRITE_TC_INT       0x00004000L

/* FIFO Flag Reset */
#define RESET_A2P_FLAGS       0x04000000L
#define RESET_P2A_FLAGS       0x02000000L

/* FIFO Management Scheme */
#define A2P_REQ_AT_4FULL      0x00000200L
#define P2A_REQ_AT_4EMPTY     0x00002000L

/* FIFO Relative Priority */
#define A2P_HI_PRIORITY       0x00000100L
#define P2A_HI_PRIORITY       0x00001000L

/* Enable Transfer Count */
#define EN_TCOUNT             0x10000000L

/* Enable Bus Mastering */
#define EN_A2P_TRANSFERS      0x00000400L
#define EN_P2A_TRANSFERS      0x00004000L

/* Identify S5933 Interrupt Sources */
#define ANY_S5933_INT         0x80
#define READ_TC_INT           0x08
#define WRITE_TC_INT          0x04
#define MASTER_ABORT_INT      0x10
#define TARGET_ABORT_INT      0x20
#define BUS_MASTER_INT        0x20

/* Global Variable Definition */
byte    interrupt_line;
word    op_reg_base_address;
void(interrupt *oldhandler)(void);
```

A

```
/******************************************************************/
/*      MAIN Code Segment                                         */
/******************************************************************/


void main()
{
        void    setup_pci_dma(void);
        void    setup_int_vect(byte bus_num,byte dev_func);
        word    vendor_id, device_id, index;

        byte    bus_num, dev_func;
        int     bios_present;


/* AMCC Default Vendor/Device ID's */
        vendor_id = 0x10E8;
        device_id = 0x4750;
        index = 0;

/* Disable Host Interrupts */
        disable();

/* Look for a valid PCI BIOS */
        if(pci_bios_present(NULL,NULL,NULL)==SUCCESSFUL){
                bios_present=TRUE;
                printf("PCI BIOS Found\n\n");
        }
        else{
                printf("PCI BIOS not present\n\n");
                bios_present=FALSE;
        }

/* If the BIOS is present, look for the AMCC device */
        if(bios_present){
                if(find_pci_device(device_id,vendor_id,index,&bus_num,
                        &dev_func)==SUCCESSFUL){
                        printf("AMCC Device Found: Bus=%d Device=%d Function=%d\n\n",
                                bus_num,(dev_func>>3),(dev_func&0x7));
                }
                else{
                        printf("AMCC Device Not Found\n\n");
                }
        }

/* Find the physical location of the S5933 in I/O space */
        read_configuration_word(bus_num,dev_func,
                PCI_CS_BASE_ADDRESS_0,&op_reg_base_address);

/* Mask Lower 2 bits of BADR0 */
        op_reg_base_address = (op_reg_base_address & 0xFFFC);

/* Call routine to install interrupt vector */
        setup_int_vect(bus_num,dev_func);

/*Enable hardware interrupts */
        enable();

/* Call routine to setup PCI initiated bus mastering */
        setup_pci_dma();
}
```

```
/*************************************************************************/
/*      Function:       setup_pci_dma                                    */
/*      Purpose:        Configure the S5933 for PCI initiated DMA        */
/*      Inputs:         None                                             */
/*      Outputs:        None                                             */
/*                                                                       */
/* The following function assumes the S5933 is configured               */
/* for PCI initiated DMA transfers and sets up the DMA channels         */
/*************************************************************************/

void setup_pci_dma(void)
{
        char src[20], dest[20], rtc[20], wtc[20];
        dword  source, destination, temp = 0;
        dword  readtc, writetc;

        /* Read in source,destination and transfer counts */
        printf("Input address values in hex format: 0x...\n\n");
        printf("Input transfer count values in decimal\n\n");
        printf("Read from address: ");
        gets(src);
        source=strtoul(src,NULL,16);
        printf("%lX\n",source);

        printf("Write to address: ");
        gets(dest);
        destination=strtoul(dest,NULL,16);
        printf("%lX\n",destination);

        printf("Read byte count: ");
        gets(rtc);
        readtc=strtoul(rtc,NULL,10);
        printf("%d\n",readtc);

        printf("Write byte count: ");
        gets(wtc);
        writetc=strtoul(wtc,NULL,10);
        printf("%d\n",writetc);

        outpd(op_reg_base_address+AMCC_OP_REG_MWAR, destination);
        outpd(op_reg_base_address+AMCC_OP_REG_MRAR, source);
        outpd(op_reg_base_address+AMCC_OP_REG_MWTC, writetc);
        outpd(op_reg_base_address+AMCC_OP_REG_MRTC, readtc);

        /* Enable Transfer Count interrupts */
        temp = inpd(op_reg_base_address+AMCC_OP_REG_INTCSR);
        outpd(op_reg_base_address+AMCC_OP_REG_INTCSR,temp|
                EN_READ_TC_INT|
                EN_WRITE_TC_INT);

        /* Enable Bus Mastering */
        temp = inpd(op_reg_base_address+AMCC_OP_REG_MCSR);
        outpd(op_reg_base_address+AMCC_OP_REG_MCSR, temp|RESET_A2P_FLAGS|
                RESET_P2A_FLAGS|
                A2P_HI_PRIORITY|
                P2A_HI_PRIORITY|
                EN_A2P_TRANSFERS|
                EN_P2A_TRANSFERS);
}
```

A

```
/***********************************************************************/
/*      Function:       handler                                      */
/*      Purpose:        Check interrupt source and service S5933 int's */
/*      Inputs:         None                                         */
/*      Outputs:        None                                         */
/***********************************************************************/

void interrupt handler(void)
{
        byte    status;

        status = inportb(op_reg_base_address+AMCC_OP_REG_INTCSR+2);

        if((status & ANY_S5933_INT)!= 0){
                /* Disable bus mastering */
                outportb(op_reg_base_address+AMCC_OP_REG_MCSR+1,0x11);

                /* Identify AMCC Interrupt Source(s) */
                /* AMCC Hardware Interrupt Source */
                if((status & READ_TC_INT) != 0)
                        /* Read TC Interrupt Code Here */
                if((status & WRITE_TC_INT) != 0)
                        /* Write TC Interrupt Code Here */
                if((status & MASTER_ABORT_INT) != 0)
                        /* Master Abort Interrupt Code Here */
                if((status & TARGET_ABORT_INT) != 0)
                        /* Target Abort Interrupt Code Here */

                /* Clear Interrupt Enables */
                outportb(op_reg_base_address+AMCC_OP_REG_INTCSR+1,0);

                /* Clear all active interrupt sources */
                outportb(op_reg_base_address+AMCC_OP_REG_INTCSR+2,status);

                /* Reenable transfer count interrupts */
                outportb(op_reg_base_address+AMCC_OP_REG_INTCSR+1,0xC0);
                }
        else{
                /* Not an S5933 Interrupt Source */
                _chain_intr(oldhandler);
                }

        /* Specific End of interrupt to clear in-service bit(s)    */
        /* Mask upper 5 bits of int. line register                */
        if(interrupt_line<8)
                outportb(0x20,0x60|(interrupt_line&0x07));
        else{
                /* Issue master then slave EOI */
                outportb(0xa0,0x60|((interrupt_line-8)&0x07));
                outportb(0x20,0x62);
                }
        }
}
```

```
/***********************************************************************/
/* SETUP_INT_VECT      */
/*      */
/* Purpose:    Install the interrupt vector for the S5933 interrupt      */
/*      handler and reference the previous handler routine.      */
/* Inputs:     S5933 Operation registers base address      */
/* Outputs:    None                                                      */
/***********************************************************************/


void setup_int_vect(byte bus_num,byte dev_func)

{
        byte    int_vector;
        int mask;

        if(read_configuration_byte(bus_num,dev_func,PCI_CS_INTERRUPT_LINE,
                &interrupt_line)==SUCCESSFUL){

                if(interrupt_line != 0xff){
                        if(interrupt_line<8){
                                int_vector=0x08+interrupt_line;
                        }
                        else{
                                int_vector=0x70+(interrupt_line-8);
                        }
                }

                /* Make sure the system 8259 enables the IRQ with OCW1 @ I/O 21h    */
                /* for IRQ0-7 or @ I/O A1h for IRQ8-15                              */
                if(interrupt_line < 8){
                        mask = inportb(0x21);
                        mask = mask & ~(1<<interrupt_line);
                        outportb(0x21,mask);
                        }
                else {
                        mask = inportb(0xA1);
                        mask = mask & ~(1<<(interrupt_line-8));
                        outportb(0xA1,mask);
                        }
        /* If AMCC interrupt service routine vector is not already installed,     */
        /* install it and save the old vector for chaining.                       */
        if(getvect(int_vector) != handler){
                /* Save the old interrupt vector */
                oldhandler=getvect(int_vector);

                /* Install the new interrupt vector */
                setvect(int_vector,handler);
                }
        }
}
```

A

1.0 INTRODUCTION

The S5933 allows PCI bus master transfers through the FIFO interface. The function of filling and emptying the FIFO is left to add-on logic. Many add-on designs implement a microprocessor or microcontroller with an integrated DMA controller that can perform this function. These devices can easily transfer data between the S5933 FIFO port and add-on memory.

Some add-on designs do not have processors or logic with DMA capabilities. This application note shows a programmable logic implementation of a simple, single channel DMA controller to perform add-on DMA transfers between the S5933 FIFO port and add-on memory. The design described allows simple implementation into an Intel 80960 processor-based add-on, but can easily be modified to support other add-on processors.

2.0 DMA CONTROLLER ARCHITECTURE

This DMA controller design has a single channel which can perform add-on reads and writes as a bus master on the add-on interface. An add-on transfer address and transfer byte count are programmed, and then DMA requests from the S5933 FIFO are monitored. Once a DMA request is received, the controller puts add-on logic in a hold state and when a hold acknowledge is returned, the DMA transfer begins.

2.1 Register Architecture

There are two registers integrated in the DMA controller: a 22-bit address register and an 18-bit transfer count register. These registers are initialized by the processor before the DMA transfer begins. These registers are write-only. The DMA controller decodes address lines A21 and A20 on the add-on bus. This address decoding scheme can be easily modified to fit into the memory map of specific applications.

2.1.1 Address Register (ADDR)

The address register contains the address of the next DMA transfer. Before the DMA transfer begins, the ADDR register is written with the starting address of the DMA transfer. All transfers must take place on double-word boundaries (A1=A0=0). The ADDR register is incremented by 4 bytes after each data phase completes.

Register Name:	DMA Transfer Address (ADDR)
Location:	A21=0, A20=1
Power-up Value:	00000000h
Attribute:	Write Only
Size:	32-bits

31	21 20		2 1 0
0 0 0 0 0 0 0 0 0 0 0		Transfer Address	0 0
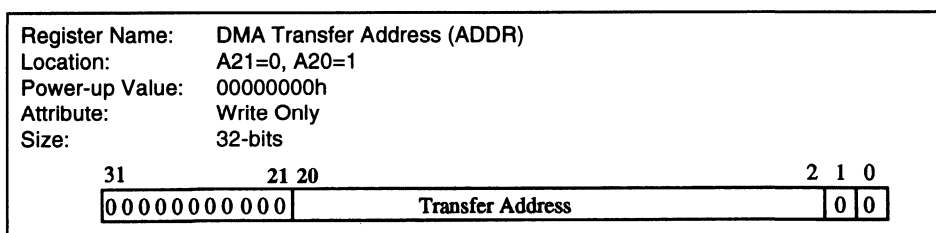
A

2.1.2 Transfer Count Register (TXCNT)

The transfer count register contains the current number of bytes left to be transferred. Before the DMA transfer begins, the TXCNT register is written with the total number of bytes to be transferred. All transfers counts must be multiples of 4 bytes. The TXCNT register is decremented by 4 bytes after each data phase completes. Bit 24 of the TXCNT register controls the direction of the DMA transfer (read from the S5933 or write to the S5933).

```
Register Name:    DMA Byte Transfer Count (TXCNT)
Location:         A21=1, A20=1
Power-up Value:   00000000h
Attribute:        Write Only
Size:             32-bits

       31        24 23      18 17                             2 1 0
      0 0 0 0 0 0 0  0 0 0 0 0 0       Transfer Byte Count       0 0

