PI7C7100
3-Port
PCI Bridge

Pericom Semiconductor Corporation

The Complete Interface Solution

2380 Bering Drive, San Jose, California 95131
Telephone: 1-877-PERICOM, (1-877-737-4266)
Fax: (408) 435-1100, E-mail: nolimits@pericom.com
Internet: http://www.pericom.com

05/08/00

LIFE SUPPORT POLICY

Pericom Semiconductor Corporation's products are not authorized for use as critical components in life support devices or systems unless a specific written agreement pertaining to such intended use is executed between the manufacturer and an officer of PSC.

1. Life support devices or systems are devices or systems which:
   a) are intended for surgical implant into the body or
   b) support or sustain life and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.

2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Pericom Semiconductor Corporation reserves the right to make changes to its products or specifications at any time, without notice, in order to improve design or performance and to supply the best possible product. Pericom Semiconductor does not assume any responsibility for use of any circuitry described other than the circuitry embodied in a Pericom Semiconductor product. The Company makes no representations that circuitry described herein is free from patent infringement or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent, patent rights or other rights, of Pericom Semiconductor Corporation.

Table of Contents

1. Introduction/Product Features 1
2. PI7C7100 Block Diagram 3
3. Signal Definitions 4
3.1 Signal Types 4
3.2 Signals 4
3.2.1 Primary Bus Interface Signals 4
3.2.2 Secondary Bus Interface Signals 6
3.2.3 Clock Signals 8
3.2.4 Miscellaneous Signals 8
3.2.5 JTAG Boundary Scan Signals 9
3.2.6 Power and Ground 9
4. PCI Bus Operation 10
4.1 Types of Transactions 10
4.2 Single Address Phase 11
4.3 Device Select (DEVSEL#) Generation 11
4.4 Data Phase 11
4.5 Write Transactions 11
4.5.1 Posted Write Transactions 11
4.5.2 Memory Write and Invalidate Transactions 12
4.5.3 Delayed Write Transactions 12
4.5.4 Write Transaction Address Boundaries 13
4.5.5 Buffering Multiple Write Transactions 13
4.5.6 Fast Back-to-Back Write Transactions 13
4.6 Read Transactions 14
4.6.1 Prefetchable Read Transactions 14
4.6.2 Non-prefetchable Read Transactions 14
4.6.3 Read Pre-fetch Address Boundaries 14
4.6.4 Delayed Read Requests 15
4.6.5 Delayed Read Completion with Target 15
4.6.6 Delayed Read Completion on Initiator Bus 15
4.7 Configuration Transactions 16
4.7.1 Type 0 Access to PI7C7100 16
4.7.2 Type 1 to Type 0 Conversion 17
4.7.3 Type 1 to Type 1 Forwarding 18
4.7.4 Special Cycles 19
4.8 Transaction Termination 19
4.8.1 Master Termination Initiated by PI7C7100 20
4.8.2 Master Abort Received by PI7C7100 20
4.8.3 Target Termination Received by PI7C7100 21
4.8.4 Target Termination Initiated by PI7C7100 23

5. Address Decoding 25
5.1 Address Ranges 25
5.2 I/O Address Decoding 25
5.2.1 I/O Base and Limit Address Registers 25
5.2.2 ISA Mode 26
5.3 Memory Address Decoding 26
5.3.1 Memory-Mapped I/O Base and Limit Address Registers 26
5.3.2 Prefetchable Memory Base and Limit Address Registers 27
5.4 VGA Support 28
5.4.1 VGA Mode 28
5.4.2 VGA Snoop Mode 28
6. Transaction Ordering 29
6.1 Transactions Governed by Ordering Rules 29
6.2 General Ordering Guidelines 29
6.3 Ordering Rules 30
6.4 Data Synchronization 31
7. Error Handling 32
7.1 Address Parity Errors 32
7.2 Data Parity Errors 32
7.2.1 Configuration Write Transactions to Configuration Space 32
7.2.2 Read Transactions 33
7.2.3 Delayed Write Transactions 33
7.2.4 Posted Write Transactions 35
7.3 Data Parity Error Reporting Summary 36
7.4 System Error (SERR#) Reporting 42
8. Exclusive Access 43
8.1 Concurrent Locks 43
8.2 Acquiring Exclusive Access across PI7C7100 43
8.3 Ending Exclusive Access 44
9. PCI Bus Arbitration 45
9.1 Primary PCI Bus Arbitration 45
9.2 Secondary PCI Bus Arbitration 45
9.2.1 Secondary Bus Arbitration Using the Internal Arbiter 45
9.2.2 Secondary Bus Arbitration Using an External Arbiter 46
9.2.3 Bus Parking 46
10. Clocks 47
10.1 Primary Clock Inputs 47
10.2 Secondary Clock Outputs 47
11. Reset 48
11.1 Primary Interface Reset 48
11.2 Secondary Interface Reset 48
11.3 Chip Reset 48
12. Supported Commands 49
12.1 Primary Interface 49
12.2 Secondary Interface 51
13. Configuration Registers 52
13.1 Config Register 1 52
13.2 Config Register 2 53

13.2.1 Config Register 1 or 2:Vendor ID Register (read only, bit 15-0; offset 00h) 54
13.2.2 Config Register 1: Device ID Register (read only, bit 31-16; offset 00h) 54
13.2.3 Config Register 2: Device ID Register (read only, bit 31-16; offset 00h) 54
13.2.4 Config Register 1: Command Register (bit 15-0; offset 04h) 54
13.2.5 Config Register 2: Command Register (bit 15-0; offset 04h) 55
13.2.6 Config Register 1 or 2: Status Register (for primary bus, bit 31-16; offset 04h) 56
13.2.7 Config Register 1 or 2: Revision ID Register (read only, bit 7-0; offset 08h) 57
13.2.8 Config Register 1 or 2: Class Code Register (read only, bit 31-8; offset 08h) 57
13.2.9 Config Register 1 or 2: Cache Line Size Register (read/write, bit 7-0; offset 0Ch) 57
13.2.10 Config Register 1: Primary Latency Timer Register (read/write, bit 15-8; offset 0Ch) 57
13.2.11 Config Register 2: Primary Latency Timer Register (read/write, bit 15-8; offset 0Ch) 57
13.2.12 Config Register 1: Header Type Register (read only, bit 23-16; offset 0Ch) 57
13.2.13 Config Register 2: Header Type Register (read only, bit 23-16; offset 0Ch) 57
13.2.14 Config Register 1: Primary Bus Number Register (read/write, bit 7-0; offset 18h) 57
13.2.15 Config Register 2: Primary Bus Number Register (read/write, bit 7-0; offset 18h) 57
13.2.16 Config Register 1 or 2: Secondary Bus Number Register (read/write, bit 15-8; offset 18h) 57
13.2.17 Config Register 1 or 2: Subordinate Bus Number Register (read/write, bit 23-16; offset 18h) 57
13.2.18 Config Register 1 or 2: Secondary Latency Timer (read/write, bit 31-24; offset 18h) 57
13.2.19 Config Register 1 or 2: I/O Base Register (read/write, bit 7-0; offset 1Ch) 57
13.2.20 Config Register 1 or 2: I/O Limit Register (read/write, bit 15-8; offset 1Ch) 57
13.2.21 Config Register 1 or 2: Secondary Status Register (bit 31-16; offset 1Ch) 58
13.2.22 Config Register 1 or 2: Memory Base Register (read/write, bit 15-0; offset 20h) 59
13.2.23 Config Register 1 or 2: Memory Limit Register (read/write, bit 31:16; offset 20h) 59
13.2.24 Config Register 1 or 2: Prefetchable Memory Base Register (read/write, bit 15-0;offset 24h) 59
13.2.25 Config Register 1 or 2: Prefetchable Memory Limit Register (read/write, bit 31-16; offset 24h) 59
13.2.26 Config Register 1 or 2: I/O Base Address Upper 16 Bits Register (read/write, bit 15-0; offset 30h) 59
13.2.27 Config Register 1 or 2: I/O Limit Address Upper 16 Bits Register (read/write, bit 31-16; offset 30h) 59
13.2.28 Config Register 1 or 2: Subsystem Vendor ID (read/write, bit 15-0; offset 34h) 59
13.2.29 Config Register 1 or 2: Subsystem ID (read/write, bit 31-16; offset 34h) 59
13.2.30 Config Register 1 or 2: Interrupt Pin Register (read only, bit 15-8; offset 3Ch) 59
13.2.31 Config Register 1 or 2: Bridge Control Register (bit 31-16; offset 3Ch) 60
13.2.32 Config Register 1 or 2: Diagnostic/Chip Control Register (bit 15-0; offset 40h) 61
13.2.33 Config Register 1 or 2: Arbiter Control Register (bit 31-16; offset 40h) 61
13.2.34 Config Register 1: Primary Prefetchable Memory Base Register (Read/Write, bit 15-0; offset 44h) 62
13.2.35 Config Register 2: Primary Prefetchable Memory Base Register (Read/Write, bit 15-0; offset 44h) 62
13.2.36 Config Register 1: Primary Prefetchable Memory Limit Register (Read/Write, bit 31-16; offset 44h) 62
13.2.37 Config Register 2: Primary Prefetchable Memory Limit Register (Read/Write, bit 31-16; offset 44h) 62
13.2.38 Config Register 1 or 2: P_SERR# Event Disable Register (bit 7-0; offset 64h) 62
13.2.39 Config Register 1: Secondary Clock Control Register (bit 15-0; offset 68h) 63
13.2.40 Config Register 2: Secondary Clock Control Register (bit 15-0; offset 68h) 63
13.2.41 Config Register 1 or 2: Non-Posted Memory Base Register (read/write, bit 15-0; offset 70h) 64
13.2.42 Config Register 1 or 2: Non-Posted Memory Limit Register (read/write, bit 31-16; offset 70h) 64
13.2.43 Config Register 1: Port Option Register (bit 15-0; offset 74h) 64
13.2.44 Config Register 2: Port Option Register (bit 15-0; offset 74h) 65
13.2.45 Config Register 1 or 2: Master Timeout Counter Register (read/write, bit 31-16; offset 74h) 66
13.2.46 Config Register 1 or 2: Retry Counter Register (read/write, bit 31-0; offset 78h) 66
13.2.47 Config Register 1 or 2: Sampling Timer Register (read/write, bit 31-0; offset 7Ch) 66
13.2.48 Config Register 1 or 2: Successful I/O Read Count Register (read/write, bit 31-0; offset 80h) 66
13.2.49 Config Register 1 or 2: Successful I/O Write Count Register (read/write, bit 31-0; offset 84h) 66
13.2.50 Config Register 1 or 2: Successful Memory Read Count Register (read/write, bit 31-0; offset 88h) 66

13.2.51 Config Register 1 or 2: Successful Memory Write Count Register (read/write, bit 31-0; offset 8Ch) 66
13.2.52 Config Register 1: Primary Successful I/O Read Count Register (read/write, bit 31-0; offset 90h) 66
13.2.53 Config Register 1: Primary Successful I/O Write Count Register (read/write, bit 31-0; offset 94h) 66
13.2.54 Config Register 1: Primary Successful Memory Read Count Register (read/write, bit 31-0; offset 98h) 66
13.2.55 Config Register 1: Primary Successful Memory Write Count Register (read/write, bit 31-0; offset 9Ch) 66
14. Bridge Behavior 67
14.1 Bridge Actions for Various Cycle Types 67
14.2 Transaction Ordering 67
14.3 Abnormal Termination (Initiated by Bridge Master) 68
14.3.1 Master Abort 68
14.3.2 Parity and Error Reporting 68
14.3.3 Reporting Parity Errors 68
14.3.4 Secondary IDSEL mapping 68
15. IEEE 1149.1 Compatible JTAG Controller 69
15.1 Boundary Scan Architecture 69
15.1.1 TAP Pins 69
15.1.2 Instruction Register 69
15.2 Boundary Scan Instruction Set 70
15.3 TAP Test Data Registers 70
15.4 Bypass Register 71
15.5 Boundary-Scan Register 71
15.6 TAP Controller 71
16. Electrical and Timing Specifications 76
16.1 Maximum Ratings 76
16.2 3.3V DC Specifications 76
16.3 3.3V AC Specifications 77
16.4 Primary and Secondary buses at 33 MHz clock timing 77
17. 256-Pin PBGA Package 78
17.1 Part Number Ordering Information 78