                                               RWCONT
                                               0 = S593X Write (Add-on Read)
                                               1 = S593X Read (Add-on Write)
```
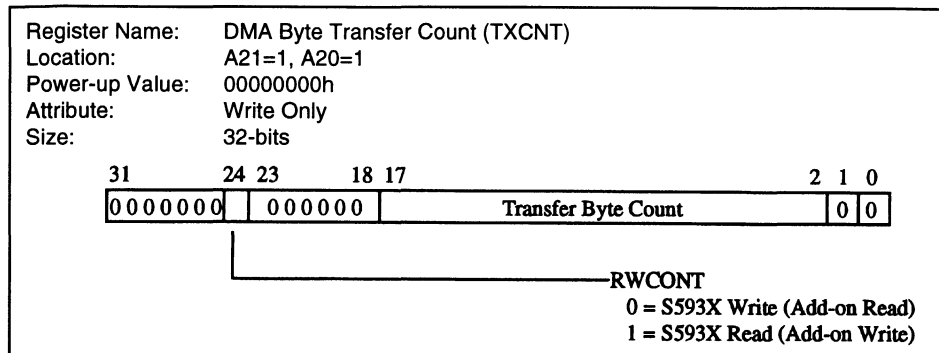
2.2 DMA Controller Bus Interface

The DMA controller bus cycles are identical to bus cycles performed by an Intel 80960Jx processor. This allows the DMA controller to transfer data to and from any peripheral or memory controller compatible with the i960Jx processor. The PLD state machines can easily be modified to emulate other processor bus cycles.

The following signals make up the bus interface for the DMA controller:

Signal	Type	Function
A31:0	Bidirectional	Address/data lines for accessing DMA registers.
RDY_IN#	Input	Ready Input for DMA controller-generated bus cycles.
RDY_OUT#	Output	Ready Output for processor-generated bus cycles.
BLAST#	3-State	Indicates that the current data phase is the last data phase of a burst.
ADS#	Bidirectional	Indicates that valid address information is currently on the bus.
W/R#	Bidirectional	Write/Read indicator.
BE3:0#	3-State	Byte Enables. Always asserted for DMA transfers.
RDFIFO#	Output	Read one double-word from the S5933 FIFO.
WRFIFO#	Output	Write one double-word to the S5933 FIFO.
HOLD	Output	Add-on processor hold request.
HLDA	Input	Add-on processor hold acknowledge.
RESET#	Input	Add-on reset signal (from the S5933)

All bidirectional signals are driven by the current add-on bus master. When the add-on processor controls the bus, the DMA controller floats these outputs.

2.3   DMA Controller Slave Access State Machine

The DMA controller registers are mapped into add-on memory space. The address decode is a 2-bit decode of address bits A21 and A20. For add-on designs with conflicting memory requirements, the PLD equations can be easily modified to decode more address bits or different combinations of address bits.

When the controller decodes its address with ADS# asserted, a register access begins. HLDA (hold acknowledge) must be deasserted to allow a register access, indicating that the DMA controller is not the add-on bus master. The SLAVE state machine controls all accesses by the add-on processor or logic. Figure 1 shows the SLAVE state machine state diagram.

Figure 1. DMA Controller Slave State Machine
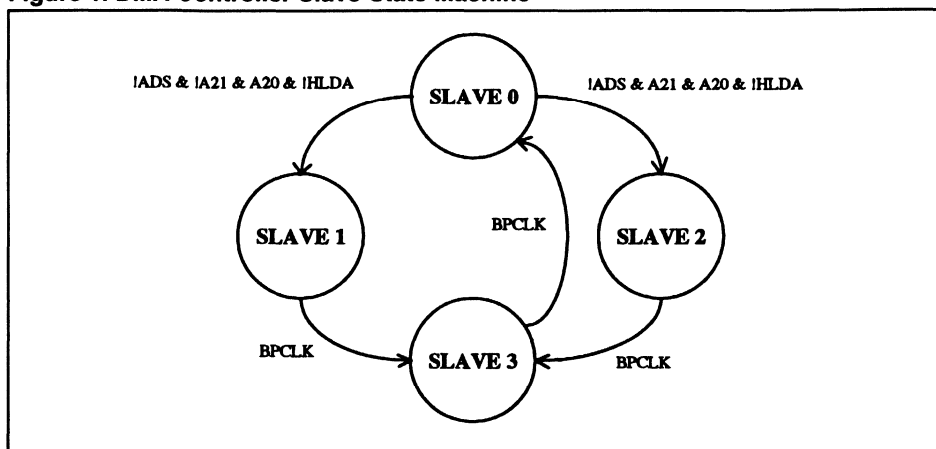
SLAVE 0 is the idle state. Depending on the state of A21 and A20 during the address phase of an access to the DMA controller, the SLAVE 1 or SLAVE 2 state is entered. SLAVE 1 is a write to the address register, SLAVE 2 is a write to the transfer count register. It only takes a single clock to write the register, so in SLAVE 1 and SLAVE 2, RDY_OUT# is asserted, and the next rising edge of BPCLK advances the state machine to SLAVE 3. SLAVE 3 is a recovery state (required for all 80960 bus cycle accesses). RDY_OUT# is deasserted, completing the access. The state machine returns to SLAVE 0 at the next rising edge of BPCLK.

For logic which does not require recovery states, SLAVE 3 may be removed, and the state machine can advance from SLAVE 1 or SLAVE 2 back to SLAVE 0 (the idle state).

A

Figure 2 shows a register write cycle by a 960Jx processor to the DMA controller. Ta indicates the processor address phase, Td is the data phase and Tr is the recovery phase. RDY_RCV# is an input into the 960 processor and is driven by the DMA controller RDY_OUT# signal. BLAST# is also driven by the processor and indicates the current data phase is the last data phase of a burst. For a single-cycle access, BLAST# is asserted during the first and only data phase.

Figure 2. Processor Write to DMA Address Register
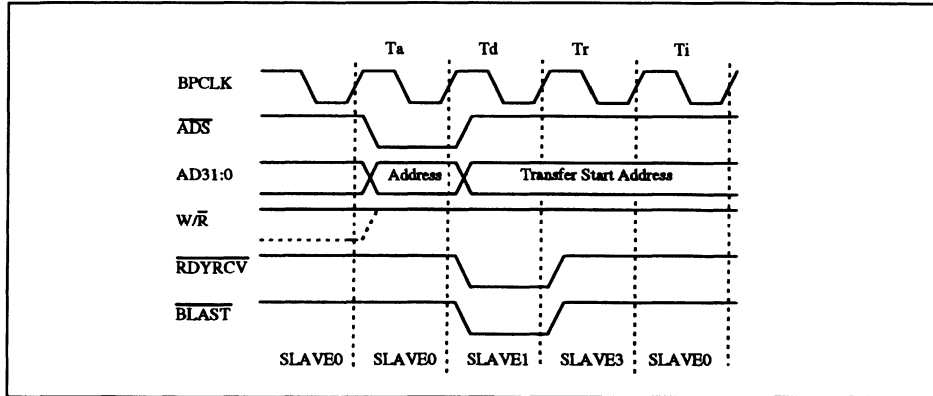
2.4   DMA Controller Bus Master State Machine

The MASTER state machine controls transfers across the add-on bus when DMA controller is the add-on bus master. In the idle state (MASTER 0), the controller monitors the S5933 FIFO status outputs. If the DMA controller is programmed to read from the PCI to add-on FIFO (RWCONT=1), RDEMPTY is monitored. If the DMA controller is programmed to write to the add-on to PCI FIFO (RWCONT=0), WRFULL is monitored. RDEMPTY and WRFULL act as the DMA requests (and they will be referred to as 'the DMA request' in the description of the MASTER state machine). Neither request will be acted upon by the DMA controller if the transfer count register is zero. Regardless of whether the transfer is to be a read or a write, the state machine operates the same way, only the W/R# output is different. The MASTER state machine state diagram is shown in Figure 3. For simplicity, only the conditions required to change states are shown. All other conditions result in the state machine remaining in the present state.

Once a valid DMA request is received (with a non-zero transfer count), the state machine advances to MASTER 1. In MASTER 1, HOLD is asserted to the add-on processor. The state machine does not advance to MASTER 2 until HLDA is returned by the processor, indicating that the DMA controller has access to the add-on bus. HLDA asserted enables the outputs for all of the bidirectional DMA controller signals (ADS#, BLAST#, W/R#, BE3:0#).

MASTER 2 is the address phase for an add-on bus cycle. In MASTER 2, ADS# is asserted and the outputs for the AD31:0 signals are enabled, driving the DMA transfer address on the add-on bus. The next rising edge of BPCLK advances the state machine to one of four data phases: MASTER 3-6. Which state occurs depends on the remaining transfer count value. Because BLAST# (last data phase of a burst indicator) is always asserted in MASTER 6, the state machine always operates so that the final operation of a DMA transfer ends in MASTER 6.

MASTER 3 is the first data phase of a four data phase burst. If the transfer count is 16 bytes or more in the MASTER 2 state, the state machine advances to MASTER 3 (because at least 4 more data phases are required). If the state machine enters MASTER 3, there is always data to transfer, so only RDY_IN# is monitored. The state machine remains in MASTER 3 until RDY_IN# is asserted by the add-on device being accessed. When RDY_IN# is asserted, the state machine advances to MASTER 4, and the ADDR and TXCNT registers are updated.

Figure 3. DMA Controller Add-on Bus Master State Machine
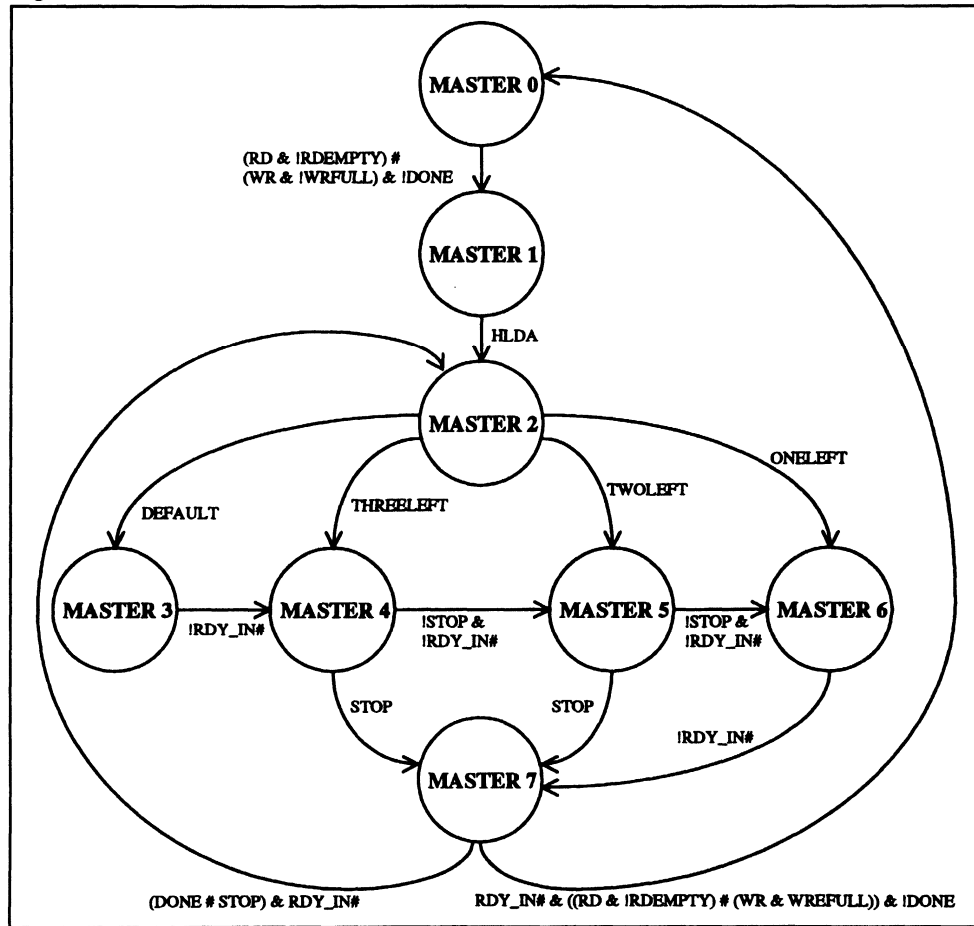
The MASTER 4 data phase can be entered from either the address phase (if the transfer count has decremented to 12) or from MASTER 3. During the MASTER 4 data phase, RDY_IN# and STOP are monitored. STOP is an internal signal which is asserted when a DMA request goes inactive. For example, if the last double-word in the PCI to add-on FIFO is read in MASTER 3, the RDEMPTY output is asserted. This causes STOP to be asserted in the MASTER 4 state. STOP asserted also prevents RDFIFO# and WRFIFO# from being asserted and does not allow data to be transferred. When STOP is asserted, the MASTER 4 state is completes when RDY_IN# is asserted, but the address and transfer count are not updated, and the state machine asserts BLAST# , advancing to MASTER 7 (recovery phase). If STOP is not asserted in MASTER 4, the data phase completes normally when RDY_IN# is asserted and the state machine advances to MASTER 5, updating the ADDR and TXCNT registers.

The MASTER 5 data phase is identical to MASTER 4. Master 5 can be entered from either the address phase (if the transfer count has decremented to 8) or from MASTER 4. If STOP is asserted, the state machine asserts BLAST# and advances to MASTER 7 when RDY_IN# is asserted. If STOP is not asserted, the state machine advances to MASTER 6 when RDY_IN# is asserted, updating the ADDR and TXCNT registers.

The MASTER 6 data phase can be entered from either the address phase (if the transfer count has decremented to 4) or from MASTER 5. Only RDY_IN# is monitored in MASTER 6. Because MASTER 7 is the next state anyway, there is no need to monitor STOP. BLAST# is always asserted during the MASTER 6 state. When RDY_IN# is asserted, the state machine advances to MASTER 7, and the ADDR and TXCNT registers are updated.

MASTER 7 is the recovery phase. In MASTER 7, RDY_IN#, RDEMPTY, WRFULL, STOP, and DONE are monitored. DONE is an internal signal which indicates that the transfer count is zero. Based on these signals, the state machine either returns to the idle state (MASTER 0) or starts another transfer by advancing to the address phase (MASTER 2). If RDY_IN# is low, the state machine remains in MASTER 7. If RDY_IN# is high and the DMA request is still active (RDEMPTY or WRFULL deasserted), the state machine advances to MASTER 2 to begin another data transfer. If RDY_IN# is high and DONE or STOP is asserted (either the transfer count is zero, or the FIFO cannot support another transfer yet), then the state machine returns to MASTER 0 and relinquishes control of the add-on bus by deasserting HOLD.

2.5   DMA Controller Bus Operation

The DMA controller emulates bus cycles of an Intel 960Jx processor. This allows the controller to transfer data directly to and from memory controllers or peripherals designed for the 960. The S5933 FIFO allows an unlimited burst length, but the 960 supports a maximum of 4 data phases burst accesses. Therefore, the DMA controller bursts no more than four consecutive data phases before issuing an address phases.

2.5.1   FIFO DMA Request for Writes to an Add-on Destination

For the S5933 PCI to add-on FIFO, RDEMPTY is deasserted when there is valid data in the FIFO. The FIFO can be filled by another PCI bus initiator using the S5933 as a target, or the S5933 can fill its own FIFO by acting as a PCI bus master. The DMA controller functions identically in either case. RDEMPTY deasserted acts as a DMA request to the controller when RWCONT =1.

2.5.2   FIFO DMA Request for Reads from an Add-on Source

For the S5933 add-on to PCI FIFO, WRFULL is deasserted when the add-on to PCI FIFO is completely full. The FIFO can be emptied by another PCI bus initiator using the S5933 as a target, or the S5933 can empty its own FIFO by acting as a PCI bus master. The DMA controller functions identically in either case. WRFULL deasserted acts as a DMA request to the controller when RWCONT=0.

2.5.3   DMA Requests and Add-on Bus Arbitration

When the DMA controller has been programmed with a source address (for DMA reads) or destination address (for DMA writes) and a transfer byte count, it is ready to receive DMA requests from the S5933. For DMA reads (RWCONT=1), RDEMPTY acts as the request. Whenever RDEMPTY is low, there is data in the PCI to add-on FIFO that needs to be transferred to an add-on destination. For DMA writes (RWCONT=0), WRFULL acts as the request. Whenever WRFULL is low there are empty locations in the add-on to PCI FIFO that need to be filled from an add-on source. RDEMPTY or WRFULL low starts the MASTER state machine.

Figure 4 shows the sequence for the DMA controller becoming the add-on bus master. The DMA controller becomes an add-on bus master using the 960 processor's hold/hold acknowledge protocol. When a DMA request is received, HOLD is asserted. When the processor returns HLDA, the DMA controller can begin transferring data to or from the S5933 FIFO. The DMA controller remains bus master until the request is removed (RDEMPTY or WRFULL asserted).

Figure 4. DMA Request and Add-on Bus Arbitration
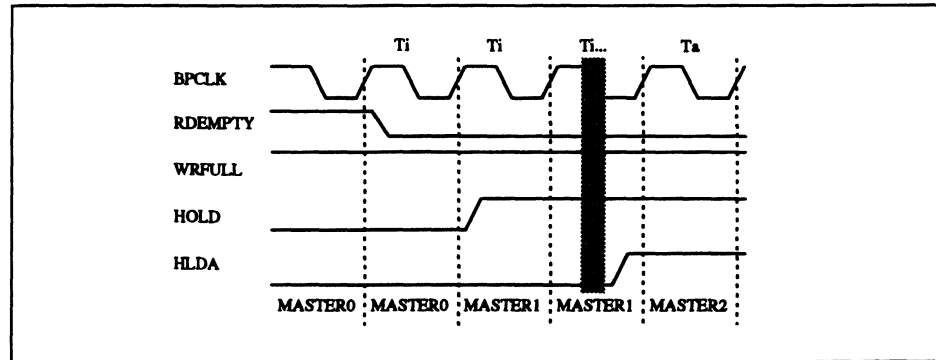
2.5.4 DMA Burst Reads

The DMA controller always attempts to read from the S5933 PCI to add-on FIFO in bursts. Figure 5 shows a two data phase burst of the last two double words in a transfer (TXCNT = 8). Typically, a write to DRAM has more wait states on the first data phase, but this is not shown, for simplicity. RDY_IN# is driven by an external device such as a memory controller and may be used to extend individual data phases for as long as necessary.

The DMA controller is designed to interface with the FIFO asynchronous to BPCLK. This means that bit 6 of location 45h of the non-volatile boot device for the S5933 should be set. In this mode of operation, the S5933 advances the FIFO pointer at each rising edge of RDFIFO#. To interface to a zero wait-state device, the S5933 would have to be configured for FIFO operation synchronous to BPCLK (where the FIFO advances at every BPCLK rising edge when RDFIFO# is asserted). This would also require modification of the PLD equations.

Figure 5. DMA Burst Read from the S5933 FIFO (Add-on Write)
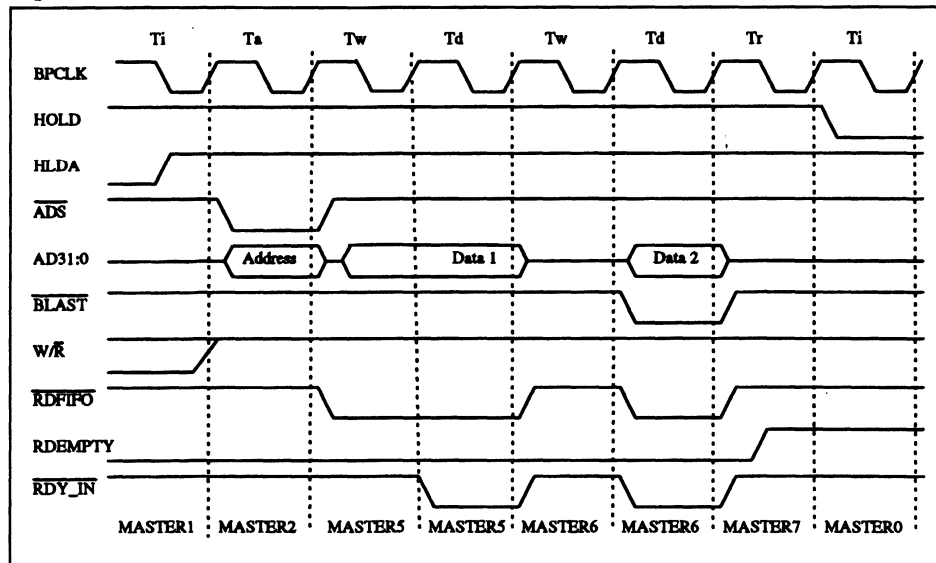
2.5.5 DMA Burst Writes

The DMA controller always attempts to write to the S5933 add-on to PCI FIFO in bursts. Figure 6 shows a two data phase burst of the last two double words in a transfer (TXCNT = 8). RDY_IN# is driven by an external device such as a memory controller and may be used to extend individual data phases for as long as necessary.

The DMA controller is designed to interface with the FIFO asynchronous to BPCLK. This means that bit 5 of location 45h of the non-volatile boot device for the S5933 should be set. In this mode of operation, the S5933 advances the FIFO pointer at each rising edge of WRFIFO#. To interface to a zero wait-state device, the S5933 would have to be configured for FIFO operation synchronous to BPCLK (where the FIFO advances at every BPCLK rising edge when WRFIFO# is asserted). This would also require modification of the PLD equations.

Figure 6. DMA Burst Write to the S5933 FIFO (Add-on Read)
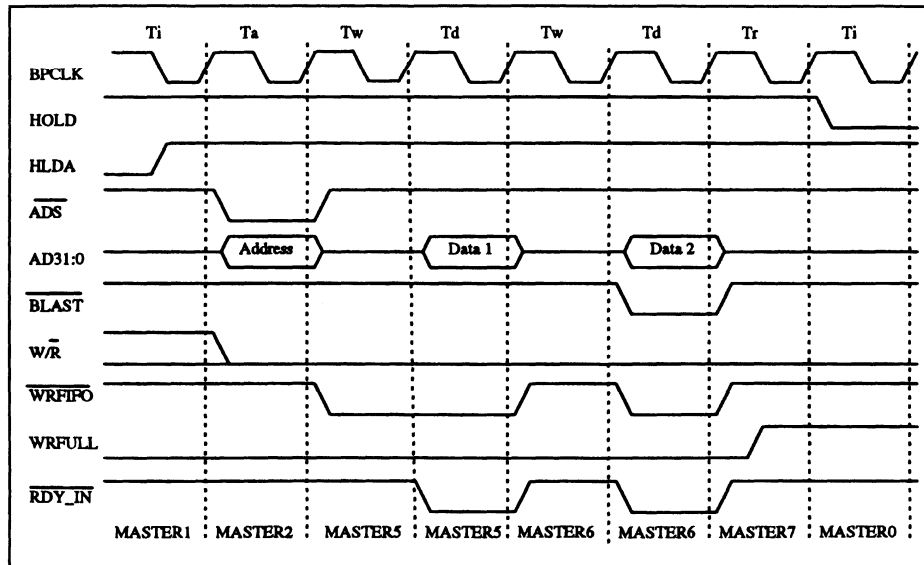
2.5.6 Removal of a DMA Request During a Burst

It is possible that during a burst read or write operation by the DMA controller, the DMA request will be removed. This indicates that the S5933 FIFO cannot support any more data transfers. For DMA reads from the S5933 PCI to add-on FIFO, this occurs when RDEMPTY is asserted. This indicates that there is no data in the FIFO to be read. For DMA writes to the S5933 add-on to PCI FIFO, this occurs when WRFULL is asserted. This indicates that the FIFO is full and no more data should be written.

The DMA controller state machine provides for this situation. Figure 7 shows a DMA burst read from the S5933 FIFO. Data 1 is transferred to Address 1, Data 2 is transferred to Address 1 + 4. RDEMPTY is asserted after the second double-word is transferred, indicating there is currently no more data to be read. The state machine has already advanced to the next data phase, but when RDEMPTY is asserted, RDFIFO# is not asserted during the final data phase, and the address and transfer count registers are not modified. This prevents the FIFO pointers from being updated, but a dummy write cycle is still run to memory at Address 1 + 8. When RDEMPTY is deasserted again (indicating there is more data available), the DMA transfer continues from the next address (Address 1 +8). This results in the data written by the dummy cycle to be overwritten with the next valid data.

When a DMA request is removed during a data phase, the DMA controller completes the data phase (without asserting RDFIFO# or WRFIFO# or updating the address/count registers) and advances to the recovery phase. From the recovery phase, if the DMA request has not been reasserted, the state machine advances to the idle state, giving up control of the add-on bus. If the DMA request has been reasserted by the time the recovery phase is completed, then the state machine advances to the address phase and begins another data transfer.

Figure 7. DMA Request (RDEMPTY) Removal During a Burst Read Transfer
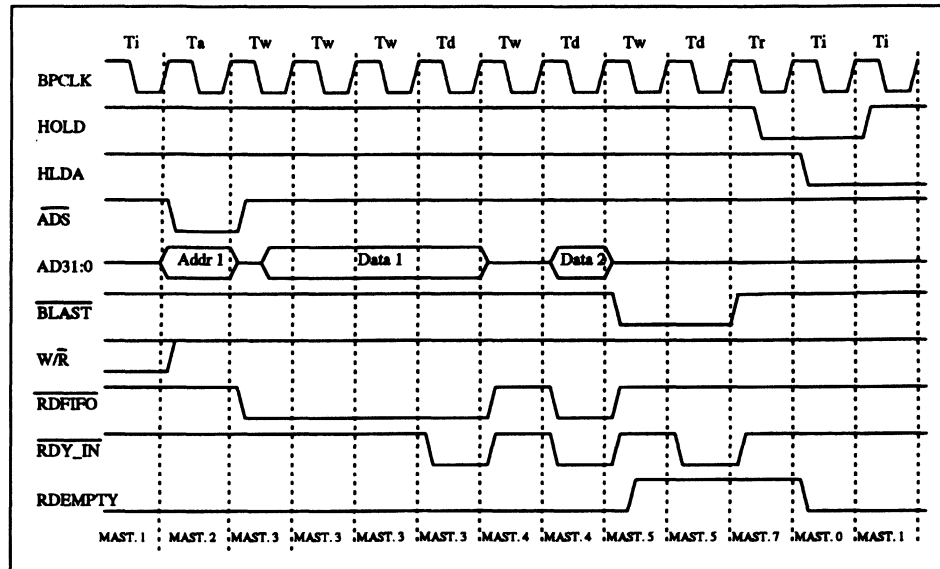
3.0 PLD MODIFICATIONS

The PLD implementation described in this application note is a general-purpose, single-channel DMA controller. The design can easily be modified for a specific add-on application. Modifications such as decreasing the address register or transfer count size are relatively simple. The design could also be easily modified for a 16-bit add-on interface. The following sections describe some possible modifications and show how they can be implemented.

3.1 Automatically Programming Transfer Count and Address Registers

Many S5933 applications may implement add-on initiated bus mastering. This allows add-on logic to program S5933 PCI bus master address registers (MWAR and MRAR) and transfer count registers (MWTC and MRTC). In this situation, it might be desirable to allow the DMA controller address and transfer count registers to be programmed during the write access to the corresponding S5933 add-on operation registers. Whenever a PCI bus master read or write address is programmed, the DMA controller address register is programmed during the same cycle. Whenever a PCI bus master read or write transfer count is programmed, the DMA controller transfer count is programmed during the same cycle. This avoids running extra add-on bus cycles to setup add-on DMA transfers and S5933 bus mastering.

A

To do this, the PLD must decode the addresses for the S5933 bus master address and transfer count registers. The following S5933 add-on operation register locations must be decoded:

Register	Offset
Bus Master Read Address Register (MRAR)	30h
Bus Master Write Address Register (MWAR)	24h
Bus Master Read Transfer Count (MRTC)	5Ch
Bus Master Write Transfer Count (MWTC)	58h

Add-on bus address lines A6:2 must be decoded. An access to either MRAR or MWAR programs the DMA address register. It should be noted that MWAR and MRAR are 32-bit registers (as PCI memory space is 4 GB), only the lower 22 bits are stored by the DMA controller address register (which should be sufficient for most applications).

An access to either MRTC or MWTC programs the DMA transfer count register. Bit 24 of the DMA controller is the RWCONT bit (indicating a read or write from the S5933). This bit should be programmed based on which transfer count register of the S5933 is programmed (indicated by A2).

The following PLD equation modifications are required to implement this function:

```
/* Inputs */
Pin   =   nSELECT;    /* S5933 SELECT# Input */

/* Logic Equations */

/* For MWTC and MRTC accesses, A2 differentiates between PCI read and write */
/* transfer count registers */

RWCONT.T = LOADC & (RWCONT $ A2.IO);

SEQUENCE slave {
/* Idle State */
present S0
     if !nADS.IO & !nSELECT & !HLDA &
          ((A5.IO & A4.IO & !A3.IO & !A2.IO) #
           (A5.IO & !A4.IO & !A3.IO & A2.IO)) next S1;
     if !nADS.IO & !nSELECT & !HLDA & A5.IO   next S2;
     default next S0;

present S1        /* Address Access Data Phase */
     next S3;

present S2        /* Transfer Count Access Data Phase */
     next S3;

present S3        /* Recovery Phase */
     next S0;
}
```

3.2 Controlling S5933 PCI Bus Mastering with AMREN and AMWEN

For S5933 applications implementing add-on initiated bus mastering, the AMREN and AMWEN inputs enable the S5933 to become a PCI bus master. These signals can be sourced from the DMA controller and asserted when the controller is ready to service DMA requests. If add-on transfer counts are enabled for the S5933, this function is not required, as the AMREN and AMWEN may remain asserted and PCI bus mastering is internally enabled by a non-zero transfer count value.

AMREN should be asserted for DMA S5933 reads (RWCONT=1) and AMWEN should be asserted for DMA S5933 writes (RWCONT=0). Once the DMA controller is initialized (address and transfer count written), the appropriate enable should be asserted. When the transfer count reaches zero, the enable must be deasserted, and the FIFO pointers should be reset (using FRC# for the PCI to add-on FIFO and FWC# for the add-on to PCI FIFO). It is only really necessary to reset the PCI to add-on FIFO because extra data may have been read which is not required. This must be flushed from the FIFO. Extra data does not enter the add-on to PCI FIFO without WRFIFO# being asserted, the DMA controller logic prevents this.

The following PLD equation modifications are required to implement this function:

```
/* Inputs */
Pin    =    FWE;         /* Add-on to PCI FIFO Empty indicator */

/* Outputs */

Pin    =    nFRC;        /* PCI to add-on FIFO reset */
Pin    =    AMREN;          /* PCI to add-on FIFO PCI master enable */
Pin    =    AMWEN;          /* Add-on to PCI FIFO PCI master enable */