List of Figures

1-1. PI7C7100 on the System Board 2
1-2. PI7C7100 in Redundant Applications 2
1-3. PI7C7100 on Network Switching Hub 2
2-1. PI7C7100 Block Diagram 3
9-1. Secondary Arbiter Example 45
15-1. Test Access Port Block Diagram 69
17-1. 256-Pin PBGA Package Drawing 74

List of Tables

4-1. PCI Transaction 10
4-2. Write Transaction Forwarding 11
4-3. Write Transaction Disconnect Address Boundaries 13
4-4. Read Pre-fetch Address Boundaries 14
4-5. Read Transaction Pre-fetching 15
4-6. Device Number to IDSEL S1_AD or S2_AD Pin Mapping 18
4-7. Posted Write Target Termination Response 21
4-8. Responses to Posted Write Target Termination 22
4-9. Responses to Delayed Read Target Termination 22
6-1. Summary of Tranaction Ordering 30
7-1. Setting the Primary Interface Detected Parity Error Bit 36
7-2. Setting the Secondary Interface Detected Parity Error Bit 37
7-3. Setting the Primary Interface Data Parity Detected Bit 37
7-4. Setting the Secondary InterfaceData Parity Detected Bit 38
7-5. Assertion of P_PERR# 39
7-6. Assertion of S_PERR# 40
7-7. Assertion of P_SERR# for Data Parity Errors 41
15-1. TAP Pins 70
15-2. JTAG Boundary Register Order 72

Appendix A - Timing Diagrams

1. Configuration Read Transaction A-3
2. Configuration Write Transaction A-3
3. Type 1 to Type 0 Configuration Read Transaction (P → S) A-3
4. Type 1 to Type 0 Configuration Write Transaction (P → S) A-4
5. Upstream Type 1 to Special Cycle Transaction (S → P) A-4
6. Downstream Type 1 to Special Cycle Transaction (P → S) A-5
7. Downstream Type 1 to Type 1 Configuration Read Transaction (P → S) A-5
8. Downstream Type 1 to Type 1 Configuration Write Transaction (P → S) A-6
9. Upstream Delayed Burst Memory Read Transaction (S → P) A-6
10. Downstream Delayed Burst Memory Read Transaction (P → S) A-7
11. Downstream Delayed Memory Read Transaction (P/33MHz → S/33MHz) A-7
12. Downstream Delayed Memory Read Transaction (S2/33MHz → S1/33MHz) A-8
13. Downstream Delayed Memory Read Transaction (S1/33MHz → S2/33MHz) A-8
14. Upstream Delayed Memory Read Transaction (S/33MHz → P/33MHz) A-9
15. Downstream Posted Memory Write Transaction (P/33MHz → S/33MHz) A-9
16. Downstream Posted Memory Write Transaction (S2/33MHz → S1/33MHz) A-10
17. Downstream Posted Memory Write Transaction (S1/33MHz → S2/33MHz) A-10
18. Upstream Posted Memory Write Transaction (S/33MHz → P/33MHz) A-11
19. Downstream Flow-Through Posted Memory Write Transaction (P/33MHz →S/33MHz) A-11
20. Downstream Flow-Through Posted Memory Write Transaction (S2/33MHz →S1/33MHz) A-12
21. Downstream Flow-Through Posted Memory Write Transaction (S1/33MHz →S2/33MHz) A-12
22. Upstream Flow-Through Posted Memory Write Transaction (S/33MHz →P/33MHz) A-13
23. Downstream Delayed I/O Read Transaction (P → S) A-13
24. Downstream Delayed I/O Read Transaction (S2/33MHz → S1/33MHz) A-14
25. Downstream Delayed I/O Read Transaction (S1/33MHz → S2/33MHz) A-14
26. Downstream Delayed I/O Read Transaction (S/33MHz → P/33MHz) A-15
27. Downstream Delayed I/O Write Transaction (P → S) A-15
28. Downstream Delayed I/O Write Transaction (S2/33MHz → S1/33MHz) A-16
29. Downstream Delayed I/O Write Transaction (S1/33MHz → S2/33MHz) A-16
30. Upstream Delayed I/O Write Transaction (S → P) A-17

Appendix B - Evaluation Board User's Manual

General Information B-3
Frequently Asked Questions B-5

Appendix C - Three-Port PCI Bridge Evaluation Board Schematics

PCI Chip C-3
PCI Edge Connector C-4
Secondary 1 PCI Bus C-5
Secondary 2 PCI Bus C-6
Top View C-7

1. Introduction

Product Description

PI7C7100 is the first triple port PCI-to-PCI Bridge device designed to be fully compliant with the 32-bit, 33MHz implementation of the *PCI Local Bus Specification, Revision 2.1*. PI7C7100 supports only synchronous bus transactions between devices on the primary 33 MHz bus and the secondary buses operating at 33 MHz. The primary and the secondary buses can also operate in concurrent mode, resulting in added increase in system performance. Concurrent bus operation off-loads and isolates unnecessary traffic from the primary bus; thereby enabling a master and a target device on the same secondary PCI bus to communicate even while the primary bus is busy.

Product Features

- 32-bit Primary & two Secondary Ports run up to 33 MHz
- All three ports compliant with the PCI Local Bus Specification, Revision 2.1
- Compliant with PCI-to-PCI Bridge Architecture Specification, Revision 1.0.
  - All I/O and memory commands
  - Type 1 to Type 0 configuration conversion
  - Type 1 to Type 1 configuration forwarding
  - Type 1 configuration-write to special cycle conversion
- Concurrent primary to secondary bus operation and independent intra-secondary port channel to reduce traffic on the primary port
- Provides internal arbitration for two sets of eight secondary bus masters
  - Programmable 2-level priority arbiter
  - Disable control for use of external arbiter
- Supports posted write buffers on all directions
- Three 128 byte FIFOs
- Enhanced address decoding
  - 32-bit I/O address range
  - 32-bit memory-mapped I/O address range
  - VGA addressing and VGA palette snooping
  - ISA-aware mode for legacy support in the first 64KB of I/O address range
- Interrupt Handling
  - PCI interrupts are routed through an external interrupt concentrator
- Supports system transaction ordering rules
- Hot-plug support on secondary buses
  - 3-State control of output buffers
- IEEE 1149.1 JTAG interface support
- 3.3V core; 3.3V PCI I/O interface with 5V I/O Tolerant
- 256-pin plastic BGA package