/* Logic Equations */

AMREN = !DONE & RD;             /* Assert for reads with tc != zero */
AMWEN = (!DONE # !FWE) & WR;    /* Assert for writes with tc != zero and */
                        /* FIFO not empty. The transfer count can */
                        /* decrement to zero, but bus master writes */
                        /* still remain, FWE indicates when the FIFO */
                        /* has finished all transfers */

nFRC = !(DONE & MASTER7 & RD); /* Assert the PCI to add-on FIFO reset */
                        /* during the recovery phase when the */
                        /* transfer count reaches zero */
```

3.3 Programming the DMA Controller with the Pass-thru Interface

Some applications may not implement a processor or intelligent logic on the add-on interface. This requires an alternate method to program the DMA controller address and transfer count registers. The S5933 pass-thru interface provides a simple method to perform this task with the host CPU. A Base Address Register is required to define a pass-thru region for the DMA registers. An 8 byte I/O region is all that is needed (a decode of the S5933 PTNUM1:0 outputs and pass-thru address line A2 are used for the add-on address decode).

Not implementing an add-on processor changes a number of things in the PLD. HOLD and HLDA are no longer required, as the DMA controller is always add-on bus master in this implementation. RDY_OUT# is no longer required (as it acted as a ready indicator to the 960 processor). RDY_OUT# is replaces with PTRDY#, which indicates a pass-thru access to a DMA controller register is now complete. PTATN# acts as a request for the DMA controller to read the pass-thru data register (writing the address or transfer count register).

A

The DMA controller MASTER state machine is modified slightly. The MASTER 1 state changes to prevent the controller from starting an add-on DMA transfer while a pass-thru access to one of its registers is occurring. The SLAVE machine requires more modification. SLAVE 0 only monitors PTATN#. If PTATN# is asserted and a DMA transfer is not in progress, the state machine advances to SLAVE 1. During SLAVE 1, PTADR# is asserted and the pass-thru address is decoded, determining if the DMA address or transfer count is to be written (or if the pass-thru access is not intended for the DMA controller). SLAVE 2 and SLAVE 3 program the appropriate DMA registers, asserting RD# and PTRDY#. The state machine then advances to SLAVE 0, returning the machine to its idle state.

The following PLD equation modifications are required to implement this function:

```
/* Inputs */
Pin    =    nPTATN;           /* Pass-thru Access Indicator */
Pin    =    PTNUM0;           /* Pass-thru Region Decode */
Pin    =    PTNUM1;
                             /* Remove HLDA Output */


/* Outputs */

Pin    =    nPTADR;           /* Pass-thru Address Strobe */
Pin    =    nRD;              /* S5933 RD# Input */
Pin    =    nPTRDY;           /* Pass-thru Access Complete Indicator */
                             /* ADR6:2 are hardwired to the PTDATA address */
                             /* HOLD output is no longer required */
                             /* nRDY_OUT becomes nPTRDY */
                             /* SELECT# and BE3:0# on the S5933 may be tied low */


/* Declarations and Intermediate Variable Definitions */

/* Remove all equations for HOLD, HLDA, nRDY_OUT */

BE3.OE      = 'b'1;           /* DMA controller is only add-on bus master */
BE2.OE      = 'b'1;
BE1.OE      = 'b'1;
BE0.OE      = 'b'1;
nPTRDY.OE = 'b'1;
nADS.OE     = 'b'1;
W_nR.OE     = 'b'1;


/* Logic Equations */

/* The bus add-on bus master state machine and the add-on slave */
/* state machine change to support pass-thru programming (pt-writes) and */
/* no add-on processor */
/* Remove all equations for HOLD, HLDA, nRDY_OUT */

nPTADR = !SLAVE1;
nRD = !(SLAVE2 # SLAVE3);
nPTRDY = !(SLAVE2 # SLAVE3);
```

```
SEQUENCE master {

present M0
      if ((RD & !RDEMPTY) # (WR & !WRFULL)) & !DONE next M1;

present M1  /* THIS IS THE ONLY STATE WHICH CHANGES */
      if !SLAVE0 next M1;     /* Do not interrupt a controller register access */
      if SLAVE0 next M2;      /* No register access, so continue */
   •
   •
   •
}

SEQUENCE slave {

present S0        /* Idle State */
      /* If the pass-thru access is occurring and */
      /* the master state machine is in the idle */
      /* state, begin the sequence. */

      if !nPTATN & MASTER0 next S1;
      if nPTATN # !MASTER0 next S0;

present S1        /* PTADR# Address Phase */
      /* PTADR# is asserted in this phase. Address decode assumes */
      /* an 8 byte region is defined for pass-thru (BADR4), and only A2 is decoded */

      if PTNUM0 & PTNUM1 & A2 next S2;    /* Address register write */
      if PTNUM0 & PTNUM1 & !A2     next S3;    /* Transfer count write */
      default next S0;

present S2        /* Address Access Data Phase - Assert PTRDY# and RD# */
      next S0;

present S3        /* Transfer Count Access Data Phase - Assert PTRDY# and RD#*/
      next S0;

/* No recovery phase is required, as the S5933 does not require it */
}
```

A

4.0 PLD EQUATIONS

The following code is written in CUPL™ by Logical Devices. Pin assignments will depend on the programmable logic device the code is programmed into. The code compiles and simulates without errors.

```
Name        DMA_CON;
Partno      ;
Date        5/19/95;
Revision    0;
Designer    JMW;
Company     AMCC;
Assembly    ;
Location    ;
Device      ;


/******************************************************************/
/* Add-on DMA Controller                                          */
/* The PLD equations below compile and have been simulated and are*/
/* believed to be correct.  AMCC assumes no liability for errors  */
/* made in the code or logic                                      */
/*                                                                */
/*                                                                */
/******************************************************************/
/*  Allowable Target Device Types:                                */
/******************************************************************/


/**  Inputs  **/

Pin         = BPCLK     ;      /* Buffered PCI Clock                  */
Pin         = RDEMPTY   ;      /* PCI to Add-on FIFO Empty Indicator */
Pin         = WRFULL    ;      /* Add-on to PCI FIFO Full Indicator  */
Pin         = HLDA      ;      /* CPU Hold Acknowledge                */
Pin         = [A7..0]   ;      /* Address/Data Bus 7:0        */
Pin         = [A15..8]  ;      /* Address/Data Bus 15:8       */
Pin         = [A23..16] ;      /* Address/Data Bus 23:16      */
Pin         = [A31..24] ;      /* Address/Data Bus 31:24      */
Pin         = nRDY_IN   ;      /* DMA Controller Ready Input   */
Pin         = nRESET    ;      /* S5933 Reset Output           */

/**  Outputs  **/

Pin         = HOLD      ;      /* CPU Hold                     */
Pin         = W_nR      ;      /* Bidirectional WR/RD          */
Pin         = nRDY_OUT  ;      /* Ready Output (to CPU)        */
Pin         = nBLAST    ;      /* Last Data Phase of Burst Indicator */
Pin         = nADS      ;      /* Bidirectional Address Strobe  */
Pin         = nRDFIFO   ;      /* S5933 FIFO Read Input        */
Pin         = nWRFIFO   ;      /* S5933 FIFO Write Input       */
Pin         = BE3       ;      /* Byte Enable Output 3         */
Pin         = BE2       ;      /* Byte Enable Output 2         */
Pin         = BE1       ;      /* Byte Enable Output 1         */
Pin         = BE0       ;      /* Byte Enable Output 0         */
```

```
/** Declarations and Intermediate Variable Definitions **/

/* State machine bits are defined as nodes.  The MA bits are the  */
/* MASTER state machine and the SL bits are the SLAVE state */
/* machine.  The 18-bit TXCNT register is composed of C17:2.  It  */
/* is divided into two groups to allow a easier fit into some     */
/* FPGAs. RWCONT is an additional bit in TXCNT which indicates if */
/* the DMA transfer is to be a S5933 read or write.  nREADY is a  */
/* registered version of nRDY_IN used in the logic for generating */
/* nRDFIFO and nWRFIFO.                                           */

NODE [C7..2];            /* TXCNT Bits 2-7     */
NODE [C17..8];           /* TXCNT Bits 8-15  */
NODE [MA2..0];           /* MASTER State Machine Bits */
NODE [SL1..0];           /* SLAVE State Machine Bits */
NODE RWCONT;             /* DMA Read/Write Indicator */
NODE nREADY;             /* Registered nRDY_IN Signal */

[C7..2].AR = !nRESET;   /* Reset all registers/state machines to zero */
[C17..8].AR = !nRESET;
[MA2..0].AR = !nRESET;
[SL1..0].AR = !nRESET;
RWCONT.AR = !nRESET;
nREADY.AR='b'0;

[C7..2].AP ='b'0;
[C17..8].AP ='b'0;
RWCONT.AP ='b'0;
[MA2..0].AP = 'b'0;
[SL1..0].AP = 'b'0;
nREADY.AP=!nRESET;

[C7..2].CK = BPCLK;      /* All reg's/state machines clocked w/BPCLK */
[C17..8].CK = BPCLK;
nREADY.CK = BPCLK;
[SL1..0].CK = BPCLK;
[MA2..0].CK = BPCLK;
[A7..0].CK = BPCLK;
[A15..8].CK = BPCLK;
[A23..16].CK = BPCLK;
[A31..24].CK = BPCLK;
BE3.CK = BPCLK;
BE2.CK = BPCLK;
BE1.CK = BPCLK;
BE0.CK = BPCLK;
RWCONT.CK =BPCLK;
```

A

```
BE3.OE = HLDA;              /* HLDA indicates that the DMA controller may */
BE2.OE = HLDA;              /* become bus master.  All outputs are enabled*/
BE1.OE = HLDA;              /* when HLDA is asserted */
BE0.OE = HLDA;
nRDY_OUT.OE = HLDA;
nBLAST.OE = HLDA;
nADS.OE = HLDA;
[A7..0].OE = MASTER2;    /* Address outputs only enabled during MASTER2*/
[A15..8].OE = MASTER2;   /* which is the address phase of the transfer */
[A23..16].OE = MASTER2;
[A31..24].OE = MASTER2;
W_nR.OE = HLDA;


/* Shorthand notation for address and transfer count values */


BYTE0 = A7 & A6 & A5 & A4 & A3 & A2;
BYTE1 = A15 & A14 & A13 & A12 & A11 & A10 & A9 & A8;
BYTE2 = A13 & A22 & A21 & A20 & A19 & A18 & A17 & A16;
CBYTE0 = !C7 & !C6 & !C5 & !C4 & !C3 & !C2;
CBYTE1 = !C17 & !C16 & !C15 & !C14 & !C13 & !C12 & !C11 & !C10 & !C9 & !C8;


/* Need to know when count is below 4 data phases (double-words) for */
/* the MASTER state machine */


ONELEFT = CBYTE1 & !C7 & !C6 & !C5 & !C4 & !C3 & C2;
TWOLEFT = CBYTE1 & !C7 & !C6 & !C5 & !C4 &  C3 & !C2;
THREELEFT = CBYTE1 & !C7 & !C6 & !C5 & !C4 & C3 & C2;


/* A read indicates an S5933 read and an add-on Write.  A write */
/* indicates an S5933 write and an add-on read                  */


RD = RWCONT;
WR = !RWCONT;


/* STOP indicates a condition where the FIFO cannot support      */
/* further transfers (empty or full) and the current burst must */
/* terminate.                                                    */

STOP = (RD & RDEMPTY) # (WR & WRFULL);


/* DONE indicates the transfer count is zero and transfers should */
/* stop.                                                          */


DONE = CBYTE0 & CBYTE1;


/* Increment address and decrement TC after each completed data  */
/* phase.  Do not count if the read FIFO is empty or the write   */
/* FIFO is full.  Run a dummy cycle and begin again when the     */
/* transfer can continue.  This assumes a 1 clock RDY pulse.  CNT */
/* only toggles when the DMA controller is local bus master */
/* (HLDA=1).                                                     */
```

```
CNT = !nRDY_IN & !STOP & HLDA & nRESET &
      (MASTER3 # MASTER4 # MASTER5 # MASTER6);


LOADA =  W_nR.IO & SLAVE1;    /* Load Address Register        */
LOADC =  W_nR.IO & SLAVE2;    /* Load Transfer Count Register  */


/**  Logic Equations  **/

nREADY.D = nRDY_IN;           /* Used for RDFIFO#/WRFIFO# Timing */

[BE3..0] = 'b'0000;           /* All Add-on accesses are 32-bits */
[A31..25] = 'b'0000000;       /* A31..22, A1..0 Driven to zero   */
A24 = A24.IO & !HLDA;
[A23..22] = 'b'00;
[A1..0] = 'b'00;


/* Toggle output when CNT is asserted and all previous bits are  */
/*'1' or when a LOAD is performed and the load value (.IO)        */
/* is different from the existing value.                         */

Field ADDR_COUNTA = [A7..2];

A2.T = CNT # LOADA & (A2 $ A2.IO);
A3.T = CNT & A2 # LOADA & (A3 $ A3.IO);
A4.T = CNT & A2 & A3 # LOADA & (A4 $ A4.IO);
A5.T = CNT & A2 & A3 & A4 # LOADA & (A5 $ A5.IO);
A6.T = CNT & A2 & A3 & A4 & A5 # LOADA & (A6 $ A6.IO);
A7.T = CNT & A2 & A3 & A4 & A5 & A6 # LOADA & (A7 $ A7.IO);

Field ADDR_COUNTB = [A15..8];

A8.T = CNTB2 # LOADA & (A8 $ A8.IO);
A9.T = CNTB2 & A8 # LOADA & (A9 $ A9.IO);
A10.T = CNTB2 & A8 & A9 # LOADA & (A10 $ A10.IO);
A11.T = CNTB2 & A8 & A9 & A10 # LOADA & (A11 $ A11.IO);
A12.T = CNTB2 & A8 & A9 & A10 & A11 # LOADA & (A12 $ A12.IO);
A13.T = CNTB2 & A8 & A9 & A10 & A11 & A12 # LOADA & (A13 $ A13.IO);
A14.T = CNTB2 & A8 & A9 & A10 & A11 & A12 & A13 # LOADA & (A14 $ A14.IO);
A15.T = CNTB2 & A8 & A9 & A10 & A11 & A12 & A13 & A14 # LOADA & (A15 $ A15.IO);

/* Assert CNTB2 when CNT is asserted and A2-7 are set */

CNTB2 = CNT & (A2 & A3 & A4 & A5 & A6 & A7);

Field ADDR_COUNTC = [A21..16];

A16.T = CNTC2 # LOADA & (A16 $ A16.IO);
A17.T = CNTC2 & A16 # LOADA & (A17 $ A17.IO);
A18.T = CNTC2 & A16 & A17 # LOADA & (A18 $ A18.IO);
A19.T = CNTC2 & A16 & A17 & A18 # LOADA & (A19 $ A19.IO);
A20.T = CNTC2 & A16 & A17 & A18 & A19 # LOADA & (A20 $ A20.IO);
A21.T = CNTC2 & A16 & A17 & A18 & A19 & A20 # LOADA & (A21 $ A21.IO);
```

A

```
/* Assert CNTC2 when CNTB2 is asserted and A8-15 are set */

CNTC2 = CNTB2 & (A8 & A9 & A10 & A11 & A12 & A13 & A14 & A15);

/* Toggle output when CNT is asserted and all previous bits are   */
/* '1' or when a LOAD is performed and the load value (.IO) is     */
/* different from the existing value. The transfer count is a      */
/* byte count.                                                     */

Field XFER_COUNTA = [C7..2];

C2.T = CNT # LOADC & (C2 $ A2.IO);
C3.T = CNT & !C2 # LOADC & (C3 $ A3.IO);
C4.T = CNT & !C2 & !C3 # LOADC & (C4 $ A4.IO);
C5.T = CNT & !C2 & !C3 & !C4 # LOADC & (C5 $ A5.IO);
C6.T = CNT & !C2 & !C3 & !C4 & !C5 # LOADC & (C6 $ A6.IO);
C7.T = CNT & !C2 & !C3 & !C4 & !C5 & !C6 # LOADC & (C7 $ A7.IO);

/* Assert CNTA when CNT is asserted and C2-7 are clear */

CNTA = CNT & !(C2#C3#C4#C5#C6#C7);

Field XFER_COUNTB = [C17..8];

C8.T = CNTA # LOADC & (C8 $ A8.IO);
C9.T = CNTA & !C8 # LOADC & (C9 $ A9.IO);
C10.T = CNTA & !C8 & !C9 # LOADC & (C10 $ A10.IO);
C11.T = CNTA & !C8 & !C9 & !C10 # LOADC & (C11 $ A11.IO);
C12.T = CNTA & !C8 & !C9 & !C10 & !C11 # LOADC & (C12 $ A12.IO);
C13.T = CNTA & !C8 & !C9 & !C10 & !C11 & !C12 # LOADC & (C13 $ A13.IO);
C14.T = CNTA & !C8 & !C9 & !C10 & !C11 & !C12 & !C13 # LOADC & (C14 $ A14.IO);
C15.T = CNTA & !C8 & !C9 & !C10 & !C11 & !C12 & !C13 & !C14 # LOADC & (C15 $
A15.IO);
C16.T = CNTA & !C8 & !C9 & !C10 & !C11 & !C12 & !C13 & !C14 & !C15 # LOADC & (C16 $
A16.IO);
C17.T = CNTA & !C8 & !C9 & !C10 & !C11 & !C12 & !C13 & !C14 & !C15 & !C16 # LOADC &
(C17 $ A17.IO);


/* This bit identifies a read vs. a write DMA 1 = read, 0 = write */

RWCONT.T = LOADC & (RWCONT $ A24.IO);

FIELD master = [MA2..0];
$DEFINE M0      'b'000
$DEFINE M1      'b'001
$DEFINE M2      'b'011
$DEFINE M3      'b'010
$DEFINE M4      'b'110
$DEFINE M5      'b'111
$DEFINE M6      'b'101
$DEFINE M7      'b'100
```

```
MASTER0 = !MA2 & !MA1 & !MA0         ;
MASTER1 = !MA2 & !MA1 &  MA0         ;
MASTER2 = !MA2 &  MA1 &  MA0         ;
MASTER3 = !MA2 &  MA1 & !MA0         ;
MASTER4 =  MA2 &  MA1 & !MA0         ;
MASTER5 =  MA2 &  MA1 &  MA0         ;
MASTER6 =  MA2 & !MA1 &  MA0         ;
MASTER7 =  MA2 & !MA1 & !MA0         ;

FIELD slave = [SL1..0];
$DEFINE S0        'b'00
$DEFINE S1        'b'01
$DEFINE S2        'b'10
$DEFINE S3        'b'11

SLAVE0 = !SL1 & !SL0         ;
SLAVE1 = !SL1 &  SL0         ;
SLAVE2 =  SL1 & !SL0         ;
SLAVE3 =  SL1 &  SL0         ;

/* State Machine Support Logic                                */
/* BLAST# can be generated by a condition where the FIFO cannot  */
/* provide or accept data, the fourth data phase of any burst, or */

nBLAST = !(((MASTER3 # MASTER4 # MASTER5) & STOP) # MASTER6);

/* Assert FIFO strobe for entire data phase, except first clock, */
/* unless the FIFO cannot support the transaction.         */

nRDFIFO = !(HLDA & nREADY & RD & !RDEMPTY &
            (MASTER3 # MASTER4 # MASTER5 # MASTER6));

nWRFIFO = !(HLDA & nREADY & WR & !WRFULL &
            (MASTER3 # MASTER4 # MASTER5 # MASTER6));

/* Write/Read strobe for add-on logic */

W_nR = RWCONT;

/* Drive address in master state 2 */

nADS = !MASTER2;

/* Drive RDY in slave state one */

nRDY_OUT = !(SLAVE1 # SLAVE2);

/* Assert HOLD to the CPU when a DMA request is received from the */
/* S5933 FIFO.                                                 */

HOLD = !MASTER0 & nRESET;
```

A

```
/******************************************************************/
/* State machine for slave accesses to program ADDR and TXCNT    */
/******************************************************************/

SEQUENCE slave {

/* Idle State */
present S0
        if !nADS.IO & !A21.IO & A20.IO & !HLDA next S1;
        if !nADS.IO &  A21.IO & A20.IO & !HLDA next S2;
        default next S0;

/* Address Access Data Phase */
present S1
        next S3;

/* Transfer Count Access Data Phase */
present S2
        next S3;

/* Recovery Phase */
present S3
        next S0;
}


/******************************************************************/
/* State machine for master access to perform DMA transfers      */
/******************************************************************/

SEQUENCE master {

/* Idle State If a DMA request occurs and TC is not zero, begin   */
/* DMA process.                                                   */
present M0
        if ((RD & !RDEMPTY) # (WR & !WRFULL)) & !DONE next M1;

/* Assert CPU HOLD, wait for acknowledge  */
present M1
        if HLDA next M2;
        if !HLDA next M1;

/* Address Phase */
present M2
        if ONELEFT next M6;     /* 1 Data phase left */
        if TWOLEFT next M5;     /* 2 Data phases left */
        if THREELEFT next M4;   /* 3 Data phases left */
        default next M3;        /* Count is 4 or more */

/* Data Phase 1 */
present M3
        if nRDY_IN next M3;
        if !nRDY_IN next M4;
```

```
/* Data Phase 2 */
present M4
        if nRDY_IN next M4;
        if !STOP & !nRDY_IN next M5;
        if STOP & !nRDY_IN next M7;

/* Data Phase 3 */
present M5
        if nRDY_IN next M5;
        if !STOP & !nRDY_IN next M6;
        if STOP & !nRDY_IN next M7;

/* Data Phase 4 */
present M6
        if !STOP & !nRDY_IN next M7;
        if nRDY_IN next M6;
        if STOP & !nRDY_IN next M7;

/* Recovery Phase */
present M7
        if (nRDY_IN & RD & !RDEMPTY & !DONE) next M2;
        if (nRDY_IN & WR & !WRFULL & !DONE) next M2;
        if (DONE # STOP) & nRDY_IN next M0;
        if !nRDY_IN next M7;
}
```

A

5.0   PLD SIMULATIONS

The following simulation is the output file from the CSIM logic simulator, simulating the PLD code in Appendix A.

CSIM(TD): CUPL Simulation Program
Version 4.5a Serial# ED-32930421
Copyright (c) 1983, 1994 Logical Devices, Inc.
CREATED Wed May 24 14:19:48 1995

LISTING FOR SIMULATION FILE: dma_con4.si

```
 1: Name         DMA;
 2: Partno       ;
 3: Date         5/19/95;
 4: Revision     0;
 5: Designer     JMW;
 6: Company      AMCC;
 7: Assembly     ;
 8: Location     ;
 9: Device       ;
10:
11: /*****************************************************************/
12: /*                                                             */
13: /*****************************************************************/
14: /** Allowable Target Device Types :                  */
15: /*****************************************************************/
16:
17: FIELD ADDR = [A31..0];
18:
19: FIELD ADDR_COUNTA = [A7,A6,A5,A4,A3,A2];
20: FIELD ADDR_COUNTB = [A15,A14,A13,A12,A11,A10,A9,A8];
21: FIELD ADDR_COUNTC = [A21,A20,A19,A18,A17,A16];
22: FIELD XFER_COUNTA = [C7,C6,C5,C4,C3,C2];
23: FIELD XFER_COUNTB = [C17,C16,C15,C14,C13,C12,C11,C10,C9,C8];
24: FIELD master = [MA2,MA1,MA0];
25: FIELD slave = [SL1,SL0];
26:                                          FIELD
ADDR=[A31,A30,A29,A28,A27,A26,A25,A24,A23,A22,A21,A20,A19,A18,A17,A16,A15,A14,
     A13,A12,A11,A10,A9,A8,A7,A6,A5,A4,A3,A2,A1,A0];
27:
28: ORDER:
29:
30: nRESET,%1,BPCLK,%1,RDEMPTY,WRFULL,%1,HLDA,%1,nBLAST,%1,nRDY_IN,%1,nADS,%3,
     nWRFIFO,nRDFIFO,%1,ADDR,%1,HOLD,%1,W_nR,%1,MA2,MA1,MA0,%1,SL1,SL0,%1,DONE,%1,
     RWCONT,%1,LOADA,LOADC,%1,CNT;
31:
```

```
================================================================================
          R       n      nn
     n    DW    n R     WR                                          R
     R B  ER    B D     RD                                            W LL
     E P  MF  H L Y n   FF                           H W           D C OO
     S C  PU  L A _ A   II                           O _ MMM  SS  O O AA  C
     E L  TL  D S I D   FF                           L n AAA  LL  N N DD  N
     T K  YL  A T N S   OO            ADDR           D R 210  10  E T AC  T
================================================================================
RESET DEVICE
0001: 0 C 11 0 Z 1 1   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ L Z LLL LL H L LL L
0002: 0 C 11 0 Z 1 1   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ L Z LLL LL H L LL L
0003: 1 C 11 0 Z 1 1   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ L X LLL LL H L LL L
0004: 1 C 11 0 Z 1 1   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ L X LLL LL H L LL L
WRITE DMA ADDRESS = 8080
0005: 1 C 11 0 Z 1 0   HH ZZZZZZZ0Z001000000000000000000ZZ L 1 LLL LH H L HL L
0006: 1 C 11 0 Z 1 1   HH ZZZZZZZ0Z000000001000000001000000ZZ L 1 LLL HH H L LL L
0007: 1 C 11 0 Z 1 1   HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL H L LL L
0008: 1 C 11 0 Z 1 1   HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL H L LL L
WRITE DMA TRANSFER COUNT (READ FROM S5933)
0009: 1 C 11 0 Z 1 0   HH ZZZZZZZ0Z011000000000000000000ZZ L 1 LLL HL H L LH L
0010: 1 C 11 0 Z 1 1   HH ZZZZZZZ1Z000000000000000001111ZZ L 1 LLL HH L H LL L
0011: 1 C 11 0 Z 1 1   HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL L H LL L
0012: 1 C 11 0 Z 1 1   HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL L H LL L
0013: 1 C 11 0 Z 1 1   HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL L H LL L
0014: 1 C 01 0 Z 1 1   HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX H 1 LLH LL L H LL L
0015: 1 C 01 0 Z 1 1   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H 1 LLH LL L H LL L
1ST BURST OF ACCESS:   ADDRESS = 8080, TC = 15
0016: 1 C 01 1 H 1 L   HH LLLLLLLLLLLLLLLLHLLLLLLLLHLLLLLLL H H LHH LL L H LL L
0017: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0018: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0019: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0020: 1 C 01 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHL LL L H LL H
```

A

```
=================================================================================
        R         n      nn
    n   DW    n R    WR                                              R
    R B ER    B D    RD                                              W LL
    E P MF  H L Y n  FF                              H W        D C OO
    S _ PU  L A _ A  II                              O _ MMM SS O O AA C
    E L TL  D S I D  FF                              L n AAA LL N N DD N
    T K YL  A T N S  OO              ADDR            D R 210 10 E T AC T
=================================================================================
0021: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHL LL L H LL L
0022: 1 C 01 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHH LL L H LL H
0023: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHH LL L H LL L
0024: 1 C 01 1 L 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLH LL L H LL H
0025: 1 C 01 1 L 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLH LL L H LL L
0026: 1 C 01 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLL LL L H LL L
0027: 1 C 01 1 H 1 L   HH LLLLLLLLLLLLLLLLLHLLLLLLLLHLLHLLLL H H LHH LL L H LL L
2ND BURST OF ACCESS:  ADDRESS = 8084, TC = 11
0028: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0029: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0030: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0031: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0032: 1 C 01 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHL LL L H LL H
0033: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHL LL L H LL L
0034: 1 C 01 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHH LL L H LL H
0035: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHH LL L H LL L
0036: 1 C 01 1 L 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLH LL L H LL H
0037: 1 C 01 1 L 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLH LL L H LL L
0038: 1 C 01 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLL LL L H LL L
0039: 1 C 01 1 H 1 L   HH LLLLLLLLLLLLLLLLLHLLLLLLLLHLHLLLLL H H LHH LL L H LL L
3RD BURST OF ACCESS:  ADDRESS = 8088, TC = 7
0040: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0041: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0042: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0043: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0044: 1 C 01 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHL LL L H LL H
0045: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHL LL L H LL L
0046: 1 C 01 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHH LL L H LL H
0047: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHH LL L H LL L
0048: 1 C 01 1 L 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLH LL L H LL H
0049: 1 C 01 1 L 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLH LL L H LL L
0050: 1 C 01 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLL LL L H LL L
0051: 1 C 01 1 H 1 L   HH LLLLLLLLLLLLLLLLLHLLLLLLLLHLHHLLLL H H LHH LL L H LL L
4TH BURST OF ACCESS:  ADDRESS = 808C, TC = 3
0052: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHL LL L H LL L
0053: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHL LL L H LL L
0054: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHL LL L H LL L
0055: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHL LL L H LL L
0056: 1 C 01 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHH LL L H LL H
0057: 1 C 01 1 H 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHH LL L H LL L
0058: 1 C 01 1 L 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLH LL L H LL H
0059: 1 C 01 1 L 1 H   HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLH LL L H LL L
0060: 1 C 01 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLL LL H H LL L
```

```
0061: 1 C 01 1 H 1 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ L H LLL LL H H LL L
0062: 1 C 01 1 H 1 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ L H LLL LL H H LL L
0063: 1 C 01 1 H 1 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ L H LLL LL H H LL L
WRITE DMA ADDRESS
0064: 1 C 11 0 Z 1 0   HH ZZZZZZZ0Z0010000000000000000000ZZ L 1 LLL LH H H HL L
0065: 1 C 11 0 Z 1 1   HH ZZZZZZZ0Z0000000010101010100ZZ L 1 LLL HH H H LL L
0066: 1 C 11 0 Z 1 1   HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL H H LL L
WRITE DMA TRANSFER COUNT (WRITE TO S5933)
0067: 1 C 11 0 Z 1 0   HH ZZZZZZZ0Z0110000000000000000000ZZ L 1 LLL HL H H LH L
0068: 1 C 11 0 Z 1 1   HH ZZZZZZZ0Z0000000000000000000101ZZ L 1 LLL HH L L LL L
0069: 1 C 11 0 Z 1 1   HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL L L LL L
0070: 1 C 11 0 Z 1 1   HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL L L LL L
0071: 1 C 10 0 Z 1 1   HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX H 1 LLH LL L L LL L
0072: 1 C 10 0 Z 1 1   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H 1 LLH LL L L LL L
1ST BURST OF ACCESS:   ADDRESS = 5550, TC = 5
0073: 1 C 10 1 H 1 L   HH LLLLLLLLLLLLLLLLLHLHLHLHLHLHLLLL H L LHH LL L L LL L
0074: 1 C 10 1 H 1 H   LH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L LHL LL L L LL L
0075: 1 C 10 1 H 1 H   LH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L LHL LL L L LL L
0076: 1 C 10 1 H 1 H   LH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L LHL LL L L LL L
0077: 1 C 10 1 H 1 H   LH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L LHL LL L L LL L
0078: 1 C 10 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L HHL LL L L LL H
0079: 1 C 10 1 H 1 H   LH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L HHL LL L L LL L
0080: 1 C 10 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L HHH LL L L LL H
0081: 1 C 10 1 H 1 H   LH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L HHH LL L L LL L
0082: 1 C 10 1 L 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L HLH LL L L LL H
0083: 1 C 10 1 L 1 H   LH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L HLH LL L L LL L
0084: 1 C 10 1 H 0 H   HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L HLL LL L L LL L
0085: 1 C 10 1 H 1 L   HH LLLLLLLLLLLLLLLLLHLHLHLHLHHLLLLL H L LHH LL L L LL L
2ND BURST OF ACCESS:   ADDRESS = 5554, TC = 1
```

```
===============================================================================
          R         n       nn
   n      DW    n  R        WR                                          R
   R  B   ER    B  D        RD                                          W  LL
   E  P   MF H  L  Y  n     FF                     H  W          D  C   OO
   S  C   PU L  A  _  A     II                     O  _  MMM SS  O  O   AA  C
   E  L   TL D  S  I  D     FF                      L  n  AAA LL  N  N   DD  N
   T  K   YL A  T  N  S     OO            ADDR       D  R  210 10  E  T   AC  T
================================================================================
0086: 1 C 10 1 L 1 H    LH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L HLH LL L L LL L
0087: 1 C 10 1 L 1 H    LH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L HLH LL L L LL L
0088: 1 C 10 1 L 1 H    LH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L HLH LL L L LL L
0089: 1 C 10 1 L 1 H    LH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L HLH LL L L LL L
0090: 1 C 10 1 H 0 H    HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H L HLL LL H L LL L
0091: 1 C 10 1 H 1 H    HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ L L LLL LL H L LL L

RDEMPTY ASSERTED DURING A READ TRANSFER

WRITE DMA ADDRESS = 8080
0092: 1 C 11 0 Z 1 0    HH ZZZZZZZ0Z0010000000000000000000ZZ L 1 LLL LH H L HL L
0093: 1 C 11 0 Z 1 1    HH ZZZZZZZ0Z0000000100000001000000ZZ L 1 LLL HH H L LL L
0094: 1 C 11 0 Z 1 1    HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL H L LL L
0095: 1 C 11 0 Z 1 1    HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL H L LL L
WRITE DMA TRANSFER COUNT (READ FROM S5933)
0096: 1 C 11 0 Z 1 0    HH ZZZZZZZ0Z0110000000000000000000ZZ L 1 LLL HL H L LH L
0097: 1 C 11 0 Z 1 1    HH ZZZZZZZ1Z0000000000000000001111ZZ L 1 LLL HH L H LL L
0098: 1 C 11 0 Z 1 1    HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL L H LL L
0099: 1 C 11 0 Z 1 1    HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL L H LL L
0100: 1 C 11 0 Z 1 1    HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX L 1 LLL LL L H LL L
0101: 1 C 01 0 Z 1 1    HH XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX H 1 LLH LL L H LL L
0102: 1 C 01 0 Z 1 1    HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H 1 LLH LL L H LL L
1ST BURST OF ACCESS:   ADDRESS = 8080, TC = 15
0103: 1 C 01 1 H 1 L    HH LLLLLLLLLLLLLLLLHLLLLLLLLHLLLLLLLL H H LHH LL L H LL L
0104: 1 C 01 1 H 1 H    HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0105: 1 C 01 1 H 1 H    HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0106: 1 C 01 1 H 1 H    HL ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H LHL LL L H LL L
0107: 1 C 11 1 L 0 H    HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHL LL L H LL L
0108: 1 C 11 1 L 1 H    HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HHL LL L H LL L
0109: 1 C 11 1 H 0 H    HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ H H HLL LL L H LL L
0110: 1 C 11 1 H 1 H    HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ L H LLL LL L H LL L
0111: 1 C 11 1 H 1 H    HH ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ L H LLL LL L H LL L
```

Introduction

This appendix provides information useful to software developers working with AMCC's S5933 PCI controller. It describes a software method of interfacing a PCI device using the S5933 in a host computer; this method consists of a series of C-language routines to access the PCI device using the PCI BIOS. This appendix also demonstrates several ways to make use of the special features of the S5933 PCI controller, including mailboxes, FIFO transfers, bus master transfers, nvRAM access and endian conversion.

Organization

A library of C-language callable routines are presented for managing a PCI device using a S5933 PCI controller, organized in a logical progression of implementation. This progression begins with PCI BIOS initialization, device initialization, and finally proceeds to data transfer via PIO and Bus Master DMA methods.

A software program to manage a PCI device may use the PCI BIOS as specified in the PCI BIOS Specification. This interface provides a hardware-independent method of communicating with PCI devices in a host computer. A C-language callable interfacing to the PCI BIOS is shown in this document.

The source code presented in this document is written entirely in Borland C/C++™ Version 3.1. Other vendor's C-language compilers can be used, with the appropriate modifications for low-level register access and software interrupt generation. Program fragments and calling sequences are shown throughout. The complete source code for the PCI BIOS interface routines is contained at the end of Appendix A.

Low-Level Access via Borland C

The source code uses the PCI BIOS to access the device in a hardware-independent manner. Using the PCI BIOS, however, necessitates low-level access to processor registers and software interrupts. Borland C/C++ provides easy methods of accessing registers using the pseudo-register variables and software interrupts using the function geninterrupt().

A hardware register can be accessed via the pseudo-register variables. These pseudo-registers are formed by pre-pending an underscore to the register name in uppercase. For example, the AX register is accessed with the variable name _AX.

Caution: The register is not guaranteed to remain as assigned. This requires careful examination of the resulting assembler output. All of the routines in this document have been verified to retain the values of the pseudo-registers until used by the PCI BIOS interrupt or saved in a temporary variable in the necessary order. If the routines are subjected to ANY optimizations, the resulting assembler code must be examined to insure the registers are still valid.

Also, some of the PCI BIOS routines require the use of the 32-bit registers (EAX, EBX, ECX, EDX). This means that the (-3) option on the BCC command line option must be used. The following command compiles the routines described in this appendix:

BCC -3 PCILIB.C

Invoking a software interrupt is also easy in Borland C/C++ using the library function geninterrupt(). The syntax to invoke the PCI BIOS software interrupt is:

geninterrupt(0x1a);

Identification of PCI Devices

A set of three PCI BIOS functions may be used to determine the existence of a specific device on the PCI bus. These functions include pci_bios_present, find_pci_device and find_pci_class_code. The pci_bios_present function determines the presence of the PCI BIOS in the host computer. If this function indicates that the PCI BIOS does not exist, then the remainder of the PCI BIOS functions will not work. The find_pci_device function determines if a specific device exists in the PCI system. The find_pci_class_code determines if a device with a specific class code exists in the system. These three functions are described in detail in the following sections.

pci_bios_present

The primary purpose of this function is to determine the presence of the PCI BIOS in the host computer. Secondary purposes are to determine various PCI hardware characteristics of the host computer. A C-language prototype for this function is:

```
int pci_bios_present(byte *hardware_mechanism,
        word *interface_level_version,
        byte *last_pci_bus_number);
```

The return value from this function indicates if the PCI BIOS is present on the host system. If the value SUCCESSFUL is returned, the PCI BIOS is present, otherwise not.

B

There are no inputs to the pci_bios_present() function.

The first output parameter, hardware_mechanism, identifies the hardware characteristics supported by the host computer. Bit 0 indicates if Mechanism #1 is supported. Bit 1 indicates if Mechanism #2 is supported. Bit 4 indicates if Special Cycle is supported via Config Mechanism #1. Bit 5 indicates if generating Special Cycle is available via Mechanism #2. The specific mechanism is supported if the respective bit is set (1), and reset (0) otherwise. Mechanism #1 is used for multiple PCI buses. (See **PCI LOCAL BUS SPECIFICATION.**)

The second output parameter, interface_level_version, indicates the PCI BIOS version level. The version is stored in Binary Coded Decimal (BCD). For example, Version 1.35 would be stored as 0x0135. The current version of the PCI BIOS is version 2.00, and therefore the value 0x0200 will be returned.

The final output parameter, last_pci_bus_number, indicates the bus number of the last PCI bus on the system. The first bus on the system will have a bus_number of 0, the second a bus_number of 1, etc.

find_pci_device

The purpose of the find_pci_device function is to determine the existence and physical location of a specific device, given a specific Vendor ID and Device ID. A C-language prototype for this function is:

```
int find_pci_device(word device_id,
        word vendor_id,
        word index,
        byte *bus_number,
        byte *device_and_function)
```

The return code from this function indicates whether the device was found, indicated by the return value of SUCCESSFUL. The function may also return the value DEVICE_NOT_FOUND, indicating that the specific device was not found. If the value BAD_VENDOR_ID is returned, a vendor ID of 0xffff was specified, which is invalid. Finally, a return value of NOT_SUCCESSFUL indicates a PCI BIOS error has occurred.

The first input parameter, device_id, is a number in the range 0 to 65535 indicating the specific device ID the program is to locate. This number is allocated by the vendor, indicating a specific device in the vendor's product line.

The second input parameter, vendor_id, is a number in the range 0 to 65534 indicating the manufacturer of the device. Vendor IDs are allocated by the PCI SIG to ensure uniqueness. The value 0xffff is an invalid ID for the Vendor ID.

The final input parameter, index, is a number in the range 0 to N indicating which device is to be found. For example, if there are two devices with the same Device and Vendor IDs, an index value of zero would locate the first device and an index value of one would locate the second value. An index value of two would return a code of DEVICE_NOT_FOUND.

The first output parameter, bus_number, is a number in the range 0 to 255 indicating which PCI bus the device is located on.

The second output parameter, device_and_function, indicates the Device Number in the upper 5 bits and the Function Number in the lower 3 bits.

The bus_number and device_and_function values returned from this function must be used to access the PCI device via the PCI BIOS function calls.

find_pci_class_code

The purpose of the find_pci_class_code function is to determine the existence and physical location of a specific device, given a specific Class Code. A C-language prototype for this function is:

```
int find_pci_class_code(dword class_code,
        word index,
        byte *bus_number,
        byte *device_and_function)
```

The return code from this function indicates whether the device was found, indicated by the return value of SUCCESSFUL. The function may also return the value DEVICE_NOT_FOUND, indicating that the specific device was not found. Finally, a return value of NOT_SUCCESSFUL indicates a PCI BIOS error has occurred.

The first input parameter, class_code, is a number 0-4095, which is divided into three bytes. The first byte specifies the base class, which broadly defines the type of function of the device. The second byte specifies the sub-class code, which more specifically defines the function of the device. Finally, the third byte indicates which register-level programming (if any) is supported by the device. The base class codes are indicated in Section 3.6 of this manual.

For specific sub-class codes and register-level programming interfaces, see **PCI LOCAL BUS SPECIFICATION** (or Section 3.6 of this manual).

The final input parameter, index, is a number in the range 0 to N indicating which device of a specific class code to be found. For example, if there are two devices with the same Class Code, an index value of zero would locate the first device and an index value of one would locate the second value. An index value of two returns a code of DEVICE_NOT_FOUND.

The first output parameter, bus_number, is a number in the range 0 to 255 indicating which PCI bus the device is located on.

The second output parameter, device_and_function, indicates the Device Number in the upper 5 bits and the Function Number in the lower 3 bits. The bus_number and device_and_function values returned from this function must be used to access the PCI device via the PCI BIOS function calls.

Putting It All Together

Code Segment 1 determines if the PCI BIOS is present and, if so, determines if a PCI device with AMCC's Vendor ID and a Device ID of zero exists. An appropriate message is displayed at each stage of the program fragment.

Note that the fragment shows NULL pointers as parameters to the pci_bios_present function. Each of the functions in the library has the feature of allowing a NULL pointer to be passed, indicating that the parameter is not needed and therefore is not assigned to any variable.

Code Segment 1