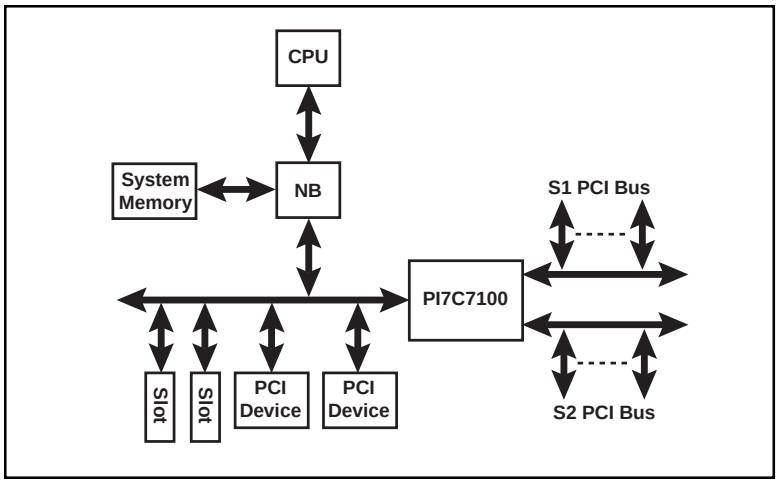


Figure 1-1. PI7C7100 on the System Board

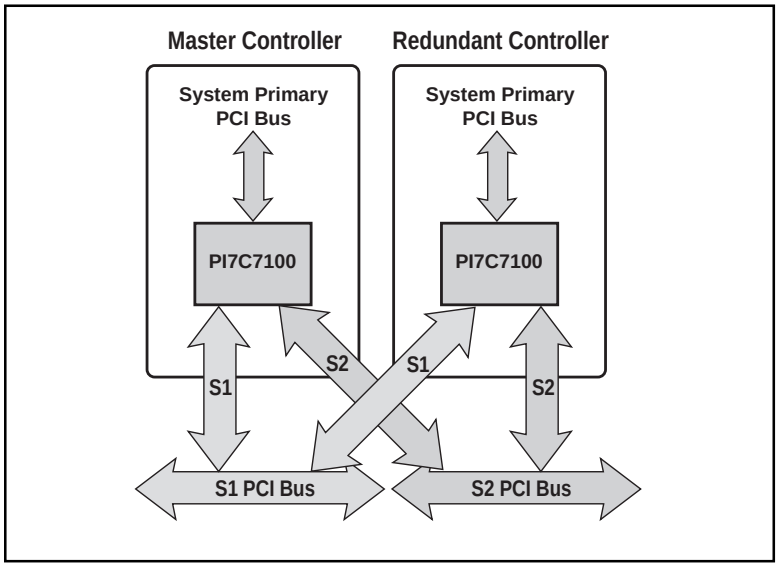


Figure 1-2. PI7C7100 in Redundant Application

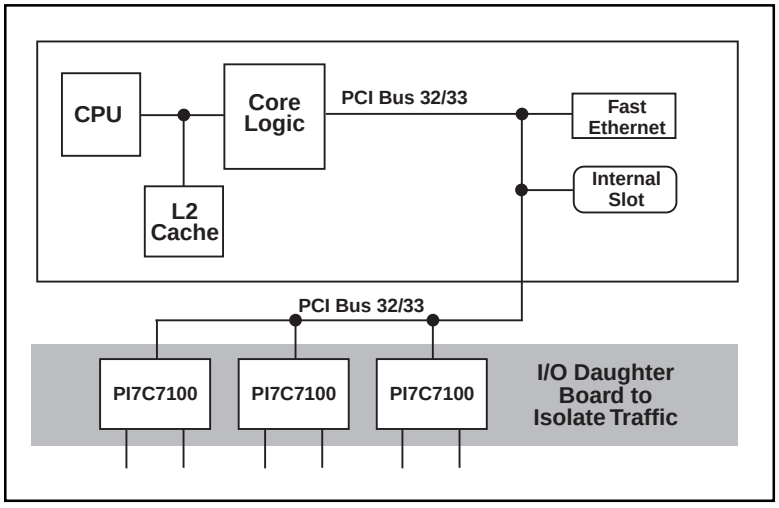


Figure 1-3. PI7C7100 on Network Switching Hub

2. PI7C7100 Block Diagram

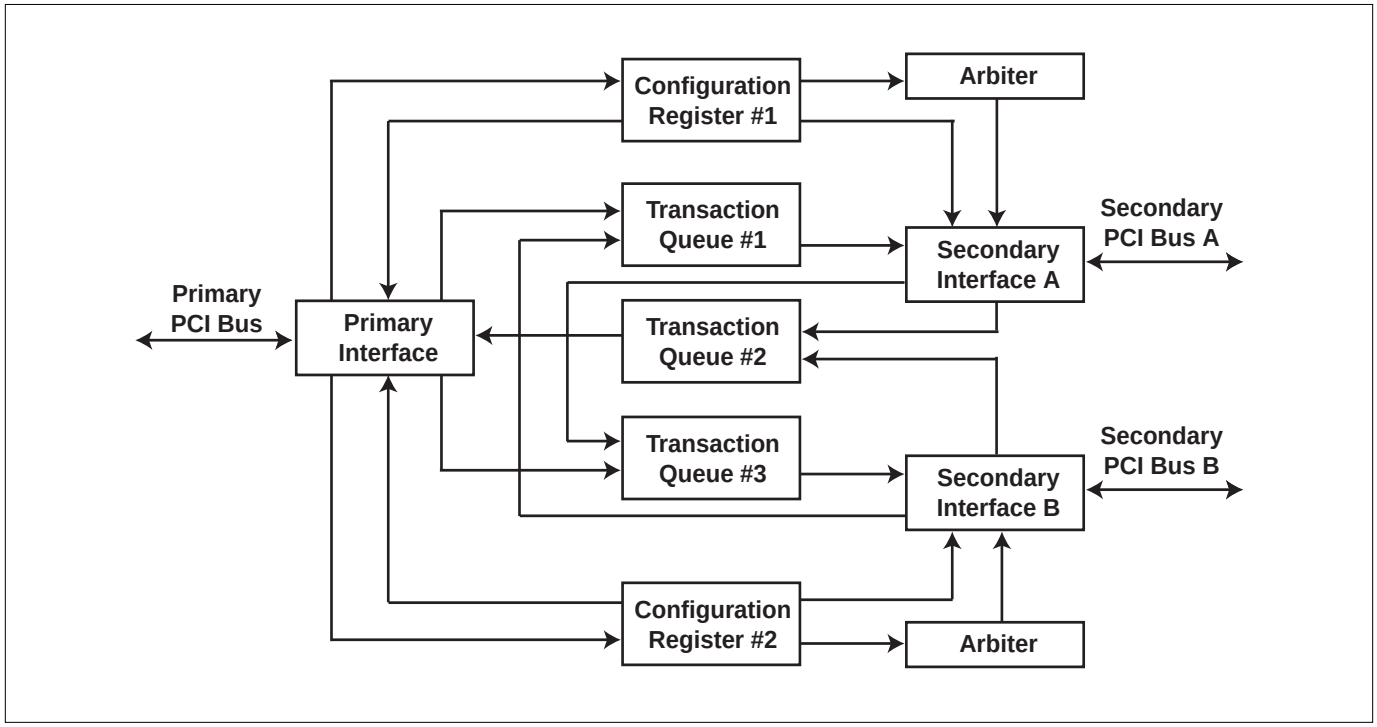


Figure 2-1. PI7C7100 Block Diagram

3. Signal Definitions

3.1 Signal Types

- PI - PCI Input (3.3V, 5V tolerant)
- PIU - PCI Input (3.3V, 5V tolerant) with weak pull-up
- PB - PCI 3-state bidirectional (3.3V, 5V tolerant)
- PO - PCI Output (3.3V)
- PSTS - PCI Sustained 3-state bidirectional (Active LOW signal which must be driven inactive for one cycle before being 3-stated to ensure HIGH performance on a shared signal line)
- PTS - PCI 3-state Output
- POD - PCI Output which either drives LOW (active state) or 3-stated
- CI - CMOS Input
- CIU - CMOS Input with weak pull-up
- CID - CMOS Input with weak pull-down
- CTO – CMOS 3-state Output

Note: All I/Os can operate off at 3.3V

3.2 Signals (**Note:** Signal name that ends with character '**#**' is active LOW.)

3.2.1 Primary Bus Interface Signals

Name	Pin #	Type	Description
P_AD[31:0]	Y7, W7, Y8, W8, V8, U8, Y9, W9, W10, V10, Y11, V11, U11, Y12, W12, V12, V16, W16, Y16, W17, Y17, U18, W18, Y18, U19, W19, Y19, U20, V20, Y20, T17, R17	PB	**Primary Address/Data.** Multiplexed address and data bus. Address is indicated by P_FRAME# assertion. Write data is stable and valid when P_IRDY# is asserted and read data is stable and valid when P_TRDY# is asserted. Data is transferred on rising clock edges when both P_IRDY# and P_TRDY# are asserted. During bus idle, PI7C7100 drives P_AD to a valid logic level when P_GNT# is asserted.
P_CBE[3:0]	V9, U12, U16, V19	PB	**Primary Command/Byte Enables.** Multiplexed command field and byte enable field. During address phase, the initiator drives the transaction type on these pins. After that the initiator drives the byte enables during data phases. During bus idle, PI7C7100 drives P_CBE[3:0] to a valid logic level when P_GNT# is asserted.
P_PAR	U15	PB	**Primary Parity.** Parity is even across P_AD[31:0], P_CBE[3:0], and P_PAR (i.e. an even number of '1's). P_PAR is an input and is valid and stable one cycle after the address phase (indicated by assertion of P_FRAME#) for address parity. For write data phases, P_PAR is an input and is valid one clock after P_IRDY# is asserted. For read data phase, P_PAR is an output and is valid one clock after P_TRDY# is asserted. Signal P_PAR is tri-stated one cycle after the PAD lines are 3-stated. During bus idle, PI7C7100 drives PPAR to a valid logic level when P_GNT# is asserted.
P_FRAME#	W13	PSTS	**Primary FRAME (Active LOW).** Driven by the initiator of a transaction to indicate the beginning and duration of an access. The de-assertion of P_FRAME# indicates the final data phase requested by the initiator. Before being 3-stated, it is driven to a de-asserted state for one cycle.
P_IRDY#	V13	PSTS	**Primary IRDY (Active LOW).** Driven by the initiator of a transaction to indicate its ability to complete current data phase on the primary side. Once asserted in a data phase, it is not de-asserted until end of data phase. Before being 3-stated, it is driven to a de-asserted state for one cycle.
P_TRDY#	U13	PSTS	**Primary TRDY (Active LOW).** Driven by the target of a transaction to indicate its ability to complete current data phase on the primary side. Once asserted in a data phase, it is not de-asserted until end of data phase. Before being 3-stated, it is driven to a de-asserted state for one cycle.

3.2.1 Primary Bus Interface Signals (continued)

Name	Pin #	Type	Description
P_DEVSEL#	Y14	PSTS	**Primary Device Select (Active LOW).** Asserted by the target indicating that the device is accepting the transaction. As a master, PI7C7100 waits for the assertion of this signal within 5 cycles of P_FRAME# assertion; otherwise, terminate with master abort. Before being 3-stated, it is driven to a de-asserted state for one cycle.
P_STOP#	W14	PSTS	**Primary STOP (Active LOW).** Asserted by the target indicating that the target is requesting the initiator to stop the current transaction. Before being 3-stated, it is driven to a de-asserted state for one cycle.
P_LOCK#	V14	PSTS	**Primary LOCK (Active LOW).** Asserted by master for multiple transactions to complete.
P_IDSEL	Y10	PI	**Primary ID Select.** Used as chip select line for Type 0 configuration access to PI7C7100 configuration space.
P_PERR#	Y15	PSTS	**Primary Parity Error (Active LOW).** Asserted when a data parity error is detected for data received on the primary interface. Before being 3-stated, it is driven to a de-asserted state for one cycle.
P_SERR#	W15	POD	**Primary System Error (Active LOW).** Can be driven LOW by any device to indicate a system error condition, PI7C7100 drives this pin on: • Address parity error • Posted write data parity error on target bus • Secondary S1_SERR# or S2_SERR# asserted • Master abort during posted write transaction • Target abort during posted write transaction • Posted write transaction discarded • Delayed write request discarded • Delayed read request discarded • Delayed transaction master timeout This signal requires an external pull-up resistor for proper operation.
P_REQ#	W6	PTS	**Primary Request (Active LOW).** This is asserted by PI7C7100 to indicate that it wants to start a transaction on the primary bus. PI7C7100 de-asserts this pin for at least 2 PCI clock cycles before asserting it again.
P_GNT#	U7	PI	**Primary Grant (Active LOW)**. When asserted, PI7C7100 can access the primary bus. During idle and P_GNT# asserted, PI7C7100 will drive P_AD, P_CBE and P_PAR to valid logic levels.
P_RESET#	Y5	PI	**Primary RESET (Active LOW).** When P_RESET# is active, all PCI signals should be asynchronously 3-stated.
P_FLUSH#	W5	PI	**Primary FIFO FLUSH (Active LOW).** When P_FLUSH# is active, all the primary FIFO(s) are cleared (invalidate all primary transactions). This signal should be pulled to a static "high."
P_M66EN	V18	–	**Reserved for Future Use.** Must be tied to ground.

3.2.2 Secondary Bus Interface Signals

Name	Pin #	Type	Description
S1_AD[31:0], S2_AD[31:0]	B20, B19, C20, C19, C18, D20, D19, D17, E19, E18, E17, F20, F19, F17, G20, G19, L20, L19, L18, M20, M19, M17, N20, N19, N18, N17, P17, R20, R19, R18, T20, T19 J4, H1, H2, H3, H4, G1, G3, G4, F2, F3, F4, E1, E4, D1, C1, B1, C5, B5, D6, C6, B6, A6, C7, B7, D8, C8, D9, C9, B9, A9, D10, C10	PB	**Secondary Address/Data.** Multiplexed address and data bus. Address is indicated by S1_FRAME# or S2_FRAME# assertion. Write data is stable and valid when S1_IRDY# or S2_IRDY# is asserted and read data is stable and valid when S1_TRDY# or S2_TRDY# is asserted. Data is transferred on rising clock edges when both S1_IRDY# and S1_TRDY# or S2_IRDY# and S2_TRDY# are asserted. During bus idle, PI7C7100 drives S1_AD or S2_AD to a valid logic level when the S1_GNT# or S2_GNT# is asserted respectively.
S1_CBE[3:0], S2_CBE[3:0]	E20, G18, K17, P20 F1, A1, A4, A7	PB	**Secondary Command/Byte Enables.** Multiplexed command field and byte enable field. During the address phase, the initiator drives the transaction type on these pins. After that the initiator drives the byte enables during data phases. During bus idle, PI7C7100 drives S1_CBE[3:0] or S2_CBE[3:0] to a valid logic level when the internal grant is asserted.
S1_PAR, S2_PAR	K18, B4	PB	**Secondary Parity.** Parity is even across S1_AD[31:0], S1_CBE[3:0], and S1_PAR or S2_AD[31:0], S2_CBE[3:0], and S2_PAR (i.e. an even number of '1's). S1_PAR or S2_PAR is an input and is valid and stable one cycle after the address phase (indicated by assertion of S1_FRAME# or S2_FRAME#) for address parity. For write data phases, S1_PAR or S2_PAR is an input and is valid one clock after S1_IRDY# or S2_IRDY# is asserted. For read data phase, S1_PAR or S2_PAR is an output and is valid one clock after S1_TRDY# or S2_TRDY# is asserted. Signal S1_PAR or S2_PAR is 3-stated one cycle after the S1_AD or S2_AD lines are tri-stated. During bus idle, PI7C7100 drives S1_PAR or S2_PAR to a valid logic level when the internal grant is asserted.
S1_FRAME#, S2_FRAME#	H20, D2	PSTS	**Secondary FRAME (Active LOW).** Driven by the initiator of a transaction to indicate the beginning and duration of an access. De-assertion of S1_FRAME# or S2_FRAME# indicates the final data phase requested by initiator. Before being 3-stated, it is driven to a de-asserted state for one cycle.
S1_IRDY#, S2_IRDY#	H19, B2	PSTS	**Secondary IRDY (Active LOW).** Driven by the initiator of a transaction to indicate its ability to complete the current data phase on the primary side. Once asserted in a data phase, it is not de-asserted until end of the data phase. Before being 3-stated, it is driven to a de-asserted state for one cycle.
S1_TRDY#, S2_TRDY#	H18, A2	PSTS	**Secondary TRDY (Active LOW).** Driven by the target of a transaction to indicate its ability to complete the current data phase on the primary side. Once asserted in a data phase, it is not de-asserted until end of the data phase. Before being 3-stated, it is driven to a de-asserted state for one cycle.
S1_DEVSEL#, S2_DEVSEL#	J20, D3	PSTS	**Secondary Device Select (Active LOW).** Asserted by the target indicating that the device is accepting the transaction. As a master, PI7C7100 waits for the assertion of this signal within 5 cycles of S1_FRAME# or S2_FRAME# assertion; otherwise, terminate with master abort. Before being 3-stated, it is driven to a de-asserted state for one cycle.