```
int bios_present;          /* TRUE if PCI BIOS found */
byte bus;                  /* Bus number of device */
byte device_and_function;  /* Device Number bits 7-3, Function bits 2-0 */

/* Determine if PCI BIOS present */
if (pci_bios_present(NULL, NULL, NULL) == SUCCESSFUL) {
        bios_present = TRUE;
}
else {
        printf("PCI BIOS not present\n");
        bios_present = FALSE;
}

if (bios_present) {

        /* Locate AMCC Vendor ID and Device ID of 0 */
        if (find_pci_device(0,0x10e8,0,&bus,&device_and_function) == SUCCESSFUL) {
                printf("AMCC Device found: bus = %d device = %d function = %d\n",
                        bus, (device_and_function >> 3), (device_and_function & 0x7));
        }
        else {
                printf("Device not found!!!\n");
        }
}
```

B

AMCC

Access to Configuration Space

Each PCI device contains an address space specifically devoted to PCI buses, used for device identification and memory and I/O configuration. A set of six functions provides access to the PCI device's Configuration Space. These functions provide for reading and writing of a byte (8 bits), double-word (16 bits) or double-word (32 bits) in the Configuration Space. C-language prototypes for these functions are:

```
int read_configuration_byte(byte bus_number,
      byte device_and_function,
      byte register_number,
      byte *byte_read);

int read_configuration_word(byte bus_number,
      byte device_and_function,
      byte register_number,
      word *word_read);

int read_configuration_dword(byte bus_number,
      byte device_and_function,
      byte register_number,
      dword *dword_read);

int write_configuration_byte(byte bus_number,
      byte device_and_function,
      byte register_number,
      byte byte_to_write);

int write_configuration_word(byte bus_number,
      byte device_and_function,
      byte register_number,
      word word_to_write);

int write_configuration_dword(byte bus_number,
      byte device_and_function,
      byte register_number,
      dword dword_to_write);
```

The return code from each of these functions indicates whether the Configuration Space was read or written successfully, indicated by the return value of SUCCESSFUL. The function may also return the value BAD_REGISTER_NUMBER, indicating that a register number greater than 255 was requested to be read or written. Finally, a return value of NOT_SUCCESSFUL indicates a PCI BIOS error has occurred.

Each of these functions has input parameters of bus_number and device_and_function. These parameters MUST be the values returned by the find_pci_bios() or find_pci_class_code() functions for the desired device.

Also, each of the functions has an input parameter of register_number. This number is a number in the range 0 to 255 indicating which register in the Configuration Space is to be read or written. The pre-defined register numbers are described in the Configuration Space section of this appendix.

The read_configuration_byte/word/dword functions have an output parameter that contains the byte/word/dword read from the Configuration Space.

As an example, register 0 contains the Vendor ID of the device and is one word (16 bits) long. To obtain the Vendor ID from the device, simply use the following function call:

```
#define PCI_CS_VENDOR_ID  0
int read_configuration_word(bus_number, device_and_function,
      PCI_CS_VENDOR_ID, &vendor_id);
```

The return value from the function MUST be checked for the value SUCCESSFUL prior to using the vendor_id value, otherwise the value is invalid.

Obtaining Interrupt Vector

The Interrupt Line Configuration Space register contains the hardware interrupt assigned to the device by the host computer (if needed). The valid number in this register is in the range 0 to 15. A value of FFh indicates that no hardware interrupt is needed by the device. Multiple devices may be assigned to a single hardware interrupt by the host computer. Device drivers for PCI devices **must** determine if the device it is servicing generated the interrupt and if not, "chain" or call the previous interrupt handler to handle the interrupt.

The hardware interrupt number can be translated into a software interrupt number via the following table:

Hardware Interrupt	Interrupt Number
IRQ0	08h
IRQ1	09h
IRQ2	0Ah
IRQ3	0Bh
IRQ4	0Ch
IRQ5	0Dh
IRQ6	0Eh
IRQ7	0Fh
IRQ8	70h
IRQ9	71h
IRQ10	72h
IRQ11	73h
IRQ12	74h
IRQ13	75h
IRQ14	76h
IRQ15	77h

Code Segment 2

```
void interrupt ( *oldhandler)(void);

void interrupt handler(void)
{
   /* Must determine if PCI device needs servicing */

   /* call the old routine */
   oldhandler();
}

main()
{
   byte bus;                   /* Bus number of device */
   byte device_and_function;   /* Device Number bits 7-3, Function bits 2-0 */
   byte interrupt_line;        /* Interrupt assigned to PCI device */
   byte interrupt_vector;      /* Software interrupt vector */

   /* Must Determine bus and device_and_function of device first */

   if (read_configuration_byte(bus, device_and_function,
       PCI_CS_INTERRUPT_LINE, &interrupt_line) == SUCCESSFUL) {
     if (interrupt_line != 0xff) {
       if (interrupt_line < 8) {
          interrupt_vector = 0x08 + interrupt_line;
       }
       else {
          interrupt_vector = 0x70 + (interrupt_line - 8);
       }
       /* save the old interrupt vector */
       oldhandler = getvect(interrupt_vector);
       /* install the new interrupt handler */
       setvect(interrupt_vector, handler);
     }
}
```

Code Segment 2 determines the interrupt vector from a PCI device and installs an interrupt handler which calls the previous interrupt handler.

First it defines an interrupt handler. This handler only calls the previous interrupt handler. A method for determining if a shared interrupt is to be serviced by this interrupt handler is not defined by PCI. AMCC suggests the use of an Operation Register — Interrupt Control/Status Register to determine if the device has an interrupt pending.

The main routine assumes that the bus and device_and_function variables are determined as described in a previous section. It then reads the Interrupt Line Configuration Space register and converts the interrupt line into a software interrupt vector. The current interrupt vector routine is then determined by the Borland C/C++ getvect() function and the new interrupt handler installed with the setvect() function.

Obtaining Base Addresses

The six Base Address Registers in the Configuration Space contain the hardware memory or I/O addresses assigned to the device by the host computer. The first Base Address (0) on a S5933 is dedicated to the Operation Registers. The remaining five base address registers are available for use by the PCI device.

The six Base Address DWORD Registers are located in the Configuration Space at addresses: 10h, 14h, 18h, 1Ch, 20h and 24h. Each Base Address may describe either a Memory or I/O Base Address. The Memory Base Address is indicated by bit zero cleared and I/O indicated by bit zero set. To determine the actual base address, bits 3-0 on a Memory address must be masked off and bits 1-0 on an I/O address must be masked off. These bits indicate various information about the memory that is generally not important for this discussion. See the **PCI LOCAL BUS SPECIFICATION** (or Section 3.1.11 of this manual).

Code Segment 3 demonstrates the use of the PCI
Base Address registers.

Code Segment 3

```
byte bus;                      /* Bus number of device */
byte device_and_function;      /* Device Number bits 7-3, Function bits 2-0 */
dword base_address;            /* Base Address */

/* Must Determine bus and device_and_function of device first */

if (read_configuration_dword(bus, device_and_function,
     PCI_CS_BASE_ADDRESS_1, &base_address)) {
  /* Determine if Memory Address */
  if (base_address & 0x01) {
    /* Mask off bits 3-0 */
    base_address &= 0xF0;
    printf("Memory Base Address = %x\n", base_address);
  }
  else {
    /* Mask off bits 1-0 */
    base_address &= 0xFC;
    printf("I/O Base Address = %x\n", base_address);
  }
}
```

This program fragment reads a double-word from the
Base Address 0 configuration space register. Bit 0 of
the returned value is then examined to determine if
the base address describes a memory or an I/O
space and either 2 or 4 bits masked off to determine
the actual base address.

The memory Base Address is a 32-bit address in
physical memory. To use this address from a real-
mode program under DOS, it must be converted to a
segment:offset format in order to access the memory
via a C pointer. This method will only work if the
memory is mapped below the 1-megabyte memory-

addressing limit of real-mode programs (20 bits). To
convert the base address to a segment:offset pointer,
one needs to extract bits 0-3 as the offset and shift-
right the upper bits four places to obtain the segment.
The program fragment performs this conversion.
(See Code Segment 4.)

First, the program fragment defines a far pointer to a
structure. The actual definition of the structure will
depend upon the particular application. Next, it ob-
tains the segment and offset from the base address.
Once this is determined, one can use the MK_FP
macro to convert the segment:offset to a far pointer.

Code Segment 4

```
dword base_address;              /* AMCC Operation Registers base address */
word base_address_segment;       /* Bits 20-4 of base address */
word base_address_offset;        /* Bits 3-0 of base address */
char far *ptr;

/* Obtain base address first */

/* Parse out segment and offset of base address */
base_address_segment = base_address >> 4;
base_address_offset = base_address & 0xf;

/* Make far pointer */
ptr = MK_FP(base_address_segment, base_address_offset);
```

S5933 PCI Controller Special Features

The S5933 PCI Controller contains a number of special features to assist the PCI device developer. These features include mailboxes, FIFO (First-In-First-Out) under program control, and FIFO under bus master control. These features are accessed via the Operation Registers. By default, the Operation Registers are mapped in I/O space. The base address of the Operation Registers may be determined as described in the **Obtaining Base Addresses** section.

I/O space can be accessed via the Borland C/C++ functions outp(), inp(), outpw() and inpw() for byte and word access respectively. Unfortunately, Borland C/C++ does not have double-word I/O functions; therefore, the following C functions are contained in the AMCC PCI Library: inpd() and outpd(). The C-language prototypes for these functions are:

```
void outpd(word port, dword value);
dword inpd(word port);
```

The first input parameter to both of these functions, port, is the hardware port to be read/written. The outpd() function has an additional parameter that is the double-word value to be written to the hardware port. The inpd() function will return the double-word value read from the hardware port.

The offsets of each of the Operation Registers are shown in the table below:

Offset	Abbreviation	Register Name
00h	OMB1	Outgoing Mailbox Register #1
04h	OMB2	Outgoing Mailbox Register #2
08h	OMB3	Outgoing Mailbox Register #3
0ch	OMB4	Outgoing Mailbox Register #4
10h	IMB1	Incoming Mailbox Register #1
14h	IMB2	Incoming Mailbox Register #2
18h	IMB3	Incoming Mailbox Register #3
1Ch	IMB4	Incoming Mailbox Register #4
20h	FIFO	Bidirectional FIFO Register
24h	MWAR	Master Write Address Register
28h	MWTC	Master Write Transfer Count
2ch	MRAR	Master Read Address Register
30h	MRTC	Master Read Transfer Count Register
34h	MBEF	Mailbox Empty/Full Status Register
38h	INTCSR	Interrupt Control/Status Register
3Ch	MCSR	Bus Master Control/Status Register

Incoming/Outgoing Mailbox Registers

The Incoming/Outgoing mailbox registers provide a method for sending command or parameter data to/from the PCI device. The Outgoing Mailbox Registers are double-word registers that may be used for sending a command and parameters to the device. The Incoming Mailbox Registers are read-only registers that may be used to return limited data back to the host computer. An AMCC-suggested command to make use of the Incoming/Outgoing Mailbox Registers would be a command to return a Company and device name. This could be used to ensure that the Vendor ID/Device ID combination is truly the expected device.

The status of each of the Incoming/Outgoing Mailbox Registers may be determined by examining the Mailbox Empty-Full Status register (MBEF). Each nibble (4 bits) indicates the status of each byte of each Incoming/Outgoing Mailbox Register. A bit set in the Incoming Mailbox Register Status register indicates that a byte has been sent from the PCI device that has not yet been read. A bit set in the Outgoing Mailbox Register Status Register indicates that a byte has been sent and the PCI device has not yet read that byte. The following table indicates the status bits corresponding to each Incoming/Outgoing Mailbox Register.

MBEF Bits	Description
0-3	Outgoing Mailbox #1
4-7	Outgoing Mailbox #2
8-11	Outgoing Mailbox #3
12-15	Outgoing Mailbox #4
16-19	Incoming Mailbox #1
20-23	Incoming Mailbox #2
24-27	Incoming Mailbox #3
28-31	Incoming Mailbox #4

Each bit within each nibble indicates the status of the corresponding byte of the Incoming/Outgoing Mailbox Register (Bit 0=Byte 0, Bit 1=Byte 1, Bit 2=Byte 2, Bit 3=Byte 3).

Code Segment 5 demonstrates sending a command to a PCI device and waiting for the response via polling.

B

Code Segment 5

```
#define IN_MAILBOX_1_FULL 0x000f0000

dword amcc_op_reg_base_address;    /* Operation Register Base Address */
dword name;                         /* Response to IDENTIFY command */
dword mailbox_status;               /* Mailbox Empty-Full Status */

if (read_configuration_dword(bus, device_and_function,
    PCI_CS_BASE_ADDRESS_0, &amcc_op_reg_base_address)) {

  /* Assume I/O base address, Mask off bits 1-0 */
  amcc_op_reg_base_address &= 0xFFFC;

  /* Send command via Outgoing Mailbox #1 */
  outpd(amcc_op_reg_base_address + AMCC_OP_REG_OMB1, IDENTIFY_COMMAND);

  /* Wait for all bytes in Incoming Mailbox to be valid */
  do {
    mailbox_status = inpd(amcc_op_reg_base_address + AMCC_OP_REG_MBEF);
  } while((mailbox_status & IN_MAILBOX_1_FULL) != IN_MAILBOX_1_FULL);

  /* Read Incoming Mailbox #1 */
  name = inpd(amcc_op_reg_base_address + AMCC_OP_REG_IMB1);

  /* Need to check if name matches expected response */
}
```

Code Segment 5 first defines a bitmask to AND with the MBEF register to determine if all bytes of the Incoming mailbox register are valid. The base address of the S593 operation registers is then obtained and assumed to be an I/O address to simplify the code. Next, a command (IDENTIFY_COMMAND) is sent to the Outgoing Register #1. This command is, as are all mailbox commands, vendor defined. The Mailbox Empty-Full Status Register is then polled until all bytes of the Incoming Mailbox #1 are valid. Typically, the code will need to have a timeout loop, as all devices will not respond to this command. Finally, the Incoming Mailbox #1 Register is read. This value must be validated against a known return value.

Code Segment 5 used polling to obtain the command results. The S5933 PCI Controller also allows the code to be interrupt-driven. The Interrupt Control-Status Register is used to indicate to the AMCC PCI controller which byte in an Incoming mailbox register will cause an interrupt to be generated.

FIFO Register Transfers

The AMCC PCI Controller also has a First-In-First-Out (FIFO) transfer feature. This allows the host computer to transfer data to the PCI Device via an I/O memory transfer. This type of transfer does make use of the processor, so the processor performance will be severely affected. Bus master transfers (described later) do not use the processor for the data transfer (other than the setup).

The I/O transfer could use a FOR loop to send/receive each byte/word/dword to the FIFO, although this would be slow. A better method would be to use an 80x86 processor instruction that allows for an I/O string move. Borland C/C++ does not have a function that incorporates these instructions, so the AMCC PCI Library contains six functions that do. The prototypes for these functions are shown below.

Function Prototypes

```
void insb(word port, void *buf, int count);  /* Reads Bytes */
void insw(word port, void *buf, int count); /* Reads Words */
void insd(word port, void *buf, int count); /* Reads DWords */

void outsb(word port, void *buf, int count); /* Writes Bytes */
void outsw(word port, void *buf, int count); /* Writes Words */
void outsd(word port, void *buf, int count); /* Writes DWords */
```

Code Segment 6

```
dword read_data[10];     /* Data to be read from PCI */
dword write_data[10];    /* Data to be written to PCI */

/* Retrieve data from PCI device */
insd(amcc_op_reg_base_address + AMCC_OP_REG_FIFO,
    read_data, sizeof(read_data) / sizeof(dword));


/* Send data to PCI device */
outsd(amcc_op_reg_base_address + AMCC_OP_REG_FIFO,
    write_data, sizeof(write_data) / sizeof(dword));
```

The first parameter to each of these functions, port, is the hardware port to read/write. The second parameter, buf, is an address of a buffer to be read/written to the hardware port. The final parameter, count, is the number of bytes/words/dwords to be read/written.

The code to implement a FIFO transfer is relatively simple. The following program fragment shows transfers in both directions (read and write PCI Device). See Code Segment 6.

First, read and write arrays are declared to be retrieved/sent to the PCI device. The function insd() is then used to retrieve data from the FIFO on the PCI device and the function outsd() is used to send the data to the FIFO. The only precaution here is that the count parameter is the number of DWORDS to be transferred, not the number of bytes, therefore sizeof(read_data) must be divided by the size of a double-word. The Bus Master Control Register may be used to determine the fullness of the inbound and outbound FIFOs.

Bus-Master Transfers

The S5933 also has a bus master transfer feature. This allows the host computer to transfer data to the PCI device without consuming processor bandwidth for the actual transfer, only for setting up the transfer, possibly monitoring the transfer, and processing the interrupt that may be generated when the transfer is complete.

Bus master transfers are more difficult to set up than a FIFO transfer. They also have the limitation that the transfer must begin on a DWORD boundary. If the data to be transferred does not begin on a DWORD boundary, the bytes preceding the boundary could be sent/received via the mailbox registers. Bus master transfers also have the advantage that both a read and a write may take place at the same time (time-sharing the PCI bus, of course).

B

Setting up a bus master transfer includes determining the physical address at which the data transfer is to take place. To determine a physical address from a real-mode segment and offset, simply shift-left the segment by 4 bits and add to the offset. This will produce a 20-bit address. To access memory above the 1-megabyte real-mode addressing limit, a physical address also must be determined. This is entirely dependent upon the DOS Extender utilized and is not addressed here.

Code Segment 7 demonstrates how to set up a Bus-Master write and poll the PCI device to determine when the transfer is complete.

First, the Master Write Address Register and Master Write Transfer Count Register must be set to the physical address of the data and the number of BYTES to be transferred respectively. The address must be a DWORD boundary. The transfer count is always the number of bytes to be transferred, and does not have to be whole multiples of a DWORD (0, 1, 2, 3, 4, ... are all legal).

Next, the transfer is started by setting the WRITE_TRANSFER_ENABLE bit in the Bus-Master Control-Status Register. The WRITE_TRANSFER_ COMPLETE bit in the Interrupt Control Status Register is then polled until set. Upon completion of the transfer, the interrupt must be cleared by writing a one to the WRITE_TRANSFER_COMPLETE bit.

The transfer does not have to be polled until complete. The transfer completion may generate an interrupt if set up in the Interrupt Control Status Register — Interrupt on Write Complete bit.

There are a few things to keep in mind with regard to Bus-Master transfers. First, if you desire to start both a read and a write bus-master transfer concurrently, make sure the write to the Read/Write Transfer Enable first examines the other bit (Write/Read Transfer Enable respectively) to see if the transfer is still executing. If so, you must set both bits, or the writing of the bit to start the transfer may stop the other transfer from executing. Second, a FIFO transfer of the same type (Read/Write) may not be executed at the same time as a Bus-Master transfer, because both FIFO and Bus-Master transfers utilize the same FIFO.

There are several parameters that may be set to "tune" the Bus-Master transfer. These parameters are located in the Bus Master Control Status Register and are described in the S5933 PCI Controller Hardware Specification Manual, and will not be repeated here.

Non-Volatile Memory Access

The on-board non-volatile memory device (EEPROM or nvRAM) on the PCI device with an S5933 PCI Controller may be accessed from the host computer. The Bus Master Control-Status Register is used to access the non-volatile memory. The access makes

Code Segment 7

```
#define READ_TRANSFER_ENABLE    0x00004000
#define WRITE_TRANSFER_COMPLETE 0x00040000
#define READ_TRANSFER_COMPLETE  0x00080000

/* Set write from address */
outpd(amcc_op_reg_base_address + AMCC_OP_REG_MWAR,
    ((dword) (FP_SEG(write_data) << 4)) + FP_OFF(write_data));

/* Set transfer count */
outpd(amcc_op_reg_base_address + AMCC_OP_REG_MWTC, sizeof(write_data));

/* Enable write transfer in MCSR */
outpd(amcc_op_reg_base_address + AMCC_OP_REG_MCSR, WRITE_TRANSFER_ENABLE);

/* Wait until write transfer is complete */
while ((inpd(amcc_op_reg_base_address + AMCC_OP_REG_INTCSR)
     & WRITE_TRANSFER_COMPLETE) == 0) {
  /* Do nothing */
}

/* Clear Interrupt */
outpd(amcc_op_reg_base_address + AMCC_OP_REG_INTCSR, WRITE_TRANSFER_COMPLETE);
```

use of the bits 31-29 in the BCSR register as the "command" to be sent to the memory. The bits 23-16 in the BCSR register are also used in conjunction with these command bits to indicate high address, low address, or data read/written to the non-volatile memory. The non-volatile memory commands are contained in the table shown below.

These "commands" are written to the BCSR register to indicate the option to be performed on the memory. Both a read and a write must first load the low and high address bytes. For example, to access the location 0x1234 in the non-volatile memory, the value 0x34 will be written to bits 23-16 and the value

100 (binary) written to bits 31-29. This will be followed by the value 0x12 written to bits 23-16 and the value 101 (binary) in bits 31-29. The actual Begin Read or Begin Write command may then be sent. The Begin Write command must also place the value to be written in bits 23-16. Bit 31 on reads indicates whether the command is complete. This bit must be polled until clear, which indicates that the command is complete. Upon completion, the data value has been written, or is available for reading from bits 23-16. Code Segment 8 demonstrates a method to access the non-volatile memory.

The non-volatile memory may also be accessed from the add-on side of the S5933. The memory must, however, only be accessed from one side at a time. The host software and the add-on software must then negotiate to determine who has access to the memory. This could be accomplished via a command/status using the mailbox registers.

Bit 31	Bit 30	Bit 29	Command
1	0	0	Load low address byte
1	0	1	Load high address byte
1	1	0	Begin Write
1	1	1	Begin Read

Code Segment 8