3.2.2 Secondary Bus Interface Signals (continued)

Name	Pin #	Type	Description
S1_STOP#, S2_STOP#	J19, C3	PSTS	**Secondary STOP (Active LOW).** Asserted by the target indicating that the target is requesting the initiator to stop the current transaction. Before being 3-stated, it is driven to a de-asserted state for one cycle.
S1_LOCK#, S2_LOCK#	J18, B3	PSTS	**Secondary LOCK (Active LOW).** Asserted by master for multiple transactions to complete.
S1_PERR#, S2_PERR#	J17, D4	PSTS	**Secondary Parity Error (Active LOW).** Asserted when a data parity error is detected for data received on the secondary interface. Before being 3-stated, it is driven to a de-asserted state for one cycle.
S1_SERR#, S2_SERR#	K20, C4	PI	**Secondary System Error (Active LOW).** Can be driven LOW by any device to indicate a system error condition.
S1_REQ#[7:0], S2_REQ#[7:0]	B11, A12, D13, C13, C15, A16, C17, B17 T2, R3, P2, P1, M2, M1, K1, K3	PIU	**Secondary Request (Active LOW).** This is asserted by an external device to indicate that it wants to start a transaction on the Secondary bus. The input is externally pulled up through a resistor to VDD.
S1_GNT#[7:0], S2_GNT#[7:0]	C11, B12, B13, A14, D14, B16, D16, B18 U1, P4, R1, N4, M3, L4, L1, K2	PO	**Secondary Grant (Active LOW).** PI7C7100 asserts this pin to access the secondary bus. PI7C7100 de-asserts this pin for at least 2 PCI clock cycles before asserting it again. During idle and S1_GNT# or S2_GNT# asserted, PI7C7100 will drive S1_AD, S1_CBE and S1_PAR or S2_AD, S2_CBE and S2_PAR to valid logic levels.
S1_RESET#, S2_RESET#	B10, T4	PO	**Secondary RESET (Active LOW).** Asserted when any of the following conditions are met: 1. Signal P_RESET# is asserted. 2. Secondary reset bit in bridge control register in configuration space is set. When asserted, all control signals are 3-stated and zeros are driven on S1_AD, S1_CBE, and S1_PAR or S2_AD, S2_CBE, and S2_PAR.
S1_EN, S2_EN	W3, W4	PIU	**Secondary Enable (Active HIGH).** When S1_EN or S2_EN is inactive, secondary PCI S1 or S2 bus will be asynchronously 3-stated.
S_M66EN	D7	–	**Reserved for Future Use.** Must be tied to ground.
S_CFN#	Y2	CIU	**Secondary Bus Central Function Control Pin.** When tied LOW, it enables the internal arbiter. When tied HIGH, an external arbiter must be used. S1_REQ0# or S2_REQ0# is reconfigured to be the secondary bus grant input, and S1_GNT0# or S2_GNT0# is reconfigured to be the secondary bus request output.

3.2.3 Clock Signals

Name	Pin #	Type	Description
P_CLK	V6	PI	**Primary Clock Input**. Provides timing for all transaction on primary interface.
S_CLKOUT [15:0]	T3, T1, P3, N3,M4, L3, L2, J1,A11, C12, A13, B14, B15, C16, A18, A19	PTS	**Secondary Clock Output.** Provides secondary clocks phase synchronous with the P_CLK.

3.2.4 Miscellaneous Signals

Name	Pin #	Type	Description
BYPASS	Y4	–	**Reserved for Future Use.** Must be tied HIGH.
PLL_TM	Y3	–	**Reserved for Future Use.** Must be tied LOW.
S_CLKIN	V5	PI	**Secondary Test Clock Input.** It should be tied to LOW in normal mode. It also may be a secondary clock input for the secondary buses if both SCAN_TM# and SCAN_EN are connected to logic "1".
SCAN_TM#	V4	CI	**Full-scan Test Mode enable (Active LOW).** When SCAN_TM# is active, the twelve scan chains will be enabled. The scan clock is P_CLK. The scan inputs and outputs are as follows: S1_REQ[7], S1_REQ[6], S1_REQ[5], S1_REQ[4], S1_REQ[3], S1_REQ[2], S2_REQ[7], S2_REQ[6], S2_REQ[5], S2_REQ[4], S2_REQ[3], S2_REQ[2] and S1_GNT[7], S1_GNT[6], S1_GNT[5], S1_GNT[4], S1_GNT[3], S1_GNT[2], S2_GNT[7], S2_GNT[6], S2_GNT[5], S2_GNT[4], S2_GNT[3], S2_GNT[2] respectively
SCAN_EN	U5	CIU	**Full-scan Enable Control.** When SCAN_EN is LOW, full-scan is in shift operation if SCAN_TM# is active. When SCAN_EN is HIGH, full-scan is in parallel operation if SCAN_TM# is active. SCAN_EN should be tied LOW in normal mode. If SCAN_TM# and SCAN_EN are connected to logic "1", S_CLKIN is the clock source for the internal secondary clock. If SCAN_TM# is connected to logic "1" and SCAN_EN is connected to logic "0", P_CLK is the clock source for the internal secondary clock. **Note:** During power-up, SCAN_EN is the reset signal for the on-chip PLL.
CMPO1	U6	–	**Reserved for Future Use.**
Reserved	R4		**Reserved**

3.2.5 JTAG Boundary Scan Signals

Name	Pin #	Type	Description
TCK	V2	CIU	**Test Clock.** Used to clock state information and data into and out of the PI7C7100 during boundary scan.
TMS	W1	CIU	**Test Mode Select**. Used to control the state of the Test Access Port controller.
TDO	V3	CTO	**Test Data Output.** When SCANEN is HIGH it is used (in conjunction with TCK) to shift data out of the Test Access Port (TAP) in a serial bit stream.
TDI	W2	CIU	**Test Data Input.** When SCANEN is HIGH it is used (in conjunction with TCK) to shift data and instructions into the Test Access Port (TAP) a serial bit stream.
TRST#	U3	CIU	**Test Reset.** Active LOW signal to reset the Test Access Port (TAP) controller into an initialized state.

3.2.6 Power and Ground

Name	Pin #	Type	Description
VDD	B8, C14, D5, D11, D15, E2, F18, J3, L17, N2, P19, U10, V1, V7, V15, W20		**+3.3V Digital Power**
VSS	A3, A5, A8, A10, A15, A17, A20, C2, D12, D18, E3, G2, G17, H17, J2, K4, K19, M18, N1, P18, R2, T18, U2, U9, U14, U17 V17 W11 Y6 Y13		**Digital Ground**
AVCC	Y1		**Analog 3.3V for PLL**
AGND	U4		**Analog Ground for PLL**

4. PCI Bus Operation

This chapter offers information about PCI transactions, transaction forwarding across PI7C7100, and transaction termination. The PI7C7100 has three 128-byte buffers for buffering of upstream and downstream transactions. These hold addresses, data, commands, and byte enables and are used for both read and write transactions.

4.1 Types of Transactions

This section provides a summary of PCI transactions performed by PI7C7100. Table 4–1 lists the command code and name of each PCI transaction. The Master and Target columns indicate support for each transaction when PI7C7100 initiates transactions as a master, on the primary (P) and secondary (S1, S2) buses, and when PI7C7100 responds to transactions as a target, on the primary (P) and secondary (S1, S2) buses.

Table 4-1. PCI Transactions

Type of Transactions		Initiates as Master		Responds as Target	
		Primary	Secondary	Primary	Secondary
0000	Interrupt acknowledge	N	N	N	N
0001	Special cycle	Y	Y	N	N
0010	I/O read	Y	Y	Y	Y
0011	I/O write	Y	Y	Y	Y
0100	Reserved	N	N	N	N
0101	Reserved	N	N	N	N
0110	Memory read	Y	Y	Y	Y
0111	Memory write	Y	Y	Y	Y
1000	Reserved	N	N	N	N
1001	Reserved	N	N	N	N
1010	Configuration read	N	Y	Y	N
1011	Configuration write	Y (Type 1 only)	Y	Y	Y (Type 1 only)
1100	Memory read multiple	Y	Y	Y	Y
1101	Dual address cycle	N	N	N	N
1110	Memory read line	Y	Y	Y	Y
1111	Memory write and invalidate	N	N	Y	Y

As indicated in Table 4–1, the following PCI commands are not supported by PI7C7100:

- PI7C7100 never initiates a PCI transaction with a reserved command code and, as a target, PI7C7100 ignores reserved command codes.
- PI7C7100 does not generate interrupt acknowledge transactions. PI7C7100 ignores interrupt acknowledge transactions as a target.
- PI7C7100 does not respond to special cycle transactions. PI7C7100 cannot guarantee delivery of a special cycle transaction to downstream buses because of the broadcast nature of the special cycle command and the inability to control the transaction as a target. To generate special cycle transactions on other PCI buses, either upstream or downstream, Type 1 configuration write must be used.
- PI7C7100 neither generates Type 0 configuration transactions on the primary PCI bus nor responds to Type 0 configuration transactions on the secondary PCI buses.
- PI7C7100 does not support DAC (Dual Address Cycle) transactions.

4.2 Single Address Phase

A 32-bit address uses a single address phase. This address is driven on P_AD[31:0], and the bus command is driven on P_CBE[3:0]. PI7C7100 supports the linear increment address mode only, which is indicated when the lowest two address bits are equal to zero. If either of the lowest two address bits is nonzero, PI7C7100 automatically disconnects the transaction after the first data transfer.

4.3 Device Select (DEVSEL#) Generation

PI7C7100 always performs positive address decoding (medium decode) when accepting transactions on either the primary or secondary buses. PI7C7100 never does subtractive decode.

4.4 Data Phase

The address phase of a PCI transaction is followed by one or more data phases. A data phase is completed when IRDY# and either TRDY# or STOP# are asserted. A transfer of data occurs only when both IRDY# and TRDY# are asserted during the same PCI clock cycle. The last data phase of a transaction is indicated when FRAME# is de-asserted and both TRDY# and IRDY# are asserted, or when IRDY# and STOP# are asserted. See Section 4.8 for further discussion of transaction termination.

Depending on the command type, PI7C7100 can support multiple data phase PCI transactions. For a detailed description of how PI7C7100 imposes disconnect boundaries, see Section 4.5.4 for write address boundaries and Section 4.6.3 read address boundaries.

4.5 Write Transactions

Write transactions are treated as either posted write or delayed write transactions.

Table 4–2 shows the method of forwarding used for each type of write operation.