```
#define NVRAM_BUSY       0x80      /* Bit 31 indicates if device busy */

#define NVCMD_LOAD_LOW    (0x4 << 5)   /* nvRAM Load Low command */
#define NVCMD_LOAD_HIGH   (0x5 << 5)   /* nvRAM Load High command */
#define NVCMD_BEGIN_READ  (0x7 << 5)   /* nvRAM Begin Read command */
#define NVCMD_BEGIN_WRITE (0x6 << 5)   /* nvRAM Begin Write command */

#define AMCC_OP_REG_MCSR_NVDATA  (AMCC_OP_REG_MCSR + 2) /* Data in byte 2 */
#define AMCC_OP_REG_MCSR_NVCMD   (AMCC_OP_REG_MCSR + 3) /* Command in byte 3 */

byte nv_data;         /* Data read from non-volatile memory */

/* Wait for nvRAM not busy */
while ((inp(amcc_op_reg_base_address + AMCC_OP_REG_MCSR + 3) & NVRAM_BUSY) == 1) {
   /* Busy wait */
}

/* Load Low address */
outp(amcc_op_reg_base_address + AMCC_OP_REG_MCSR_NVCMD, NVCMD_LOAD_LOW);
outp(amcc_op_reg_base_address + AMCC_OP_REG_MCSR_NVDATA, 0x34);

/* Load High address */
outp(amcc_op_reg_base_address + AMCC_OP_REG_MCSR_NVCMD, NVCMD_LOAD_HIGH);
outp(amcc_op_reg_base_address + AMCC_OP_REG_MCSR_NVDATA, 0x12);

/* Send Begin Read command */
outp(amcc_op_reg_base_address + AMCC_OP_REG_MCSR_NVCMD, NVCMD_BEGIN_READ);

/* Wait for nvRAM not busy */
while ((inp(amcc_op_reg_base_address + AMCC_OP_REG_MCSR + 3) & NVRAM_BUSY) == 1) {
   /* Busy wait */
}

/* Get data from nvRAM Data register */
nv_data = inp(amcc_op_reg_base_address + AMCC_OP_REG_MCSR_NVDATA);
```

B

Glossary

Byte	A single 8-bit unit of memory.
Bus Master Transfer	A method of transferring data to/from a PCI device without consuming processor bandwidth.
Bus Number	A number (0-255) that selects a specific bus within a host computer.
DWORD	A single 32-bit unit of memory (Double Word).
Class Code	A number (0-4095) that specifies the generic function of the device. Each byte of the class code more precisely indicates the function of the device.
Configuration Space	An address space specifically devoted to PCI buses. It is used for device identification and memory and I/O configuration.
Device ID	A predefined field in the configuration space that uniquely identifies a device (along with the Vendor ID).
Device Number	A number (0-31) that uniquely selects a device on a PCI bus.
FIFO Transfer	A method of transferring data to/from a PCI device using a processor.
Function Number	A number (0-7) that selects a function within a multi-function device.
Mailbox	An AMCC method of transferring command/status information to a PCI device.
PCI	Peripheral Component Interconnect bus.
PCI BIOS	A software method of interfacing with PCI via a standardized methodology.
PIO	Programmed Input/Output. A method of data transfer.
Vendor ID	A predefined field that uniquely identifies a device (along with the Device ID).
WORD	A single 16-bit unit of memory.

```
/******************************************************************************/
/*                                                                            */
/* Module: AMCC.H                                                             */
/*                                                                            */
/* Purpose: Definitions for AMCC PCI Library                                  */
/*                                                                            */
/******************************************************************************/


/******************************************************************************/
/*   General Constants                                                        */
/******************************************************************************/

#define TRUE      1
#define FALSE     0

typedef unsigned char byte;     /* 8-bit  */
typedef unsigned short word;    /* 16-bit */
typedef unsigned long dword;    /* 32-bit */

#define CARRY_FLAG 0x01          /* 80x86 Flags Register Carry Flag bit */


/******************************************************************************/
/*    PCI Functions                                                           */
/******************************************************************************/

#define PCI_FUNCTION_ID                       0xb1
#define PCI_BIOS_PRESENT             0x01
#define FIND_PCI_DEVICE              0x02
#define FIND_PCI_CLASS_CODE     0x03
#define READ_CONFIG_BYTE                      0x08
#define READ_CONFIG_WORD             0x09
#define READ_CONFIG_DWORD       0x0a
#define WRITE_CONFIG_BYTE       0x0b
#define WRITE_CONFIG_WORD                     0x0c
#define WRITE_CONFIG_DWORD                    0x0d


/******************************************************************************/
/*    PCI Return Code List                                                    */
/******************************************************************************/

#define SUCCESSFUL              0x00
#define NOT_SUCCESSFUL          0x01
#define FUNC_NOT_SUPPORTED      0x81
#define BAD_VENDOR_ID           0x83
#define DEVICE_NOT_FOUND        0x86
#define BAD_REGISTER_NUMBER     0x87


/******************************************************************************/
/*    PCI Configuration Space Registers                                       */
/******************************************************************************/

#define PCI_CS_VENDOR_ID        0x00
#define PCI_CS_DEVICE_ID        0x02
#define PCI_CS_COMMAND          0x04
#define PCI_CS_STATUS           0x06
#define PCI_CS_REVISION_ID      0x08
#define PCI_CS_CLASS_CODE       0x09
```

B

```
#define  PCI_CS_CACHE_LINE_SIZE   0x0c
#define  PCI_CS_MASTER_LATENCY    0x0d
#define  PCI_CS_HEADER_TYPE       0x0e
#define  PCI_CS_BIST              0x0f
#define  PCI_CS_BASE_ADDRESS_0    0x10
#define  PCI_CS_BASE_ADDRESS_1    0x14
#define  PCI_CS_BASE_ADDRESS_2    0x18
#define  PCI_CS_BASE_ADDRESS_3    0x1c
#define  PCI_CS_BASE_ADDRESS_4    0x20
#define  PCI_CS_BASE_ADDRESS_5    0x24
#define  PCI_CS_EXPANSION_ROM     0x30
#define  PCI_CS_INTERRUPT_LINE    0x3c
#define  PCI_CS_INTERRUPT_PIN     0x3d
#define  PCI_CS_MIN_GNT           0x3e
#define  PCI_CS_MAX_LAT           0x3f


/*************************************************************************/
/*    AMCC Operation Register Offsets                                    */
/*************************************************************************/

#define  AMCC_OP_REG_OMB1         0x00
#define  AMCC_OP_REG_OMB2         0x04
#define  AMCC_OP_REG_OMB3         0x08
#define  AMCC_OP_REG_OMB4         0x0c
#define  AMCC_OP_REG_IMB1         0x10
#define  AMCC_OP_REG_IMB2         0x14
#define  AMCC_OP_REG_IMB3         0x18
#define  AMCC_OP_REG_IMB4         0x1c
#define  AMCC_OP_REG_FIFO         0x20
#define  AMCC_OP_REG_MWAR         0x24
#define  AMCC_OP_REG_MWTC         0x28
#define  AMCC_OP_REG_MRAR         0x2c
#define  AMCC_OP_REG_MRTC         0x30
#define  AMCC_OP_REG_MBEF         0x34
#define  AMCC_OP_REG_INTCSR       0x38
#define  AMCC_OP_REG_MCSR         0x3c
#define  AMCC_OP_REG_MCSR_NVDATA  (AMCC_OP_REG_MCSR + 2) /* Data in byte 2 */
#define  AMCC_OP_REG_MCSR_NVCMD   (AMCC_OP_REG_MCSR + 3) /* Command in byte 3 */


/*************************************************************************/
/*    AMCCLIB Prototypes                                                 */
/*************************************************************************/

int pci_bios_present(byte *hardware_mechanism,
                     word *interface_level_version,
                     byte *last_pci_bus_number);

int find_pci_device(word device_id,
                    word vendor_id,
                    word index,
                    byte *bus_number,
                    byte *device_and_function);

int find_pci_class_code(dword class_code,
                    word index,
                    byte *bus_number,
                    byte *device_and_function);
```

```
int read_configuration_byte(byte bus_number,
                            byte device_and_function,
                            byte register_number,
                            byte *byte_read);

int read_configuration_word(byte bus_number,
                            byte device_and_function,
                            byte register_number,
                            word *word_read);

int read_configuration_dword(byte bus_number,
                             byte device_and_function,
                             byte register_number,
                             dword *dword_read);

int write_configuration_byte(byte bus_number,
                             byte device_and_function,
                             byte register_number,
                             byte byte_to_write);

int write_configuration_word(byte bus_number,
                             byte device_and_function,
                             byte register_number,
                             word word_to_write);

int write_configuration_dword(byte bus_number,
                              byte device_and_function,
                              byte register_number,
                              dword dword_to_write);

void outpd(word port, dword value);

dword inpd(word port);

void insb(word port, void *buf, int count);
void insw(word port, void *buf, int count);
void insd(word port, void *buf, int count);

void outsb(word port, void *buf, int count);
void outsw(word port, void *buf, int count);
void outsd(word port, void *buf, int count);
```

B

```
/**************************************************************************/
/*                                                                        */
/* Module: AMCCLIB.C                                                      */
/*                                                                        */
/* Purpose: Define a C interface to the PCI BIOS                          */
/*                                                                        */
/* Functions Defined:                                                     */
/*                                                                        */
/*     PCI_BIOS_PRESENT                                                   */
/*     FIND_PCI_DEVICE                                                    */
/*     FIND_PCI_CLASS_CODE                                                */
/*     READ_CONFIGURATION_BYTE                                            */
/*     READ_CONFIGURATION_WORD                                            */
/*     READ_CONFIGURATION_DWORD                                           */
/*     WRITE_CONFIGURATION_BYTE                                           */
/*     WRITE_CONFIGURATION_WORD                                           */
/*     WRITE_CONFIGURATION_DWORD                                          */
/*     OUTPD                                                              */
/*     INPD                                                               */
/*     INSB                                                               */
/*     INSW                                                               */
/*     INSD                                                               */
/*     OUTSB                                                              */
/*     OUTSW                                                              */
/*     OUTSD                                                              */
/*                                                                        */
/* Local Functions                                                        */
/*                                                                        */
/*     READ_CONFIGURATION_AREA                                            */
/*     WRITE_CONFIGURATION_AREA                                           */
/*                                                                        */
/**************************************************************************/


/**************************************************************************/
/*                                                                        */
/* Include Files                                                          */
/*                                                                        */
/**************************************************************************/
#include <dos.h>
#include <stddef.h>

#include "amcc.h"


/**************************************************************************/
/*                                                                        */
/* Local Prototypes                                                       */
/*                                                                        */
/**************************************************************************/

static int read_configuration_area(byte function,
                                    byte bus_number,
                                    byte device_and_function,
                                    byte register_number,
                                    dword *data);

static int write_configuration_area(byte function,
                                     byte bus_number,
                                     byte device_and_function,
```

```
                                   byte register_number,
                                   dword value);

/****************************************************************************/
/*                                                                        */
/* Define macros to obtain individual bytes from a word register          */
/*                                                                        */
/****************************************************************************/


#define HIGH_BYTE(ax) (ax >> 8)
#define LOW_BYTE(ax) (ax & 0xff)



/****************************************************************************/
/*                                                                        */
/*   PCI_BIOS_PRESENT                                                      */
/*                                                                        */
/* Purpose: Determine the presence of the PCI BIOS                        */
/*                                                                        */
/* Inputs: None                                                           */
/*                                                                        */
/* Outputs:                                                               */
/*                                                                        */
/*    byte *hardware_mechanism                                            */
/*        Identifies the hardware characteristics used by the platform.   */
/*        Value not assigned if NULL pointer.                             */
/*            Bit 0 - Mechanism #1 supported                              */
/*            Bit 1 - Mechanism #2 supported                              */
/*                                                                        */
/*    word *interface_level_version                                       */
/*       PCI BIOS Version - Value not assigned if NULL pointer.           */
/*                High Byte - Major version number                        */
/*                Low Byte  - Minor version number                        */
/*                                                                        */
/*       byte *last_pci_bus_number                                        */
/*           Number of last PCI bus in the system. Value not assigned if NULL*/
/*       pointer                                                          */
/*                                                                        */
/*    Return Value - Indicates presence of PCI BIOS                       */
/*        SUCCESSFUL - PCI BIOS Present                                   */
/*        NOT_SUCCESSFUL - PCI BIOS Not Present                           */
/*                                                                        */
/****************************************************************************/

int pci_bios_present(byte *hardware_mechanism,
                             word *interface_level_version,
                             byte *last_pci_bus_number)
{
    int ret_status;             /* Function Return Status. */
    byte bios_present_status;/* Indicates if PCI bios present */
    dword pci_signature;        /* PCI Signature ('P', 'C', 'I', ' ') */
    word ax, bx, cx, flags;  /* Temporary variables to hold register values */

    /* Load entry registers for PCI BIOS */
    _AH = PCI_FUNCTION_ID;
    _AL = PCI_BIOS_PRESENT;

    /* Call PCI BIOS Int 1Ah interface */
```

```
      geninterrupt(0x1a);

      /* Save registers before overwritten by compiler usage of registers */
      ax = _AX;
      bx = _BX;
      cx = _CX;
      pci_signature = _EDX;
      flags = _FLAGS;
      bios_present_status = HIGH_BYTE(ax);


      /* First check if CARRY FLAG Set, if so, BIOS not present */
      if ((flags & CARRY_FLAG) == 0) {

         /* Next, must check that AH (BIOS Present Status) == 0 */
         if (bios_present_status == 0) {

            /* Check bytes in pci_signature for PCI Signature */
            if ((pci_signature & 0xff)          == 'P' &&
                ((pci_signature >> 8) & 0xff)   == 'C' &&
                ((pci_signature >> 16) & 0xff)  == 'I' &&
                ((pci_signature >> 24) & 0xff)  == ' ') {

               /* Indicate to caller that PCI bios present */
               ret_status = SUCCESSFUL;

               /* Extract calling parameters from saved registers */
               if (hardware_mechanism != NULL) {
                  *hardware_mechanism = LOW_BYTE(ax);
               }
               if (hardware_mechanism != NULL) {
                  *interface_level_version = bx;
               }
               if (hardware_mechanism != NULL) {
                  *last_pci_bus_number = LOW_BYTE(cx);
               }
            }
         }
         else {
            ret_status = NOT_SUCCESSFUL;
         }
      }
      else {
         ret_status = NOT_SUCCESSFUL;
      }

      return (ret_status);
   }

/***************************************************************************/
/*                                                                         */
/*   FIND_PCI_DEVICE                                                       */
/*                                                                         */
/* Purpose: Determine the location of PCI devices given a specific Vendor  */
/*          Device ID and Index number.  To find the first device, specify */
/*          0 for index, 1 in index finds the second device, etc.          */
/*                                                                         */
/* Inputs:                                                                 */
```

B

```
/*                                                                          */
/*     word device_id                                                       */
/*         Device ID of PCI device desired                                  */
/*                                                                          */
/*     word vendor_id                                                       */
/*         Vendor ID of PCI device desired                                  */
/*                                                                          */
/*     word index                                                           */
/*         Device number to find (0 - (N-1))                                */
/*                                                                          */
/* Outputs:                                                                 */
/*                                                                          */
/*     byte *bus_number                                                     */
/*         PCI Bus in which this device is found                            */
/*                                                                          */
/*     byte *device_and_function                                            */
/*         Device Number in upper 5 bits, Function Number in lower 3 bits   */
/*                                                                          */
/*     Return Value - Indicates presence of device                         */
/*         SUCCESSFUL - Device found                                        */
/*         NOT_SUCCESSFUL - BIOS error                                      */
/*         DEVICE_NOT_FOUND - Device not found                             */
/*         BAD_VENDOR_ID - Illegal Vendor ID (0xffff)                       */
/*                                                                          */
/**************************************************************************/

int find_pci_device(word device_id,
                    word vendor_id,
                    word index,
                    byte *bus_number,
                    byte *device_and_function)
{
    int ret_status;      /* Function Return Status */
    word ax, bx, flags; /* Temporary variables to hold register values */

    /* Load entry registers for PCI BIOS */
    _CX = device_id;
    _DX = vendor_id;
    _SI = index;
    _AH = PCI_FUNCTION_ID;
    _AL = FIND_PCI_DEVICE;

    /* Call PCI BIOS Int 1Ah interface */
    geninterrupt(0x1a);

    /* Save registers before overwritten by compiler usage of registers */
    ax = _AX;
    bx = _BX;
    flags = _FLAGS;

    /* First check if CARRY FLAG Set, if so, error has occurred */
    if ((flags & CARRY_FLAG) == 0) {

        /* Get Return code from BIOS */
        ret_status = HIGH_BYTE(ax);
        if (ret_status == SUCCESSFUL) {

            /* Assign Bus Number, Device & Function if successful */
```

```
            if (bus_number != NULL) {
                *bus_number = HIGH_BYTE(bx);
            }
            if (device_and_function != NULL) {
                *device_and_function = LOW_BYTE(bx);
            }
        }
    }
    else {
      ret_status = NOT_SUCCESSFUL;
    }

    return (ret_status);
}

/******************************************************************************/
/*                                                                          */
/*   FIND_PCI_CLASS_CODE                                                     */
/*                                                                          */
/* Purpose: Returns the location of PCI devices that have a specific Class  */
/*          Code.                                                           */
/*                                                                          */
/* Inputs:                                                                  */
/*                                                                          */
/*     word class_code                                                      */
/*         Class Code of PCI device desired                                 */
/*                                                                          */
/*     word index                                                           */
/*         Device number to find (0 - (N-1))                                */
/*                                                                          */
/* Outputs:                                                                 */
/*                                                                          */
/*     byte *bus_number                                                     */
/*         PCI Bus in which this device is found                            */
/*                                                                          */
/*     byte *device_and_function                                            */
/*         Device Number in upper 5 bits, Function Number in lower 3 bits   */
/*                                                                          */
/*     Return Value - Indicates presence of device                         */
/*         SUCCESSFUL - Device found                                        */
/*         NOT_SUCCESSFUL - BIOS error                                      */
/*         DEVICE_NOT_FOUND - Device not found                             */
/*                                                                          */
/******************************************************************************/

int find_pci_class_code(dword class_code,
                        word index,
                        byte *bus_number,
                        byte *device_and_function)
{
    int ret_status;    /* Function Return Status */
    word ax, bx, flags; /* Temporary variables to hold register values */

    /* Load entry registers for PCI BIOS */
    _ECX = class_code;
    _SI = index;
    _AH = PCI_FUNCTION_ID;
    _AL = FIND_PCI_CLASS_CODE;
```

B

```
    /* Call PCI BIOS Int 1Ah interface */
    geninterrupt(0x1a);

    /* Save registers before overwritten by compiler usage of registers */
    ax = _AX;
    bx = _BX;
    flags = _FLAGS;

    /* First check if CARRY FLAG Set, if so, error has occurred */
    if ((flags & CARRY_FLAG) == 0) {

        /* Get Return code from BIOS */
        ret_status = HIGH_BYTE(ax);
        if (ret_status == SUCCESSFUL) {

            /* Assign Bus Number, Device & Function if successful */
            if (bus_number != NULL) {
                *bus_number = HIGH_BYTE(bx);
            }
            if (device_and_function != NULL) {
                *device_and_function = LOW_BYTE(bx);
            }
        }
    }
    else {
        ret_status = NOT_SUCCESSFUL;
    }

    return (ret_status);
}

/*****************************************************************************/
/*                                                                         */
/*   READ_CONFIGURATION_BYTE                                               */
/*                                                                         */
/* Purpose: Reads a byte from the configuration space of a specific device */
/*                                                                         */
/* Inputs:                                                                 */
/*                                                                         */
/*    byte bus_number                                                      */
/*        PCI bus to read configuration data from                         */
/*                                                                         */
/*    byte device_and_function                                            */
/*        Device Number in upper 5 bits, Function Number in lower 3 bits   */
/*                                                                         */
/*    byte register_number                                                */
/*        Register Number of configuration space to read                  */
/*                                                                         */
/* Outputs:                                                                */
/*                                                                         */
/*    byte *byte_read                                                      */
/*        Byte read from Configuration Space                              */
/*                                                                         */
/*    Return Value - Indicates presence of device                         */
/*        SUCCESSFUL - Device found                                        */
/*        NOT_SUCCESSFUL - BIOS error                                      */
/*        BAD_REGISTER_NUMBER - Invalid Register Number                    */
/*                                                                         */
```

```
    /**************************************************************************/

    int read_configuration_byte(byte bus_number,
                                byte device_and_function,
                                byte register_number,
                                byte *byte_read)
    {
        int ret_status; /* Function Return Status */
        dword data;

        /* Call read_configuration_area function with byte data */
        ret_status = read_configuration_area(READ_CONFIG_BYTE,
                                             bus_number,
                                             device_and_function,
                                             register_number,
                                             &data);
        if (ret_status == SUCCESSFUL) {