Table 4-2. Write Transaction Forwarding

Type of Transaction	Type of Forwarding
Memory write	Posted
Memory write and invalidate	Posted
I/O write	Delayed
Type 1 configuration write	Delayed

For timing diagrams, see Figures 15-22 and 27-30 in Appendix A

4.5.1 Posted Write Transactions

Posted write forwarding is used for “Memory Write” and “Memory Write and Invalidate” transactions.

When PI7C7100 determines that a memory write transaction is to be forwarded across the bridge, PI7C7100 asserts DEVSEL# with medium timing and TRDY# in the same cycle, provided that enough buffer space is available in the posted memory write queue for the address and at least one DWORD of data. Under this condition, PI7C7100 accepts write data without obtaining access to the target bus. The PI7C7100 can accept one DWORD of write data every PCI clock cycle. That is, no target wait state is inserted. The write data is stored in an internal posted write buffers and is subsequently delivered to the target.

The PI7C7100 continues to accept write data until one of the following events occurs:

- The initiator terminates the transaction by de-asserting FRAME# and IRDY#.
- An internal write address boundary is reached, such as a cache line boundary or an aligned 4KB boundary, depending on the transaction type.
- The posted write data buffer fills up.

When one of the last two events occurs, the PI7C7100 returns a target disconnect to the requesting initiator on this data phase to terminate the transaction.

Once the posted write data moves to the head of the posted data queue, PI7C7100 asserts its request on the target bus. This can occur while PI7C7100 is still receiving data on the initiator bus. When the grant for the target bus is received and the target bus is detected in the idle condition, PI7C7100 asserts FRAME# and drives the stored write address out on the target bus. On the following cycle, PI7C7100 drives the first DWORD of write data and continues to transfer write data until all write data corresponding to that transaction is delivered, or until a target termination is received. As long as write data exists in the queue, PI7C7100 can drive one DWORD of write data each PCI clock cycle; that is, no master wait states are inserted. If write data is flowing through PI7C7100 and the initiator stalls, PI7C7100 may have to insert wait states on the target bus if the queue empties.

PI7C7100 ends the transaction on the target bus when one of the following conditions is met:

- All posted write data has been delivered to the target.
- The target returns a target disconnect or target retry (PI7C7100 starts another transaction to deliver the rest of write data).
- The target returns a target abort (PI7C7100 discards remaining write data).
- The master latency timer expires, and PI7C7100 no longer has the target bus grant (PI7C7100 starts another transaction to deliver remaining write data).

Section 4.8.3.2 provides detailed information about how PI7C7100 responds to target termination during posted write transactions.

4.5.2 Memory Write and Invalidate Transactions

Posted write forwarding is used for Memory Write and Invalidate transactions.

PI7C7100 always converts Memory Write and Invalidate transactions to Memory Write transactions.

The PI7C7100 disconnects Memory Write and Invalidate commands at aligned cache line boundaries. The cache line size value in the cache line size register gives the number of DWORD in a cache line.

If the value in the cache line size register does meet the memory write and invalidate conditions, the PI7C7100 returns a target disconnect to the initiator either on a cache line boundary or when the posted write buffer fills.

When the Memory Write and Invalidate transaction is disconnected before a cache line boundary is reached, typically because the posted write buffer fills, the transaction is converted to Memory Write transaction.

4.5.3 Delayed Write Transactions

Delayed write forwarding is used for I/O write transactions and Type 1 configuration write transactions.

A delayed write transaction guarantees that the actual target response is returned back to the initiator without holding the initiating bus in wait states. A delayed write transaction is limited to a single DWORD data transfer.

When a write transaction is first detected on the initiator bus, and PI7C7100 forwards it as a delayed transaction, PI7C7100 claims the access by asserting DEVSEL# and returns a target retry to the initiator. During the address phase, PI7C7100 samples the bus command, address, and address parity one cycle later. After IRDY# is asserted, PI7C7100 also samples the first data DWORD, byte enable bits, and data parity. This information is placed into the delayed transaction queue. The transaction is queued only if no other existing delayed transactions have the same address and command, and if the delayed transaction queue is not full. When the delayed write transaction moves to the head of the delayed transaction queue and all ordering constraints with posted data are satisfied. The PI7C7100 initiates the transaction on the target bus. PI7C7100 transfers the write data to the target. If PI7C7100 receives a target retry in response to the write transaction on the target bus, it continues to repeat the write transaction until the data transfer is completed, or until an error condition is encountered.

If PI7C7100 is unable to deliver write data after 2^{24}(default) or 2^{32}(maximum) attempts, PI7C7100 ceases further write attempts and returns a target abort to the initiator. The delayed transaction is removed from the delayed transaction queue. PI7C7100 also asserts P_SERR# if the primary SERR# enable bit is set in the command register. See Section 7.4 for information on the assertion of P_SERR#. When the initiator repeats the same write transaction (same command, address, byte enable bits, and data), and the completed delayed transaction is at the head of the queue, the PI7C7100

claims the access by asserting DEVSEL# and returns TRDY# to the initiator, to indicate that the write data was transferred. If the initiator requests multiple DWORD, PI7C7100 also asserts STOP# in conjunction with TRDY# to signal a target disconnect. Note that only those bytes of write data with valid byte enable bits are compared. If any of the byte enable bits are turned off (driven HIGH), the corresponding byte of write data is not compared.

If the initiator repeats the write transaction before the data has been transferred to the target, PI7C7100 returns a target retry to the initiator. PI7C7100 continues to return a target retry to the initiator until write data is delivered to the target, or until an error condition is encountered. When the write transaction is repeated, PI7C7100 does not make a new entry into the delayed transaction queue. Section 4.8.3.1 provides detailed information about how PI7C7100 responds to target termination during delayed write transactions.

PI7C7100 implements a discard timer that starts counting when the delayed write completion is at the head of the delayed transaction queue. The initial value of this timer can be set to one of two values, selectable through both the primary and secondary master timeout bits in the bridge control register. If the initiator does not repeat the delayed write transaction before the discard timer expires, PI7C7100 discards the delayed write transaction from the delayed transaction queue. PI7C7100 also conditionally asserts P_SERR# (see Section 7.4).

4.5.4 Write Transaction Address Boundaries

PI7C7100 imposes internal address boundaries when accepting write data. The aligned address boundaries are used to prevent PI7C7100 from continuing a transaction over a device address boundary and to provide an upper limit on maximum latency. PI7C7100 returns a target disconnect to the initiator when it reaches the aligned address boundaries under conditions shown in Table 4–3.

Table 4-3. Write Transaction Disconnect Address Boundaries

Type of Transaction	Condition	Aligned Address Boundary
Delayed write	All	Disconnects after one data transfer
Posted memory write	Memory write disconnect control bit = 0 (1)	4KB aligned address boundary
Posted memory write	Memory write disconnect control bit = 1 (1)	Disconnects at cache line boundary
Posted memory write and invalidate	Cache line size not equal to 1, 2, 4, 8, 16	4KB aligned address boundary
Posted memory write and invalidate	Cache line size = 1, 2, 4, 8	nth cache line boundary, where a cache line boundary is reached and less than eight free DWORD of posted write buffer space remains
Posted memory write and invalidate	Cache line size = 16	16-DWORD aligned address boundary

Note 1. Memory-write-disconnect-control bit is bit 1 of the chip control register at offset 40h in configuration space.

4.5.5 Buffering Multiple Write Transactions

PI7C7100 continues to accept posted memory write transactions as long as space for at least one DWORD of data in the posted write data buffer remains. If the posted write data buffer fills before the initiator terminates the write transaction, PI7C7100 returns a target disconnect to the initiator.

Delayed write transactions are posted as long as at least one open entry in the delayed transaction queue exists. Therefore, several posted and delayed write transactions can exist in data buffers at the same time. See Chapter 6 for information about how multiple posted and delayed write transactions are ordered.

4.5.6 Fast Back-to-Back Write Transactions

PI7C7100 can recognize and post fast back-to-back write transactions. When PI7C7100 cannot accept the second transaction because of buffer space limitations, it returns a target retry to the initiator.

4.6 Read Transactions

Delayed read forwarding is used for all read transactions crossing PI7C7100. Delayed read transactions are treated as either prefetchable or non-prefetchable. Table 4-4 shows the read behavior, prefetchable or non-prefetchable, for each type of read operation. For Timing diagrams, see Figures 11-14 and 23-26 in Appendix A

4.6.1 Prefetchable Read Transactions

A prefetchable read transaction is a read transaction where PI7C7100 performs speculative DWORD reads, transferring data from the target before it is requested from the initiator. This behavior allows a prefetchable read transaction to consist of multiple data transfers. However, byte enable bits cannot be forwarded for all data phases as is done for the single data phase of the non-prefetchable read transaction. For prefetchable read transactions, PI7C7100 forces all byte enable bits to be turned on for all data phases.

Prefetchable behavior is used for memory read line and memory read multiple transactions, as well as for memory read transactions that fall into prefetchable memory space.

The amount of data that is pre-fetched depends on the type of transaction. The amount of pre-fetching may also be affected by the amount of free buffer space available in PI7C7100, and by any read address boundaries encountered.

Pre-fetching should not be used for those read transactions that have side effects in the target device, that is, control and status registers, FIFOs, and so on. The target device's base address register or registers indicate if a memory address region is prefetchable.

4.6.2 Non-prefetchable Read Transactions

A non-prefetchable read transaction is a read transaction where PI7C7100 requests one and only one DWORD from the target and disconnects the initiator after delivery of the first DWORD of read data. Unlike prefetchable read transactions, PI7C7100 forwards the read byte enable information for the data phase.

Non-prefetchable behavior is used for I/O and configuration read transactions, as well as for memory read transactions that fall into non-prefetchable memory space.

If extra read transactions could have side effects, for example, when accessing a FIFO, use non-prefetchable read transactions to those locations. Accordingly, if it is important to retain the value of the byte enable bits during the data phase, use non-prefetchable read transactions. If these locations are mapped in memory space, use the memory read command and map the target into non-prefetchable (memory-mapped I/O) memory space to use non-prefetching behavior.

4.6.3 Read Pre-fetch Address Boundaries

PI7C7100 imposes internal read address boundaries on read pre-fetched data. When a read transaction reaches one of these aligned address boundaries, the PI7C7100 stops pre-fetched data, unless the target signals a target disconnect before the read pre-fetched boundary is reached. When PI7C7100 finishes transferring this read data to the initiator, it returns a target disconnect with the last data transfer, unless the initiator completes the transaction before all pre-fetched read data is delivered. Any leftover pre-fetched data is discarded.

Prefetchable read transactions in flow-through mode pre-fetch to the nearest aligned 4KB address boundary, or until the initiator de-asserts FRAME#. Section 4.6.6 describes flow-through mode during read operations.

Table 4-5 shows the read pre-fetch address boundaries for read transactions during non-flow-through mode.

Table 4-4. Read Pre-fetch Address Boundaries

Type of Transaction	Address Space	Cache Line Size (CLS)	Pre-fetch Aligned Address Boundary
Config read	-	-	One DWORD (no pre-fetch)
I/O read	-	-	One DWORD (no pre-fetch)
Memory read	Non-prefetchable	-	One DWORD (no pre-fetch)
Memory read	Prefetchable	CLS not equal to 1, 2, 4, 8	16-DWORD aligned address boundary
Memory read	Prefetchable	CLS = 1, 2, 4, 8	Cache line address boundary
Memory read line	-	CLS not equal to 1, 2, 4, 8	16-DWORD aligned address boundary
Memory read line	-	CLS = 1, 2, 4, 8	Cache line boundary
Memory read multiple	-	CLS not equal to 1, 2, 4, 8	32-DWORD aligned address boundary
Memory read multiple	-	CLS = 1, 2, 4, 8	2 times of cache line boundary

Table 4-5. Read Transaction Pre-Fetching

Type of Transaction	Read Behavior
I/O read	Pre-fetching never done
Configuration read	Pre-fetching never done
Memory read	Downstream: pre-fetching used if address in prefetchable space
	Upstream: pre-fetching used if pre-fetch disable is off (default)
Memory read line	Pre-fetching always used
Memory read multiple	Pre-fetching always used

See Section 5.3 for detailed information about prefetchable and non-prefetchable address spaces.

4.6.4 Delayed Read Requests

PI7C7100 treats all read transactions as delayed read transactions, which means that the read request from the initiator is posted into a delayed transaction queue. Read data from the target is placed in the read data queue directed toward the initiator bus interface and is transferred to the initiator when the initiator repeats the read transaction.

When PI7C7100 accepts a delayed read request, it first samples the read address, read bus command, and address parity. When IRDY# is asserted, PI7C7100 then samples the byte enable bits for the first data phase. This information is entered into the delayed transaction queue. PI7C7100 terminates the transaction by signaling a target retry to the initiator. Upon reception of the target retry, the initiator is required to continue to repeat the same read transaction until at least one data transfer is completed, or until a target response (target abort or master abort) other than a target retry is received.

4.6.5 Delayed Read Completion with Target

When delayed read request reaches the head of the delayed transaction queue, PI7C7100 arbitrates for the target bus and initiates the read transaction only if all previously queued posted write transactions have been delivered. PI7C7100 uses the exact read address and read command captured from the initiator during the initial delayed read request to initiate the read transaction. If the read transaction is a non-prefetchable read, PI7C7100 drives the captured byte enable bits during the next cycle. If the transaction is a prefetchable read transaction, it drives all byte enable bits to zero for all data phases. If PI7C7100 receives a target retry in response to the read transaction on the target bus, it continues to repeat the read transaction until at least one data transfer is completed, or until an error condition is encountered. If the transaction is terminated via normal master termination or target disconnect after at least one data transfer has been completed, PI7C7100 does not initiate any further attempts to read more data.

If PI7C7100 is unable to obtain read data from the target after 2^{24}(default) or 2^{32}(maximum) attempts, PI7C7100 ceases further read attempts and returns a target abort to the initiator. The delayed transaction is removed from the delayed transaction queue. The number of attempts is programmable. PI7C7100 also asserts P_SERR# if the primary SERR# enable bit is set in the command register. See Section 7.4 for information on the assertion of P_SERR#.

Once PI7C7100 receives DEVSEL# and TRDY# from the target, it transfers the data read to the opposite direction read data queue, pointing toward the opposite interface, before terminating the transaction. For example, read data in response to a downstream read transaction initiated on the primary bus is placed in the upstream read data queue. The PI7C7100 can accept one DWORD of read data each PCI clock cycle; that is, no master wait states are inserted. The number of DWORD transferred during a delayed read transaction depends on the conditions given in Table 4–5 (assuming no disconnect is received from the target).

4.6.6 Delayed Read Completion on Initiator Bus

When the transaction has been completed on the target bus, and the delayed read data is at the head of the read data queue, and all ordering constraints with posted write transactions have been satisfied, the PI7C7100 transfers the data to the initiator when the initiator repeats the transaction. For memory read transactions, PI7C7100 aliases the memory read, memory read line, and memory read multiple bus commands when matching the bus command of the transaction to the bus command in the delayed transaction queue. PI7C7100 returns a target disconnect along with the transfer of the last DWORD of read data to the initiator. If PI7C7100 initiator terminates the transaction before all read data has been transferred, the remaining read data left in data buffers is discarded.

When the master repeats the transaction and starts transferring prefetchable read data from data buffers while the read transaction on the target bus is still in progress and before a read boundary is reached on the target bus, the read transaction starts operating in flow-through mode. Because data is flowing through the data buffers from the target to the initiator, long read bursts can then be sustained. In this case, the read transaction is allowed to continue until the initiator terminates the transaction, or until an aligned 4KB address boundary is reached, or until the buffer fills, whichever comes first. When the buffer empties, PI7C7100 reflects the stalled condition to the initiator by de-asserting TRDY# until more read data is available; otherwise, PI7C7100 does not insert any target wait states. When the initiator terminates the transaction, PI7C7100 de-assertion of FRAME# on the initiator bus is forwarded to the target bus. Any remaining read data is discarded.

PI7C7100 implements a discard timer that starts counting when the delayed read completion is at the head of the delayed transaction queue, and the read data is at the head of the read data queue. The initial value of this timer is programmable through configuration register. If the initiator does not repeat the read transaction and before the discard timer expires, PI7C7100 discards the read transaction and read data from its queues. PI7C7100 also conditionally asserts P_SERR# (see Section 7.4).

PI7C7100 has the capability to post multiple delayed read requests, up to a maximum of four in each direction. If an initiator starts a read transaction that matches the address and read command of a read transaction that is already queued, the current read command is not posted as it is already contained in the delayed transaction queue.

See Section 6 for a discussion of how delayed read transactions are ordered when crossing PI7C7100.

4.7 Configuration Transactions

Configuration transactions are used to initialize a PCI system. Every PCI device has a configuration space that is accessed by configuration commands. All registers are accessible in configuration space only.

In addition to accepting configuration transactions for initialization of its own configuration space, the PI7C7100 also forwards configuration transactions for device initialization in hierarchical PCI systems, as well as for special cycle generation.

To support hierarchical PCI bus systems, two types of configuration transactions are specified: Type 0 and Type 1.

Type 0 configuration transactions are issued when the intended target resides on the same PCI bus as the initiator. A Type 0 configuration transaction is identified by the configuration command and the lowest two bits of the address set to 00b.

Type 1 configuration transactions are issued when the intended target resides on another PCI bus, or when a special cycle is to be generated on another PCI bus. A Type 1 configuration command is identified by the configuration command and the lowest two address bits set to 01b.

The register number is found in both Type 0 and Type 1 formats and gives the DWORD address of the configuration register to be accessed. The function number is also included in both Type 0 and Type 1 formats and indicates which function of a multifunction device is to be accessed. For single-function devices, this value is not decoded. The addresses of Type 1 configuration transaction include a 5-bit field designating the device number that identifies the device on the target PCI bus that is to be accessed. In addition, the bus number in Type 1 transactions specifies the PCI bus to which the transaction is targeted. For timing diagrams, see Figures 1-8 in Appendix A.

4.7.1 Type 0 Access to PI7C7100

The configuration space is accessed by a Type 0 configuration transaction on the primary interface. The configuration space cannot be accessed from the secondary bus. The PI7C7100 responds to a Type 0 configuration transaction by asserting P_DEVSEL# when the following conditions are met during the address phase:

- The bus command is a configuration read or configuration write transaction.
- Lowest two address bits P_AD[1:0] must be 00b.
- Signal P_IDSEL must be asserted.

Function code is either 0 for configuration space of S1, or 1 for configuration space of S2 as PI7C7100 is a multi-function device.

PI7C7100 limits all configuration access to a single DWORD data transfer and returns target-disconnect with the first data transfer if additional data phases are requested. Because read transactions to configuration space do not have side effects, all bytes in the requested DWORD are returned, regardless of the value of the byte enable bits.

Type 0 configuration write and read transactions do not use data buffers; that is, these transactions are completed immediately, regardless of the state of the data buffers. The PI7C7100 ignores all Type 0 transactions initiated on the secondary interface.

4.7.2 Type 1 to Type 0 Conversion

Type 1 configuration transactions are used specifically for device configuration in a hierarchical PCI bus system. A PCI-to-PCI bridge is the only type of device that should respond to a Type 1 configuration command. Type 1 configuration commands are used when the configuration access is intended for a PCI device that resides on a PCI bus other than the one where the Type 1 transaction is generated.

PI7C7100 performs a Type 1 to Type 0 translation when the Type 1 transaction is generated on the primary bus and is intended for a device attached directly to the secondary bus. PI7C7100 must convert the configuration command to a Type 0 format so that the secondary bus device can respond to it. Type 1 to Type 0 translations are performed only in the downstream direction; that is, PI7C7100 generates a Type 0 transaction only on the secondary bus, and never on the primary bus.

PI7C7100 responds to a Type 1 configuration transaction and translates it into a Type 0 transaction on the secondary bus when the following conditions are met during the address phase:

- The lowest two address bits on P_AD[1:0] are 01b.
- The bus number in address field P_AD[23:16] is equal to the value in the secondary bus number register in configuration space.
- The bus command on P_CBE[3:0] is a configuration read or configuration write transaction.
- When PI7C7100 translates the Type 1 transaction to a Type 0 transaction on the secondary interface, it performs the following translations to the address:
- Sets the lowest two address bits on S1_AD[1:0] or S2_AD[1:0] to 00b.
- Decodes the device number and drives the bit pattern specified in Table 4–6 on S1_AD[31:16] or S2_AD[31:16] for the purpose of asserting the device's IDSEL signal.
- Sets S1_AD[15:11] or S2_AD[15:11] to 0.
- Leaves unchanged the function number and register number fields.

PI7C7100 asserts a unique address line based on the device number. These address lines may be used as secondary bus IDSEL signals. The mapping of the address lines depends on the device number in the Type 1 address bits P_AD[15:11]. Table 4–6 presents the mapping that PI7C7100 uses

Table 4–6. Device Number to IDSEL S1_AD or S2_AD Pin Mapping

Device Number	P_AD<15:11>	Secondary IDSEL S1_AD[31:16] or S2_AD[31:16]	S1_AD or S2_AD Bit
0h	00000	0000 0000 0000 0001	16
1h	00001	0000 0000 0000 0010	17
2h	00010	0000 0000 0000 0100	18
3h	00011	0000 0000 0000 1000	19
4h	00100	0000 0000 0001 0000	20
5h	00101	0000 0000 0010 0000	21
6h	0110	0000 0000 0100 0000	22
7h	00111	0000 0000 1000 0000	23
8h	01000	0000 0001 0000 0000	24
9h	01001	0000 0010 0000 0000	25
Ah	01010	0000 0100 0000 0000	26
Bh	01011	0000 1000 0000 0000	27
Ch	01100	0001 0000 0000 0000	28
Dh	01101	0010 0000 0000 0000	29
Eh	01110	0100 0000 0000 0000	30
Fh	01111	1000 0000 0000 0000	31
10h-1Eh	10000-11110	0000 0000 0000 0000	-
1Fh	11111	Generate special cycle (P_AD[7:2] = 00h) 0000 0000 0000 0000 (P_AD[7:2] = 00h)	-

PI7C7100 can assert up to 16 unique address lines to be used as IDSEL signals for up to 16 devices on the secondary bus, for device numbers ranging from 0 through 15. Because of elec3cal loading constraints of the PCI bus, more than 16 IDSEL signals should not be necessary. However, if device numbers greater than 15 are desired, some external method of generating IDSEL lines must be used, and no upper address bits are then asserted. The configuration transaction is still translated and passed from the primary bus to the secondary bus. If no IDSEL pin is asserted to a secondary device, the transaction ends in a master abort.

PI7C7100 forwards Type 1 to Type 0 configuration read or write transactions as delayed transactions. Type 1 to Type 0 configuration read or write transactions are limited to a single 32-bit data transfer.

4.7.3 Type 1 to Type 1 Forwarding

Type 1 to Type 1 transaction forwarding provides a hierarchical configuration mechanism when two or more levels of PCI-to-PCI bridges are used.