            /* Extract byte */
            *byte_read = data & 0xff;
        }

        return (ret_status);
    }



    /**************************************************************************/
    /*                                                                        */
    /*   READ_CONFIGURATION_WORD                                              */
    /*                                                                        */
    /* Purpose: Reads a word from the configuration space of a specific device */
    /*                                                                        */
    /* Inputs:                                                                */
    /*                                                                        */
    /*    byte bus_number                                                     */
    /*        PCI bus to read configuration data from                         */
    /*                                                                        */
    /*    byte device_and_function                                           */
    /*        Device Number in upper 5 bits, Function Number in lower 3 bits  */
    /*                                                                        */
    /*    byte register_number                                               */
    /*        Register Number of configuration space to read                  */
    /*                                                                        */
    /* Outputs:                                                               */
    /*                                                                        */
    /*    word *word_read                                                     */
    /*        Word read from Configuration Space                              */
    /*                                                                        */
    /*    Return Value - Indicates presence of device                         */
    /*        SUCCESSFUL - Device found                                       */
    /*        NOT_SUCCESSFUL - BIOS error                                     */
    /*        BAD_REGISTER_NUMBER - Invalid Register Number                   */
    /*                                                                        */
    /**************************************************************************/

    int read_configuration_word(byte bus_number,
                                byte device_and_function,
                                byte register_number,
```

```
                              word *word_read)
{
    int ret_status; /* Function Return Status */
    dword data;

    /* Call read_configuration_area function with word data */
    ret_status = read_configuration_area(READ_CONFIG_WORD,
                                         bus_number,
                                         device_and_function,
                                         register_number,
                                         &data);
    if (ret_status == SUCCESSFUL) {

        /* Extract word */
        *word_read = data & 0xffff;
    }

    return (ret_status);
}

/*************************************************************************/
/*                                                                       */
/*   READ_CONFIGURATION_DWORD                                            */
/*                                                                       */
/* Purpose: Reads a dword from the configuration space of a specific device */
/*                                                                       */
/* Inputs:                                                               */
/*                                                                       */
/*    byte bus_number                                                    */
/*        PCI bus to read configuration data from                        */
/*                                                                       */
/*    byte device_and_function                                           */
/*        Device Number in upper 5 bits, Function Number in lower 3 bits  */
/*                                                                       */
/*    byte register_number                                               */
/*        Register Number of configuration space to read                 */
/*                                                                       */
/* Outputs:                                                              */
/*                                                                       */
/*    dword *dword_read                                                  */
/*        Dword read from Configuration Space                            */
/*                                                                       */
/*    Return Value - Indicates presence of device                        */
/*        SUCCESSFUL - Device found                                      */
/*        NOT_SUCCESSFUL - BIOS error                                    */
/*        BAD_REGISTER_NUMBER - Invalid Register Number                  */
/*                                                                       */
/*************************************************************************/

int read_configuration_dword(byte bus_number,
                             byte device_and_function,
                             byte register_number,
                             dword *dword_read)
{
    int ret_status; /* Function Return Status */
    dword data;
```

```
    /* Call read_configuration_area function with dword data */
    ret_status = read_configuration_area(READ_CONFIG_DWORD,
                                          bus_number,
                                          device_and_function,
                                          register_number,
                                          &data);
    if (ret_status == SUCCESSFUL) {

        /* Extract dword */
        *dword_read = data;
    }

    return (ret_status);
}

/*****************************************************************************/
/*                                                                         */
/*  READ_CONFIGURATION_AREA                                                */
/*                                                                         */
/* Purpose: Reads a byte/word/dword from the configuration space of a      */
/*          specific device                                                */
/*                                                                         */
/* Inputs:                                                                 */
/*                                                                         */
/*    byte bus_number                                                      */
/*        PCI bus to read configuration data from                         */
/*                                                                         */
/*    byte device_and_function                                            */
/*        Device Number in upper 5 bits, Function Number in lower 3 bits   */
/*                                                                         */
/*    byte register_number                                                */
/*        Register Number of configuration space to read                  */
/*                                                                         */
/* Outputs:                                                                */
/*                                                                         */
/*    dword *dword_read                                                    */
/*        Dword read from Configuration Space                             */
/*                                                                         */
/*    Return Value - Indicates presence of device                         */
/*        SUCCESSFUL - Device found                                        */
/*        NOT_SUCCESSFUL - BIOS error                                      */
/*        BAD_REGISTER_NUMBER - Invalid Register Number                   */
/*                                                                         */
/*****************************************************************************/

static int read_configuration_area(byte function,
                                    byte bus_number,
                                    byte device_and_function,
                                    byte register_number,
                                    dword *data)
{
    int ret_status; /* Function Return Status */
    word ax, flags; /* Temporary variables to hold register values */
    dword ecx;      /* Temporary variable to hold ECX register value */

    /* Load entry registers for PCI BIOS */
    _BH = bus_number;
    _BL = device_and_function;
```

B

```
    _DI = register_number;
    _AH = PCI_FUNCTION_ID;
    _AL = function;

    /* Call PCI BIOS Int 1Ah interface */
    geninterrupt(0x1a);

    /* Save registers before overwritten by compiler usage of registers */
    ecx = _ECX;
    ax = _AX;
    flags = _FLAGS;

    /* First check if CARRY FLAG Set, if so, error has occurred */
    if ((flags & CARRY_FLAG) == 0) {

        /* Get Return code from BIOS */
        ret_status = HIGH_BYTE(ax);

        /* If successful, return data */
        if (ret_status == SUCCESSFUL) {
            *data = ecx;
        }
    }
    else {
        ret_status = NOT_SUCCESSFUL;
    }


    return (ret_status);
}

/**************************************************************************/
/*                                                                        */
/*  WRITE_CONFIGURATION_BYTE                                              */
/*                                                                        */
/* Purpose: Writes a byte to the configuration space of a specific device */
/*                                                                        */
/* Inputs:                                                                */
/*                                                                        */
/*    byte bus_number                                                     */
/*        PCI bus to write configuration data to                          */
/*                                                                        */
/*    byte device_and_function                                            */
/*        Device Number in upper 5 bits, Function Number in lower 3 bits   */
/*                                                                        */
/*    byte register_number                                                */
/*        Register Number of configuration space to write                 */
/*                                                                        */
/*    byte byte_to_write                                                  */
/*        Byte to write to Configuration Space                            */
/*                                                                        */
/* Outputs:                                                               */
/*                                                                        */
/*    Return Value - Indicates presence of device                        */
/*        SUCCESSFUL - Device found                                       */
/*        NOT_SUCCESSFUL - BIOS error                                     */
/*        BAD_REGISTER_NUMBER - Invalid Register Number                   */
/*                                                                        */
```

```
/************************************************************************/

int write_configuration_byte(byte bus_number,
                             byte device_and_function,
                             byte register_number,
                             byte byte_to_write)
{
    int ret_status; /* Function Return Status */

    /* Call write_configuration_area function with byte data */
    ret_status = write_configuration_area(WRITE_CONFIG_BYTE,
                                          bus_number,
                                          device_and_function,
                                          register_number,
                                          byte_to_write);
    return (ret_status);
}


/************************************************************************/
/*                                                                    */
/*   WRITE_CONFIGURATION_WORD                                         */
/*                                                                    */
/* Purpose: Writes a word to the configuration space of a specific device */
/*                                                                    */
/* Inputs:                                                            */
/*                                                                    */
/*    byte bus_number                                                 */
/*        PCI bus to read configuration data from                    */
/*                                                                    */
/*    byte device_and_function                                       */
/*        Device Number in upper 5 bits, Function Number in lower 3 bits */
/*                                                                    */
/*    byte register_number                                           */
/*        Register Number of configuration space to read             */
/*                                                                    */
/*    word word_to_write                                             */
/*        Word to write to Configuration Space                       */
/*                                                                    */
/* Outputs:                                                           */
/*                                                                    */
/*    Return Value - Indicates presence of device                    */
/*        SUCCESSFUL - Device found                                  */
/*        NOT_SUCCESSFUL - BIOS error                                */
/*        BAD_REGISTER_NUMBER - Invalid Register Number              */
/*                                                                    */
/************************************************************************/

int write_configuration_word(byte bus_number,
                             byte device_and_function,
                             byte register_number,
                             word word_to_write)
{
    int ret_status; /* Function Return Status */

    /* Call read_configuration_area function with word data */
    ret_status = write_configuration_area(WRITE_CONFIG_WORD,
                                          bus_number,
```

B

```
                                        device_and_function,
                                        register_number,
                                        word_to_write);
      return (ret_status);
}


/*************************************************************************/
/*                                                                     */
/*   WRITE_CONFIGURATION_DWORD                                         */
/*                                                                     */
/* Purpose: Writes a dword from the configuration space of a specific device */
/*                                                                     */
/* Inputs:                                                             */
/*                                                                     */
/*    byte bus_number                                                  */
/*        PCI bus to write configuration data from                    */
/*                                                                     */
/*    byte device_and_function                                        */
/*        Device Number in upper 5 bits, Function Number in lower 3 bits */
/*                                                                     */
/*    byte register_number                                            */
/*        Register Number of configuration space to read              */
/*                                                                     */
/*    dword dword_to_write                                            */
/*        Dword to write to Configuration Space                       */
/*                                                                     */
/* Outputs:                                                            */
/*                                                                     */
/*    Return Value - Indicates presence of device                     */
/*        SUCCESSFUL - Device found                                   */
/*        NOT_SUCCESSFUL - BIOS error                                 */
/*        BAD_REGISTER_NUMBER - Invalid Register Number               */
/*                                                                     */
/*************************************************************************/

int write_configuration_dword(byte bus_number,
                              byte device_and_function,
                              byte register_number,
                              dword dword_to_write)
{
    int ret_status; /* Function Return Status */

    /* Call write_configuration_area function with dword data */
    ret_status = write_configuration_area(WRITE_CONFIG_DWORD,
                                          bus_number,
                                          device_and_function,
                                          register_number,
                                          dword_to_write);
    return (ret_status);
}


/*************************************************************************/
/*                                                                     */
/*   WRITE_CONFIGURATION_AREA                                          */
/*                                                                     */
/* Purpose: Writes a byte/word/dword to the configuration space of a   */
/*          specific device                                           */
```

```
/*                                                                      */
/* Inputs:                                                              */
/*                                                                      */
/*    byte bus_number                                                   */
/*        PCI bus to read configuration data from                      */
/*                                                                      */
/*    byte device_and_function                                         */
/*        Device Number in upper 5 bits, Function Number in lower 3 bits */
/*                                                                      */
/*    byte register_number                                             */
/*        Register Number of configuration space to read               */
/*                                                                      */
/*    dword value                                                       */
/*        Value to write to Configuration Space                        */
/*                                                                      */
/* Outputs:                                                             */
/*                                                                      */
/*    Return Value - Indicates presence of device                      */
/*        SUCCESSFUL - Device found                                     */
/*        NOT_SUCCESSFUL - BIOS error                                   */
/*        BAD_REGISTER_NUMBER - Invalid Register Number                 */
/*                                                                      */
/*************************************************************************/

static int write_configuration_area(byte function,
                                    byte bus_number,
                                    byte device_and_function,
                                    byte register_number,
                                    dword value)
{
    int ret_status; /* Function Return Status */
    word ax, flags; /* Temporary variables to hold register values */

    /* Load entry registers for PCI BIOS */
    _BH = bus_number;
    _BL = device_and_function;
    _ECX = value;
    _DI = register_number;
    _AH = PCI_FUNCTION_ID;
    _AL = function;

    /* Call PCI BIOS Int 1Ah interface */
    geninterrupt(0x1a);

    /* Save registers before overwritten by compiler usage of registers */
    ax = _AX;
    flags = _FLAGS;

    /* First check if CARRY FLAG Set, if so, error has occurred */
    if ((flags & CARRY_FLAG) == 0) {

        /* Get Return code from BIOS */
        ret_status = HIGH_BYTE(ax);
    }
    else {
        ret_status = NOT_SUCCESSFUL;
    }
```

B

```
    return (ret_status);
}


/***************************************************************************/
/*                                                                       */
/*  OUTPD                                                                 */
/*                                                                       */
/* Purpose: Outputs a DWORD to a hardware port                           */
/*                                                                       */
/* Inputs:                                                               */
/*                                                                       */
/*    word port                                                          */
/*        hardware port to write DWORD to                                */
/*                                                                       */
/*    dword value                                                        */
/*        value to be written to port                                    */
/*                                                                       */
/* Outputs:                                                              */
/*                                                                       */
/*    None                                                               */
/*                                                                       */
/***************************************************************************/

void outpd(word port, dword value)
{
    _DX = port;
    _EAX = value;

    /* Since asm cannot generate OUT  DX, EAX must force in */
    __emit__(0x66, 0xEF);
}


/***************************************************************************/
/*                                                                       */
/*  INPD                                                                  */
/*                                                                       */
/* Purpose: Inputs a DWORD from a hardware port                          */
/*                                                                       */
/* Inputs:                                                               */
/*                                                                       */
/*    word port                                                          */
/*        hardware port to write DWORD to                                */
/*                                                                       */
/* Outputs:                                                              */
/*                                                                       */
/*    dword value                                                        */
/*        value read from the port                                       */
/*                                                                       */
/***************************************************************************/

dword inpd(word port)
{
    /* Set DX register to port number to be input from */
    _DX = port;

    /* Since asm cannot generate IN  EAX, DX, must force in */
    __emit__(0x66, 0xED);
```

```
        return(_EAX);
    }


    /***********************************************************************/
    /*                                                                     */
    /*   INSB                                                              */
    /*                                                                     */
    /* Purpose: Inputs a string of BYTEs from a hardware port              */
    /*                                                                     */
    /* Inputs:                                                             */
    /*                                                                     */
    /*     word port                                                       */
    /*         hardware port to read from                                  */
    /*                                                                     */
    /*     void *buf                                                       */
    /*         Buffer to read data into                                    */
    /*                                                                     */
    /*     int count                                                       */
    /*         Number of BYTEs to read                                     */
    /*                                                                     */
    /* Outputs:                                                            */
    /*                                                                     */
    /*     None                                                            */
    /*                                                                     */
    /***********************************************************************/

    void insb(word port, void *buf, int count)
    {
        _ES = FP_SEG(buf);    /* Segment of buf */
        _DI = FP_OFF(buf);    /* Offset of buf  */
        _CX = count;          /* Number to read */
        _DX = port;           /* Port           */
        asm   REP INSB;
    }


    /***********************************************************************/
    /*                                                                     */
    /*   INSW                                                              */
    /*                                                                     */
    /* Purpose: Inputs a string of WORDs from a hardware port              */
    /*                                                                     */
    /* Inputs:                                                             */
    /*                                                                     */
    /*     word port                                                       */
    /*         hardware port to read from                                  */
    /*                                                                     */
    /*     void *buf                                                       */
    /*         Buffer to read data into                                    */
    /*                                                                     */
    /*     int count                                                       */
    /*         Number of WORDs to read                                     */
    /*                                                                     */
    /* Outputs:                                                            */
    /*                                                                     */
    /*     None                                                            */
    /*                                                                     */
    /***********************************************************************/
```

B

```
void insw(word port, void *buf, int count)
{
    _ES = FP_SEG(buf);    /* Segment of buf */
    _DI = FP_OFF(buf);    /* Offset of buf  */
    _CX = count;          /* Number to read */
    _DX = port;           /* Port           */
    asm   REP INSW;
}


/***************************************************************************/
/*                                                                       */
/*   INSD                                                                 */
/*                                                                       */
/* Purpose: Inputs a string of DWORDs from a hardware port               */
/*                                                                       */
/* Inputs:                                                               */
/*                                                                       */
/*    word port                                                          */
/*        hardware port to read from                                     */
/*                                                                       */
/*    void *buf                                                          */
/*        Buffer to read data into                                       */
/*                                                                       */
/*    int count                                                          */
/*        Number of DWORDs to read                                       */
/*                                                                       */
/* Outputs:                                                              */
/*                                                                       */
/*    None                                                               */
/*                                                                       */
/***************************************************************************/

void insd(word port, void *buf, int count)
{
    _ES = FP_SEG(buf);    /* Segment of buf */
    _DI = FP_OFF(buf);    /* Offset of buf  */
    _CX = count;          /* Number to read */
    _DX = port;           /* Port           */
    __emit__(0xf3, 0x66, 0x6D); /* asm  REP INSD */
}


/***************************************************************************/
/*                                                                       */
/*   OUTSB                                                                */
/*                                                                       */
/* Purpose: Outputs a string of BYTEs to a hardware port                 */
/*                                                                       */
/* Inputs:                                                               */
/*                                                                       */
/*    word port                                                          */
/*        hardware port to write to                                      */
/*                                                                       */
/*    void *buf                                                          */
/*        Buffer to write data from                                      */
/*                                                                       */
/*    int count                                                          */
/*        Number of BYTEs to write                                       */
/*                                                                       */
```

```
/* Outputs:                                                               */
/*                                                                        */
/*    None                                                                */
/*                                                                        */
/**************************************************************************/

void outsb(word port, void *buf, int count)
{
   _ES = FP_SEG(buf);
   _SI = FP_OFF(buf);    /* Offset of buf  */
   _CX = count;          /* Number to read */
   _DX = port;           /* Port           */
   asm   REP OUTSB;
}


/**************************************************************************/
/*                                                                        */
/*   OUTSW                                                                 */
/*                                                                        */
/* Purpose: Outputs a string of WORDs to a hardware port                  */
/*                                                                        */
/* Inputs:                                                                */
/*                                                                        */
/*    word port                                                           */
/*       hardware port to write to                                        */
/*                                                                        */
/*    void *buf                                                           */
/*       Buffer to write data from                                        */
/*                                                                        */
/*    int count                                                           */
/*       Number of WORDs to write                                         */
/*                                                                        */
/* Outputs:                                                               */
/*                                                                        */
/*    None                                                                */
/*                                                                        */
/**************************************************************************/

void outsw(word port, void *buf, int count)
{
   _ES = FP_SEG(buf);
   _SI = FP_OFF(buf);    /* Offset of buf  */
   _CX = count;          /* Number to read */
   _DX = port;           /* Port           */
   asm   REP OUTSW;
}
```

B

```
/****************************************************************************/
/*                                                                          */
/*   OUTSD                                                                   */
/*                                                                          */
/* Purpose: Outputs a string of DWORDS to a hardware port                   */
/*                                                                          */
/* Inputs:                                                                  */
/*                                                                          */
/*    word port                                                             */
/*        hardware port to write to                                         */
/*                                                                          */
/*    void *buf                                                             */
/*        Buffer to write data from                                         */
/*                                                                          */
/*    int count                                                             */
/*        Number of DWORDs to write                                         */
/*                                                                          */
/* Outputs:                                                                 */
/*                                                                          */
/*    None                                                                  */
/*                                                                          */
/****************************************************************************/