When PI7C7100 detects a Type 1 configuration transaction intended for a PCI bus downstream from the secondary bus, PI7C7100 forwards the transaction unchanged to the secondary bus. Ultimately, this transaction is translated to a Type 0 configuration command or to a special cycle transaction by a downstream PCI-to-PCI bridge. Downstream Type 1 to Type 1 forwarding occurs when the following conditions are met during the address phase:

- The lowest two address bits are equal to 01b.
- The bus number falls in the range defined by the lower limit (exclusive) in the secondary bus number register and the upper limit (inclusive) in the subordinate bus number register.
- The bus command is a configuration read or write transaction.

PI7C7100 also supports Type 1 to Type 1 forwarding of configuration write transactions upstream to support upstream special cycle generation. A Type 1 configuration command is forwarded upstream when the following conditions are met:

- The lowest two address bits are equal to 01b.
- The bus number falls outside the range defined by the lower limit (inclusive) in the secondary bus number register and the upper limit (inclusive) in the subordinate bus number register.
- The device number in address bits AD[15:11] is equal to 11111b.
- The function number in address bits AD[10:8] is equal to 111b.
- The bus command is a configuration write transaction.

The PI7C7100 forwards Type 1 to Type 1 configuration write transactions as delayed transactions. Type 1 to Type 1 configuration write transactions are limited to a single data transfer.

4.7.4 Special Cycles

The Type 1 configuration mechanism is used to generate special cycle transactions in hierarchical PCI systems. Special cycle transactions are ignored by acting as a target and are not forwarded across the bridge. Special cycle transactions can be generated from Type 1 configuration write transactions in either the upstream or the downstream direction.

PI7C7100 initiates a special cycle on the target bus when a Type 1 configuration write transaction is being detected on the initiating bus and the following conditions are met during the address phase:

- The lowest two address bits on AD[1:0] are equal to 01b.
- The device number in address bits AD[15:11] is equal to 11111b.
- The function number in address bits AD[10:8] is equal to 111b.
- The register number in address bits AD[7:2] is equal to 000000b.
- The bus number is equal to the value in the secondary bus number register in configuration space for downstream forwarding or equal to the value in the primary bus number register in configuration space for upstream forwarding.
- The bus command on CBE# is a configuration write command.

When PI7C7100 initiates the transaction on the target interface, the bus command is changed from configuration write to special cycle. The address and data are forwarded unchanged. Devices that use special cycles ignore the address and decode only the bus command. The data phase contains the special cycle message. The transaction is forwarded as a delayed transaction, but in this case the target response is not forwarded back (because special cycles result in a master abort). Once the transaction is completed on the target bus, through detection of the master abort condition, PI7C7100 responds with TRDY# to the next attempt of the configuration transaction from the initiator. If more than one data transfer is requested, PI7C7100 responds with a target disconnect operation during the first data phase.

4.8 Transaction Termination

This section describes how PI7C7100 returns transaction termination conditions back to the initiator.

The initiator can terminate transactions with one of the following types of termination:

- **Normal termination**
  Normal termination occurs when the initiator de-asserts FRAME# at the beginning of the last data phase, and de-asserts IRDY# at the end of the last data phase in conjunction with either TRDY# or STOP# assertion from the target.
- **Master abort**
  A master abort occurs when no target response is detected. When the initiator does not detect a DEVSEL# from the target within five clock cycles after asserting FRAME#, the initiator terminates the transaction with a master abort. If FRAME# is still asserted, the initiator de-asserts FRAME# on the next cycle, and then de-asserts IRDY# on the following cycle. IRDY# must be asserted in the same cycle in which FRAME# de-asserts. If FRAME# is already de-asserted, IRDY# can be de-asserted on the next clock cycle following detection of the master abort condition.

The target can terminate transactions with one of the following types of termination:

- **Normal termination**—TRDY# and DEVSEL# asserted in conjunction with FRAME# de-asserted and IRDY# asserted.
- **Target retry**—STOP# and DEVSEL# asserted with TRDY# de-asserted during the first data phase. No data transfers occur during the transaction. This transaction must be repeated.
- **Target disconnect with data transfer**—STOP#, DEVSEL# and TRDY# asserted. It signals that this is the last data transfer of the transaction.
- **Target disconnect without data transfer**—STOP# and DEVSEL# asserted with TRDY# de-asserted after previous data transfers have been made. Indicates that no more data transfers will be made during this transaction.
- **Target abort**—STOP# asserted with DEVSEL# and TRDY# de-asserted.

Indicates that target will never be able to complete this transaction. DEVSEL# must be asserted for at least one cycle during the transaction before the target abort is signaled.

4.8.1 Master Termination Initiated by PI7C7100

PI7C7100, as an initiator, uses normal termination if DEVSEL# is returned by target within five clock cycles of PI7C7100's assertion of FRAME# on the target bus. As an initiator, PI7C7100 terminates a transaction when the following conditions are met:

- During a delayed write transaction, a single DWORD is delivered.
- During a non-prefetchable read transaction, a single DWORD is transferred from the target.
- During a prefetchable read transaction, a pre-fetch boundary is reached.
- For a posted write transaction, all write data for the transaction is transferred from data buffers to the target.
- For burst transfer, with the exception of "Memory Write and Invalidate" transactions, the master latency timer expires and the PI7C7100's bus grant is de-asserted.
- The target terminates the transaction with a retry, disconnect, or target abort.

If PI7C7100 is delivering posted write data when it terminates the transaction because the master latency timer expires, it initiates another transaction to deliver the remaining write data. The address of the transaction is updated to reflect the address of the current DWORD to be delivered.

If PI7C7100 is pre-fetching read data when it terminates the transaction because the master latency timer expires, it does not repeat the transaction to obtain more data.

4.8.2 Master Abort Received by PI7C7100

If the initiator initiates a transaction on the target bus and does not detect DEVSEL# returned by the target within five clock cycles of the assertion of FRAME#, PI7C7100 terminates the transaction with a master abort. This sets the received-master-abort bit in the status register corresponding to the target bus.

For delayed read and write transactions, PI7C7100 is able to reflect the master abort condition back to the initiator. When PI7C7100 detects a master abort in response to a delayed transaction, and when the initiator repeats the transaction, PI7C7100 does not respond to the transaction with DEVSEL# which induces the master abort condition back to the initiator. The transaction is then removed from the delayed transaction queue. When a master abort is received in response to a posted write transaction, PI7C7100 discards the posted write data and makes no more attempt to deliver the data. PI7C7100 sets the received-master-abort bit in the status register when the master abort is received on the primary bus, or it sets the received master abort bit in the secondary status register when the master abort is received on the secondary interface. When master abort is detected in posted write transaction with both master-abort-mode bit (bit 5 of bridge control register) and the SERR# enable bit (bit 8 of command register for secondary bus S1 or S2) are set, PI7C7100 asserts P_SERR# if the master-abort-on-posted-write is not set. The master-abort-on-posted-write bit is bit 4 of the P_SERR# event disable register (offset 64h).

Note: When PI7C7100 performs a Type 1 to special cycle conversion, a master abort is the expected termination for the special cycle on the target bus. In this case, the master abort received bit is not set, and the Type 1 configuration transaction is disconnected after the first data phase.

4.8.3 Target Termination Received by PI7C7100

When PI7C7100 initiates a transaction on the target bus and the target responds with DEVSEL#, the target can end the transaction with one of the following types of termination:

- Normal termination (upon de-assertion of FRAME#)
- Target retry
- Target disconnect
- Target abort

PI7C7100 handles these terminations in different ways, depending on the type of transaction being performed.

4.8.3.1 Delayed Write Target Termination Response

When PI7C7100 initiates a delayed write transaction, the type of target termination received from the target can be passed back to the initiator. Table 4–7 shows the response to each type of target termination that occurs during a delayed write transaction.

PI7C7100 repeats a delayed write transaction until one of the following conditions is met:

- PI7C7100 completes at least one data transfer.
- PI7C7100 receives a master abort.
- PI7C7100 receives a target abort.

PI7C7100 makes 2^{24}(default) or 2^{32}(maximum) write attempts resulting in a response of target retry.

Table 4-7. Posted Write Target Termination Response

Target Termination	Response
Normal	Returning disconnect to initiator with first data transfer only if multiple data phases requested.
Target retry	Returning target retry to initiator. Continue write attempts to target.
Target disconnect	Returning disconnect to initiator with first data transfer only if multiple data phases requested.
Target abort	Returning target abort to initiator.Set received target abort bit in target interface status register.Set signaled target abort bit in initiator interface status register.

After the PI7C7100 makes 2^{24}(default) attempts of the same delayed write transaction on the target bus, PI7C7100 asserts P_SERR# if the SERR# enable bit (bit 8 of command register for secondary bus S1 or S2) is set and the delayed-write-non-delivery bit is not set. The delayed-write-non-delivery bit is bit 5 of P_SERR# event disable register (offset 64h). PI7C7100 stops initiating transaction in response to that delayed write transaction. The delayed write request is discarded. Upon a subsequent write transaction attempt by the initiator, PI7C7100 returns a target abort. See Section 7.4 for a description of system error conditions.

4.8.3.2 Posted Write Target Termination Response

When PI7C7100 initiates a posted write transaction, the target termination cannot be passed back to the initiator. Table 4–8 shows the response to each type of target termination that occurs during a posted write transaction.

Table 4-8. Responses to Posted Write Target Termination

Target Termination	Response
Normal	No additional action.
Target retry	Repeating write transaction to target.
Target disconnect	Initiate write transaction for delivering remaining posted write data.
Target abort	Set received-target-abort bit in the target interface status register. Assert P_SERR# if enabled, and set the signaled-system-error bit in primary status register.

Note that when a target retry or target disconnect is returned and posted write data associated with that transaction remains in the write buffers, PI7C7100 initiates another write transaction to attempt to deliver the rest of the write data. If there is a target retry, the exact same address will be driven as for the initial write transaction attempt. If a target disconnect is received, the address that is driven on a subsequent write transaction attempt will be updated to reflect the address of the current DWORD. If the initial write transaction is Memory-Write-and-Invalidate transaction, and a partial delivery of write data to the target is performed before a target disconnect is received, PI7C7100 will use the memory write command to deliver the rest of the write data. It is because an incomplete cache line will be transferred in the subsequent write transaction attempt.

After the PI7C7100 makes 2^{24}(default) write transaction attempts and fails to deliver all posted write data associated with that transaction, PI7C7100 asserts P_SERR# if the primary SERR# enable bit is set (bit 8 of command register for secondary bus S1 or S2) and posted-write-non-delivery bit is not set. The posted-write-non-delivery bit is the bit 2 of P_SERR# event disable register (offset 64h). The write data is discarded. See Section 7.4 for a discussion of system error conditions.

4.8.3.3 Delayed Read Target Termination Response

When PI7C7100 initiates a delayed read transaction, the abnormal target responses can be passed back to the initiator. Other target responses depend on how much data the initiator requests. Table 4–9 shows the response to each type of target termination that occurs during a delayed read transaction.

PI7C7100 repeats a delayed read transaction until one of the following conditions is met:

- PI7C7100 completes at least one data transfer.
- PI7C7100 receives a master abort.
- PI7C7100 receives a target abort.
- PI7C7100 makes 2^{24}(default) read attempts resulting in a response of target retry.

Table 4-9. Responses to Delayed Read Target Termination

Target Termination	Response
Normal	If prefetchable, target disconnect only if initiator requests more data than read from target. If non-prefetchable, target disconnect on first data phase.
Target retry	Reinitiate read transaction to target.
Target disconnect	If initiator requests more data than read from target, return target disconnect to initiator.
Target abort	Return target abort to initiator. Set received target abort bit in the target interface status register. Set signaled target abort bit in the initiator interface status register.