void outsd(word port, void *buf, int count)
{
   _ES = FP_SEG(buf);
   _SI = FP_OFF(buf);    /* Offset of buf  */
   _CX = count;          /* Number to read */
   _DX = port;           /* Port           */
   __emit__(0xf3, 0x66, 0x6F); /* asm   REP OUTSD; */
}
```

INDEX

C

C

5V Supply-TTL I/O Clock Driver Products

P/N	Output Frequency with Respect to Input Frequency				Special Features	Package
	Total Outputs	Number of Outputs ÷ 1	Number of Outputs ÷ 2	Number of Outputs ÷ 2 or 4		
SC3506	20	10	10	N/A	—	52 PQFP

See Precision Clocking Products data book or Network Interface Products data book.

5V Supply-LVTTL I/O Clock Driver Products

P/N	Output Frequency with Respect to Input Frequency			Special Features	Package
	Total Outputs	Number of Outputs ÷ 1	Number of Outputs ÷ 2		
SC3306	20	10	10		52 PQFP
SC3308	20	20	N/A		52 PQFP
SC3318	10	10	N/A		28 SOIC
SC3368	14	6	8	Selectable single or dual clock input.	28 SOIC

See Precision Clocking Products data book or Network Interface Products data book.

3.3V Supply-LVTTL I/O Clock Driver Products

P/N	Output Frequency with Respect to Input Frequency			Special Features	Package
	Total Outputs	Number of Outputs ÷ 1	Number of Outputs ÷ 2		
S3LV308	20	20	N/A		52 PQFP

See Precision Clocking Products data book or Network Interface Products data book.

D

AMCC

5V/3.3V Clock Generator and Synthesizer Products

		Output Frequency Group with Respect to Input Frequency					
P/N	Description	Input Reference	Number	Type	Max. Freq.	Min. Delay Adjust Increment	Number of Selectable Output Relationships
S4402	Multiphase Clk Generator	TTL	6	TTL	80	3.2 ns	21
S4403	Multiphase Clk Generator	TTL	10	TTL	80	3.2 ns	21
S4405	Multiphase Clk Generator with PECL I/O	PECL/TTL	6 1	TTL PECL	80 160	3.2 ns —	21 —
S4406	Clock Generator with Delay Adj. & Invert	TTL	12	TTL	66	4 ns @ 66 MHz	7 x 4 Banks of 3 Outputs
S4503	Clock Synthesizer	XTAL	2 1	TTL PECL	80 300	N/A	Multiply 2-32 Divide 2-16
S4505 S4506 S4507	RAMBUS™ Compatible Clock Generator	XTAL	2	RAMBUS Compatible	300	N/A	1

See Precision Clocking Products data book or Network Interface Products data book.

ATM Products

Product	Function	Operating Speed	Data Path	Package	Power Supply
S3011	SONET/ATM/E-4 Tx	139/155 Mbit/s	8:1 bit	80 TEP	+5.0V
S3012	SONET/ATM/E-4 Rx	139/155 Mbit/s	1:8 bit	80 TEP	+5.0V
S3020	ATM Tx	622 Mbit/s	8:1 bit	52 TEP	+5.0V
S3021	ATM Rx	622 Mbit/s	1:8 bit	52 TEP	+5.0V

See Network Interface Products data book.

100VG AnyLAN Interface Products

Product	Function	Operating Speed	Data Path	Package	Power Supply
S2100	100VG AnyLAN Transceiver	120 Mbit/s	4:1/1:4 bit	52 PQFP	+5.0V

See Network Interface Products data book.

Fibre Channel Products

Product	Function	Operating Speed	Data Path	Package	Power Supply
S2036	Open Fiber Control	266/531/1062 Mbit/s	N/A	28 SOIC	+5V
S2042	Fibre Channel Transmitter	266/531/1062 Mbit/s	10:1/20:1 bit	52 TQFP	+3.3V
S2043	Fibre Channel Receiver	266/531/1062 Mbit/s	1:10/1:20 bit	52 TQFP	+3.3V
S2044	GLM Compliant Fibre Channel Transmitter	266/531/1062 Mbit/s	10:1/20:1 bit	52 TQFP	+3.3V
S2045	GLM Compliant Fibre Channel Receiver	266/531/1062 Mbit/s	1:10/1:20 bit	52 TQFP	+3.3V
S2046	Gigabit Ethernet Transmitter	1250 Mbit/s	10:1/20:1 bit	52 TQFP	+3.3V
S2047	Gigabit Ethernet Receiver	1250 Mbit/s	1:10/1:20 bit	52 TQFP	+3.3V
S2052	Gigabit Fibre Channel Transceiver	266/531/1062/ 1250 Mbit/s	10:1/1:10 bit	52 PQFP	+3.3V

See Network Interface Products data book.

D

HIPPI Products

Product	Function	Operating Speed	Data Path	Package	Power Supply
S2020	HIPPI Source	800 Mbit/s	32 bit	225 PGA/ 208 TEP	-5.2/+5V
S2021	HIPPI Destination	800 Mbit/s	32 bit	225 PGA/ 208 TEP	-5.2/+5V

See Network Interface Products data book.

PCI Bus Products

Product	Function	Description
S5933	PCI Controller	General Purpose PCI Interface

See S5933 PCI Controller data book.

SONET/SDH Products

Product	Function	Operating Speed	Data Path	Package	Power Supply
S3005	SONET/E-4 Tx	139/155/622 Mbit/s	8:1 bit	68 LDCC 80 TEP	-4.5/5.0V
S3006	SONET/E-4 Rx	139/155/622 Mbit/s	1:8 bit	68 LDCC 80 TEP	-4.5/5.0V
S3014	Clock Recovery	155/622 Mbit/s	1 bit	44 PLCC	-5.2/+5.0V
S3015	E4/OC-3/STM-1 Interface Tx	139/155 Mbit/s	1 bit	52 TEP	+5V
S3016	E4/OC-3/STM-1 Interface Rx	139/155 Mbit/s	1 bit	52 TEP	+5V
S3017	SONET/SDH Tx	622 Mbit/s	8:1 bit	52 TEP	+5V
S3018	SONET/SDH Rx	622 Mbit/s	8:1 bit	52 TEP	+5V
S3025/26	Clock Recovery	155/622 Mbit/s	1 bit	16 PLCC	+5V
S3028	SONET/SDH Tx	155/622 Mbit/s	8:1/1:8 bit	64 PQFP	+5V

See Network Interface Products data book.

Crosspoint Switch Products

Product	Function	Operating Speed	Data Path	Package	Power Supply
S2016	Crosspoint Switch	1.5 Gbit/s	16 x16	120 TEP	+5V
S2024	Crosspoint Switch	600/800 Mbit/s	32x32	196 LDCC	-5.2/+5V
S2025	Crosspoint Switch	1.5 Gbit/s	32x32	196 LDCC	+5V

See Network Interface Products data book.

D

AMCC

ASIC Products

Part Number	Technology	Equivalent Gates (Full Adder Method)	Number of I/O	Structured Array Blocks
Q20004	1 Micron Bipolar	671	30	None
Q20010	1 Micron Bipolar	1469	66	None
Q20025	1 Micron Bipolar	4032	102	None
Q20045	1 Micron Bipolar	6782	130	None
Q20080	1 Micron Bipolar	11242	164	None
Q20120	1 Micron Bipolar	18777	200	None
Q20P010	1 Micron Bipolar	928	36	1 GHz PLL
Q20P025	1 Micron Bipolar	3120	51	1 GHz PLL
Q20M100	1 Micron Bipolar	13475	195	4K RAM

See Network Interface Products data book.

AMCC SALES OFFICES

Europe/Israel
Applications & Sales

Weltenburger Str. 70
81677 Munich, Germany
Tel: 011/49-89-92404-136
Fax: 011/49-8121-3180

ASIA Pacific & ROW

Applied Micro Circuits Corp.
Corporate Headquarters
6195 Lusk Blvd.
San Diego, CA 92121
Tel: 619/450-9333
Fax: 619/450-9885

Northeast

Applied Micro Circuits Corp.
25 Burlington Mall Road
Suite 300
Burlington, MA 01803
Tel: 617/270-0674
Fax: 617/221-5853

South Atlantic

Applied Micro Circuits Corp.
Atrium Executive Center
80 Orville Drive
Bohemia, NY 11716
Tel: 516/244-1460
Fax: 516/244-1464

Mid-America, Arizona &
New Mexico

Applied Micro Circuits Corp.
840 E. Central Parkway
Suite 120
Plano, TX 75074
Tel: 972/423-7989
Fax: 972/424-6617

Northwest

Applied Micro Circuits Corp.
950 S. Bascom Avenue
Suite 1113
San Jose, CA 95128
Tel: 408/289-1194
Fax: 408/289-1527

Southwest

Applied Micro Circuits Corp.
501 N. El Camino Real
Suite 217
San Clemente, CA 92680
Tel: 714/366-4105
Fax: 714/366-4126

NORTH AMERICAN SALES REPRESENTATIVES

ALABAMA

Glen White Associates
3322 S. Memorial Pkwy.
Suite 40
Huntsville, AL 35801
Tel: 205/882-6751
Fax: 205/880-2750

ARIZONA

Quatra Associates, Inc.
10235 S. 51st Street
Suite 160
Phoenix, AZ 85044
Tel: 602/753-5544
Fax: 602/753-0640

CALIFORNIA

Littlefield & Smith Assoc.
11230 Sorrento Valley Road
Suite 115
San Diego, CA 92121
Tel: 619/455-0055
Fax: 619/455-1218

Elrepco
777 Cuesta Drive
Suite 200
Mt. View, CA 94040
Tel: 415/962-0660
Fax: 415/965-1644

DynaRep, Inc.
3002 Dow Avenue, Suite 226
Tustin, CA 92780
Tel: 714/573-1223
Fax: 714/573-0778

DynaRep, Inc.
28720 Roadside Drive
Suite 224
Agoura Hills, CA 91301
Tel: 818/735-5410
Fax: 818/735-5409

CANADA

(Vancouver, B.C.)
Components West
4020 148th Ave., N.E.
Suite C
Redmond, WA 98052
Tel: 206/885-5880
Fax: 206/882-0642

Electronic Sales
   Professionals, Inc. (ESP)
215 Stafford Road West
Unit 104
Nepean, Ontario
Canada K2H 9C1
Tel: 613/828-6881
Fax: 613/828-5725

Electronic Sales
   Professionals, Inc. (ESP)
137 Main Street No.
Suite 204
Markham, Ontario
Canada L3P 1Y2
Tel: 905/294-3520
Fax: 905/294-3806

Electronic Sales
   Professionals, Inc. (ESP)
10690 Peloquin Street
Suite 210
Montreal, Quebec
Canada H2C 2K3
Tel: 514/388-6596
Fax: 514/388-8402

COLORADO

Luscombe Engineering Co.
1500 Kansas Ave., Suite 1B
Longmont, CO 80501
Tel: 303/772-3342
Fax: 303/772-8783

Luscombe Engineering Co.
6239 Northwoods Glenn Drive
Parker, CO 80134
Tel: 303/841-7478
Fax: 303/841-7328

CONNECTICUT

Dynamic Technologies
6 Way Road
Middlefield, CT 06455
Tel: 860/349-7055
Fax: 860/349-7056

DELAWARE

Delta Technical Sales, Inc.
122 North York Road
Suite 9
Hatboro, PA 19040
Tel: 215/957-0600
Fax: 215/957-0920

FLORIDA

Photon Sales, Inc.
1600 Sarno Road
Suite 21
Melbourne, FL 32935
Tel: 407/259-8999
Fax: 407/259-1323

Photon Sales, Inc.
715 Florida Street
P.O. Box 560776
Orlando, FL 32806-0776
Tel: 407/841-7423
Fax: 407/896-6197

Photon Sales, Inc.
815 NE 1st Court
Delray Beach, FL 33483
Tel: 407/278-6513
Fax: 407/278-6549

Photon Sales, Inc.
2510 Cypress Bend Drive
Clearwater, FL 34621
Tel: 407/841-7423
Fax: 407/896-6197

GEORGIA

Glen White Associates
2444 Highway 120
Suite 101
Duluth, GA 30155
Tel: 770/418-1500
Fax: 770/418-1660

E

AMCC

IDAHO

Components West
8323 S.W. Cirrus Drive
Beaverton, OR 97008
Tel: 503/520-1900
Fax: 503/520-1906

ILLINOIS

Phase II Marketing, Inc.
2260 Hicks Road
Suite 410
Rolling Meadows, IL 60008
Tel: 847/577-9401
Fax: 847/577-9491

(Southern Illinois)
SPS Associates, Ltd.
1250 N. Winchester
Suite C
Olathe, KS 66061
Tel: 913/764-4460
Fax: 913/764-6161

INDIANA

Century Technical Sales, Inc.
3520 W. 86th Street
Suite 261
Indianapolis, IN 46268
Tel: 317/876-0101
Fax: 317/875-5566

Century Technical Sales, Inc.
36 Country Forest Drive
Ft. Wayne, IN 46818
Tel: 219/489-6058
Fax: 219/489-8823

IOWA

Customer 1st
2950 Metro Drive
Suite 110
Bloomington, MN 55425
Tel: 612/851-7909
Fax: 612/851-7907

KANSAS

SPS Associates, Ltd.
1250 N. Winchester
Suite C
Olathe, KS 66061
Tel: 913/764-4460
Fax: 913/764-6161

KENTUCKY

Century Technical Sales, Inc.
8977 Columbia Road
Suite G
Loveland, OH 45140
Tel: 513/677-5088
Fax: 513/677-1775

MAINE

Comp Rep Associates
100 Everett Street
Westwood, MA 02090
Tel: 617/329-3454
Fax: 617/329-6395

MARYLAND

DGR, Incorporated
22 West Padonia Road
Suite B 317
Timonium, MD 21093
Tel: 410/453-0969
Fax: 410/453-6199

MASSACHUSETTS

Comp Rep Associates
100 Everett Street
Westwood, MA 02090
Tel: 617/329-3454
Fax: 617/329-6395

MINNESOTA

Customer 1st
2950 Metro Drive
Suite 110
Bloomington, MN 55425
Tel: 612/851-7909
Fax: 612/851-7907

MISSISSIPPI

Glen White Associates
3322 S. Memorial Pkwy.
Suite 40
Huntsville, AL 35801
Tel: 205/882-6751
Fax: 205/880-2750

MISSOURI

SPS Associates, Ltd.
3301 Rider Trail South
Suite 110
Earth City, MO 63045
Tel: 314/291-0520
Fax: 314/291-7138

MONTANA

Components West
4020 148th Ave., N.E.
Suite C
Redmond, WA 98052
Tel: 206/885-5880
Fax: 206/882-0642

NEBRASKA

SPS Associates, Ltd.
1250 N. Winchester
Suite C
Olathe, KS 66061
Tel: 913/764-4460
Fax: 913/764-6161

NEVADA

Elrepco
777 Cuesta Drive
Suite 200
Mt. View, CA 94040
Tel: 415/962-0660
Fax: 415/965-1644

NEW HAMPSHIRE

Comp Rep Associates
100 Everett Street
Westwood, MA 02090
Tel: 617/329-3454
Fax: 617/329-6395

NEW JERSEY

(Southern New Jersey)
Delta Technical Sales, Inc.
122 North York Road
Suite 9
Hatboro, PA 19040
Tel: 215/957-0600
Fax: 215/957-0920

(Northern New Jersey)
ERA, Inc.
354 Veterans Memorial Hwy.
Commack, NY 11725
Tel: 516/543-0510
Fax: 516/543-0758

NEW MEXICO

Quatra Associates, Inc.
600 Autumnwood Place S.E.
Albuquerque, NM 87123-4347
Tel: 505/296-6781
Fax: 505/292-2092

NEW YORK

Quality Components
116 Fayette Street
Manlius, NY 13104
Tel: 315/682-8885
Fax: 315/682-2277

ERA, Inc.
354 Veterans Memorial Hwy.
Commack, NY 11725
Tel: 516/543-0510
Fax: 516/543-0758

NORTH CAROLINA

Glen White Associates
6070J Six Forks Road
Raleigh, NC 27609
Tel: 919/848-1931
Fax: 919/847-3294

NORTH DAKOTA

Customer 1st
2950 Metro Drive
Suite 110
Bloomington, MN 55425
Tel: 612/851-7909
Fax: 612/851-7907

OHIO

Century Technical Sales, Inc.
8977 Columbia Road
Suite G
Loveland, OH 45140
Tel: 513/677-5088
Fax: 513/677-1775

Century Technical Sales, Inc.
24610 Detroit Road #170
Westlake, OH 44145
Tel: 216/899-0071
Fax: 216/899-1072

Century Technical Sales, Inc.
6610 Busch Blvd. #250
Columbus, OH 43229
Tel: 614/433-7500
Fax: 614/433-9085

OKLAHOMA

Logic 1 Sales, Inc.
200 East Spring Valley
Suite A
Richardson, TX 75081
Tel: 214/234-0765
Fax: 214/669-3042

OREGON

Components West
8323 S.W. Cirrus Drive
Beaverton, OR 97008
Tel: 503/520-1900
Fax: 503/520-1906

PENNSYLVANIA

Century Technical Sales, Inc.
130 Spang Road
Baden, PA 15005
Tel: 412/934-2326
Fax: 412/934-3031

(Eastern Pennsylvania)
Delta Technical Sales, Inc.
122 North York Road
Suite 9
Hatboro, PA 19040
Tel: 215/957-0600
Fax: 215/957-0920

Applied Micro Circuits Corporation
6195 Lusk Blvd., San Diego, CA 92121 • (619) 450-9333

RHODE ISLAND

Comp Rep Associates
100 Everett Street
Westwood, MA 02090
Tel: 617/329-3454
Fax: 617/329-6395

SOUTH CAROLINA

Glen White Associates
13420 West Reese Blvd.
Huntersville, SC 28078
Tel: 704/875-3777
Fax: 704/875-3843

Glen White Associates
6070J Six Forks Road
Raleigh, NC 27609
Tel: 919/848-1931
Fax: 919/847-3294

SOUTH DAKOTA

Customer 1st
2950 Metro Drive
Suite 110
Bloomington, MN 55425
Tel: 612/851-7909
Fax: 612/851-7907

TENNESSEE

Glen White Associates
3322 S. Memorial Pkwy.
Suite 40
Huntsville, AL 35801
Tel: 205/882-6751
Fax: 205/880-2750

TEXAS

Logic 1 Sales, Inc.
200 East Spring Valley
Suite A
Richardson, TX 75081
Tel: 214/234-0765
Fax: 214/669-3042

Logic 1 Sales, Inc.
9111 Jollyville Road
Suite 112
Austin, TX 78759-7433
Tel: 512/ 345-2952
Fax: 512/346-5309

Logic 1 Sales, Inc.
4606 FM 1960 West
Suite 320
Houston, TX 77069
Tel: 713/444-7594
Fax: 713/444-8236

UTAH

First Source
8341 S. 700 East
Sandy, UT 84070
Tel: 801/561-1999
Fax: 801/561-4525

VERMONT

Comp Rep Associates
100 Everett Street
Westwood, MA 02090
Tel: 617/329-3454
Fax: 617/329-6395

VIRGINIA

DGR, Incorporated
22 West Padonia Road
Suite B 317
Timonium, MD 21093
Tel: 410/453-0969
Fax: 410/453-6199

WASHINGTON

Components West
4020 148th Ave., N.E.
Suite C
Redmond, WA 98052
Tel: 206/885-5880
Fax: 206/882-0642

WASHINGTON, D.C.

DGR, Incorporated
22 West Padonia Road
Suite B 317
Timonium, MD 21093
Tel: 410/453-0969
Fax: 410/453-6199

WISCONSIN

Phase II Marketing, Inc.
205 Bishop's Way
Suite 220
Brookfield, WI 53005
Tel: 414/771-9986
Fax: 414/771-9935

(Western Wisconsin)
Customer 1st
2950 Metro Drive
Suite 110
Bloomington, MN 55425
Tel: 612/851-7909
Fax: 612/851-7907

NORTH AMERICAN DISTRIBUTOR

Insight Electronics
1-800-677-6011 ext. 94
http://www.ikn.com

E

AMCC

INTERNATIONAL SALES OFFICES

UNITED KINGDOM

AMEGA Electronics, LTD.
Loddon Business Centre
Roentgen Road,
Daneshill East
Basingstoke
Hants RG24 8NG
United Kingdom
Tel: 44-1256-305330
Fax: 44-1256-305335

ITALY

ACSIS s.r.l.
Via Alberto Mario, 26
20149 Milano, Italy
Tel: 39-248022522
Fax: 39-248012289

ESCO Italiana SPA
Viale Fratelli Casiraghi, 355
Sesto San Giovanni
20099 Milano, Italy
Tel: 39-2-2409241
Fax: 39-2-2409255

FRANCE

A2M
Siege Social
5 rue Carle Vernet
92315 Sevres Cedex
France
Tel: 33-1-46-237903
Fax: 33-1-46-237923

FINLAND

Yleiselektroniikka Oy
Luomannotko 6
FIN-02201 ESPOO
Finland
Tel: 358-9-4526-21
Fax: 358-9-4526-2202

GERMANY

Tekelec Airtronic GmbH
Kapuzinerstrasse 9
80337 Munich
Germany
Tel: 49-89-51640
Fax: 49-89-535129

NORWAY

Bit Elektronikk A/S
Smedvingen 4
P.O. Box 194
1360 Nesbru
Norway
Tel: 47-66-98-13-70
Fax: 47-66-98-13-71

SWITZERLAND/ AUSTRIA

Abalec AG
Grabenstrasse 9
CH-8952 Schlieren
Zurich, Switzerland
Tel: 41-17-300-455
Fax: 41-17-309-801

DENMARK

Dan-Contact
Smakkegaardsvej 145
DK-2820 Gentofte
Denmark
Tel: 45-39-683633
Fax: 45-39-683362

NETHERLANDS AND BELGIUM

Tekelec Airtronic B.V.
Ypsilon House
Engelandlaan 310
2711 DZ Zoetermeer
Postbus 7140
Netherlands
Tel: 31-79-3461430
Fax: 31-79-3417504

SWEDEN

DipCom Electronics
Torshamnsgatan 35
Box 1230
S-16428 KISTA
Sweden
Tel: 46-8-7522480
Fax: 46-8-7513649

AUSTRALIA

ICD
Melbourne
Unit 2, 14 Melrich Road
Bayswater, Victoria 3153
Australia
Tel: 61-3-9761-3455
Fax: 61-3-9761-3415

NEW ZEALAND

ICD
Auckland
Unit 7, 110 Mays Road
Penrose, Auckland
New Zealand
Tel: 64-9-636-5984
Fax: 64-9-636-5985

ISRAEL

Eldis Technologies
36 Kehilat Zion Street
Herzlia, 46382
Israel
Tel: 972-9-562-666
Fax: 972-9-562-642

SPAIN

Diode Electronica
C/Orense 34
28020 Madrid, Spain
Tel: 34-1-555-36-86
Fax: 34-1-555-71-59

CHINA

Twin-Star Trading Co., Ltd.
Room B, 26/F, Lever Centre
69-71 King Yip Street
Kwun Tong, Kowloon
Hong Kong
Tel: 852-2341-4282
Fax: 852-2763-7717

JAPAN

Teksel Co., LTD.
TBC
Higashi 2-27-10
Shibuya-ku, Tokyo 150
JAPAN
Tel: 81-3-5467-9000
Fax: 81-3-5467-9346

KOREA

Buksung Industrial Co., LTD.
7F Saehan Bldg.
1653-6 Shinlimdong
Kwanak-Ku, Seoul,
Korea
Tel: 82-2-866-1360
Fax: 82-2-862-1273

SINGAPORE

Gates Engineering Pte, LTD.
1123 Serangoon Road
#03-01 UMW Building
Singapore 328207
Tel: 65-299-9937
Fax: 65-299-7636

TAIWAN

Johnson Trading &
  Engineering Co.
6F, No. 12, Lane 83, Sec. 1
Kuang Fu Road
San Chung City, Taipei Hsien
Taiwan, R.O.C.
Tel: 886-2-999-8281
Fax: 886-2-999-8283

INDIA

Interex India
1372/A, 31st 'B' Cross
4th 'T' Block, Jayanagar
Bandgalore – 560 041
India

Interex India
2629 Terminal Blvd.
Mountain View, CA 94049
Tel: 415/254-0627
Fax: 415/254-1540