After PI7C7100 makes 2^{24}(default) attempts of the same delayed read transaction on the target bus, PI7C7100 asserts P_SERR# if the primary SERR# enable bit is set (bit 8 of command register for secondary bus S1 or S2) and the delayed-write-non-delivery bit is not set. The delayed-write-non-delivery bit is bit 5 of P_SERR# event disable register (offset 64h). PI7C7100 stops initiating transactions in response to that delayed read transaction. The delayed read request is discarded. Upon a subsequent read transaction attempt by the initiator, PI7C7100 returns a target abort. See Section 7.4 for a description of system error conditions.

4.8.4 Target Termination Initiated by PI7C7100

PI7C7100 can return a target retry, target disconnect, or target abort to an initiator for reasons other than detection of that condition at the target interface.

4.8.4.1 Target Retry

PI7C7100 returns a target retry to the initiator when it cannot accept write data or return read data as a result of internal conditions. PI7C7100 returns a target retry to an initiator when any of the following conditions is met:

For delayed write transactions:

- The transaction is being entered into the delayed transaction queue.
- Transaction has already been entered into delayed transaction queue, but target response has not yet been received.
- Target response has been received but has not progressed to the head of the return queue.
- The delayed transaction queue is full, and the transaction cannot be queued.
- A transaction with the same address and command has been queued.
- A locked sequence is being propagated across PI7C7100, and the write transaction is not a locked transaction.
- The target bus is locked and the write transaction is a locked transaction.
- Use more than 16 clocks to accept this transaction.

For delayed read transactions:

- The transaction is being entered into the delayed transaction queue.
- The read request has already been queued, but read data is not yet available.
- Data has been read from target, but it is not yet at head of the read data queue, or a posted write transaction precedes it.
- The delayed transaction queue is full, and the transaction cannot be queued.
- A delayed read request with the same address and bus command has already been queued.
- A locked sequence is being propagated across PI7C7100, and the read transaction is not a locked transaction.
- PI7C7100 is currently discarding previously pre-fetched read data.
- The target bus is locked and the write transaction is a locked transaction.
- Use more than 16 clocks to accept this transaction.

For posted write transactions:

- The posted write data buffer does not have enough space for address and at least one DWORD of write data.
- A locked sequence is being propagated across PI7C7100, and the write transaction is not a locked transaction.

When a target retry is returned to the initiator of a delayed transaction, the initiator must repeat the transaction with the same address and bus command as well as the data if it is a write transaction, within the time frame specified by the master timeout value. Otherwise, the transaction is discarded from the buffers.

4.8.4.2 Target Disconnect

PI7C7100 returns a target disconnect to an initiator when one of the following conditions is met:

- PI7C7100 hits an internal address boundary.
- PI7C7100 cannot accept any more write data.
- PI7C7100 has no more read data to deliver.

See Section 4.5.4 for a description of write address boundaries, and Section 4.6.3 for a description of read address boundaries.

4.8.4.3 Target Abort

PI7C7100 returns a target abort to an initiator when one of the following conditions is met:

- PI7C7100 is returning a target abort from the intended target.
- PI7C7100 is unable to obtain delayed read data from the target or to deliver delayed write data to the target after 2^{24} (default) attempts.

When PI7C7100 returns a target abort to the initiator, it sets the signaled target abort bit in the status register corresponding to the initiator interface.

5. Address Decoding

PI7C7100 uses three address ranges that control I/O and memory transaction forwarding. These address ranges are defined by base and limit address registers in the configuration space. This chapter describes these address ranges, as well as ISA-mode and VGA-addressing support.

5.1 Address Ranges

PI7C7100 uses the following address ranges that determine which I/O and memory transactions are forwarded from the primary PCI bus to the secondary PCI bus, and from the secondary bus to the primary bus:

- Two 32-bit I/O address ranges
- Two 32-bit memory-mapped I/O (non-prefetchable memory) ranges
- Two 32-bit prefetchable memory address ranges

Transactions falling within these ranges are forwarded downstream from the primary PCI bus to the two secondary PCI buses. Transactions falling outside these ranges are forwarded upstream from the two secondary PCI buses to the primary PCI bus.

No address translation is required in PI7C7100. The addresses that are not marked for downstream are always forwarded upstream. However, if an address of a transaction initiated from S1 bus is located in the marked address range for downstream in S2 bus and not in the marked address range for downstream in S1 bus, the transaction will be forwarded to S2 bus instead of primary bus. By the same token, if an address of a transaction initiated from S2 bus is located in the marked address range for downstream in S1 bus and not in the marked address range for downstream in S2 bus, the transaction will be forwarded to S1 bus instead of primary bus.

5.2 I/O Address Decoding

PI7C7100 uses the following mechanisms that are defined in the configuration space to specify the I/O address space for downstream and upstream forwarding:

- I/O base and limit address registers
- The ISA enable bit
- The VGA mode bit
- The VGA snoop bit

This section provides information on the I/O address registers and ISA mode.

Section 5.4 provides information on the VGA modes.

To enable downstream forwarding of I/O transactions, the I/O enable bit must be set in the command register in configuration space. All I/O transactions initiated on the primary bus will be ignored if the I/O enable bit is not set. To enable upstream forwarding of I/O transactions, the master enable bit must be set in the command register. If the master-enable bit is not set, PI7C7100 ignores all I/O and memory transactions initiated on the secondary bus. The master-enable bit also allows upstream forwarding of memory transactions if it is set.

CAUTION

If any configuration state affecting I/O transaction forwarding is changed by a configuration write operation on the primary bus at the same time that I/O transactions are ongoing on the secondary bus, PI7C7100 response to the secondary bus I/O transactions is not predictable. Configure the I/O base and limit address registers, ISA enable bit, VGA mode bit, and VGA snoop bit before setting I/O enable and master enable bits, and change them subsequently only when the primary and secondary PCI buses are idle.

5.2.1 I/O Base and Limit Address Registers

PI7C7100 implements one set of I/O base and limit address registers in configuration space that define an I/O address range per port downstream forwarding. PI7C7100 supports 32-bit I/O addressing, which allows I/O addresses downstream of PI7C7100 to be mapped anywhere in a 4GB I/O address space.

I/O transactions with addresses that fall inside the range defined by the I/O base and limit registers are forwarded downstream from the primary PCI bus to the secondary PCI bus. I/O transactions with addresses that fall outside this range are forwarded upstream from the secondary PCI bus to the primary PCI bus.

The I/O range can be turned off by setting the I/O base address to a value greater than that of the I/O limit address. When the I/O range is turned off, all I/O transactions are forwarded upstream, and no I/O transactions are forwarded downstream. The I/O range has a minimum granularity of 4KB and is aligned on a 4KB boundary. The maximum I/O range is 4GB in size. The I/O base register consists of an 8-bit field at configuration address 1Ch, and a 16-bit field at address 30h. The top 4 bits of the 8-bit field define bits [15:12] of the I/O base address. The bottom 4 bits read only as 1h to indicate that PI7C7100 supports 32-bit I/O addressing. Bits [11:0] of the base address are assumed to be 0, which naturally aligns the base address to a 4KB boundary. The 16 bits contained in the I/O base upper 16 bits register at configuration offset 30h define AD[31:16] of the I/O base address. All 16 bits are read/write. After primary bus reset or chip reset, the value of the I/O base address is initialized to 0000 0000h.

The I/O limit register consists of an 8-bit field at configuration offset 1Dh and a 16-bit field at offset 32h. The top 4 bits of the 8-bit field define bits [15:12] of the I/O limit address. The bottom 4 bits read only as 1h to indicate that 32-bit I/O addressing is supported. Bits [11:0] of the limit address are assumed to be FFFh, which naturally aligns the limit address to the top of a 4KB I/O address block. The 16 bits contained in the I/O limit upper 16 bits register at configuration offset 32h define AD[31:16] of the I/O limit address. All 16 bits are read/write. After primary bus reset or chip reset, the value of the I/O limit address is reset to 0000 0FFFh.

Note: The initial states of the I/O base and I/O limit address registers define an I/O range of 0000 0000h to 0000 0FFFh, which is the bottom 4KB of I/O space. Write these registers with their appropriate values before setting either the I/O enable bit or the master enable bit in the command register in configuration space.

5.2.2 ISA Mode

PI7C7100 supports ISA mode by providing an ISA enable bit in the bridge control register in configuration space. ISA mode modifies the response of PI7C7100 inside the I/O address range in order to support mapping of I/O space in the presence of an ISA bus in the system. This bit only affects the response of PI7C7100 when the transaction falls inside the address range defined by the I/O base and limit address registers, and only when this address also falls inside the first 64KB of I/O space (address bits [31:16] are 0000h).

When the ISA enable bit is set, PI7C7100 does not forward downstream any I/O transactions addressing the top 768 bytes of each aligned 1KB block. Only those transactions addressing the bottom 256 bytes of an aligned 1KB block inside the base and limit I/O address range are forwarded downstream. Transactions above the 64KB I/O address boundary are forwarded as defined by the address range defined by the I/O base and limit registers.

Accordingly, if the ISA enable bit is set, PI7C7100 forwards upstream those I/O transactions addressing the top 768 bytes of each aligned 1KB block within the first 64KB of I/O space. The master enable bit in the command configuration register must also be set to enable upstream forwarding. All other I/O transactions initiated on the secondary bus are forwarded upstream only if they fall outside the I/O address range.

When the ISA enable bit is set, devices downstream of PI7C7100 can have I/O space mapped into the first 256 bytes of each 1KB chunk below the 64KB boundary, or anywhere in I/O space above the 64KB boundary.

5.3 Memory Address Decoding

PI7C7100 has three mechanisms for defining memory address ranges for forwarding of memory transactions:

- Memory-mapped I/O base and limit address registers
- Prefetchable memory base and limit address registers
- VGA mode

This section describes the first two mechanisms. Section 5.4.1 describes VGA mode.

To enable downstream forwarding of memory transactions, the memory enable bit must be set in the command register in configuration space. To enable upstream forwarding of memory transactions, the master-enable bit must be set in the command register. The master-enable bit also allows upstream forwarding of I/O transactions if it is set.

CAUTION

If any configuration state affecting memory transaction forwarding is changed by a configuration write operation on the primary bus at the same time that memory transactions are ongoing on the secondary bus, response to the secondary bus memory transactions is not predictable. Configure the memory-mapped I/O base and limit address registers, prefetchable memory base and limit address registers, and VGA mode bit before setting the memory enable and master enable bits, and change them subsequently only when the primary and secondary PCI buses are idle.

5.3.1 Memory-Mapped I/O Base and Limit Address Registers

Memory-mapped I/O is also referred to as non-prefetchable memory. Memory addresses that cannot automatically be pre-fetched but that can be conditionally pre-fetched based on command type should be mapped into this space. Read transactions to non-prefetchable space may exhibit side effects; this space may have non-memory-like behavior. PI7C7100 pre-fetches in this space only if the memory read line or memory read multiple commands are used; transactions using the memory read command are limited to a single data transfer.

The memory-mapped I/O base address and memory-mapped I/O limit address registers define an address range that PI7C7100 uses to determine when to forward memory commands. PI7C7100 forwards a memory transaction from the primary to the secondary interface if the transaction address falls within the memory-mapped I/O address range. PI7C7100 ignores memory transactions initiated on the secondary interface that fall into this address range. Any transactions that fall outside this address range are ignored on the primary interface and are forwarded upstream from the secondary interface (provided that they do not fall into the prefetchable memory range or are not forwarded downstream by the VGA mechanism).

The memory-mapped I/O range supports 32-bit addressing only. The PCI-to-PCI Bridge Architecture Specification does not provide for 64-bit addressing in the memory-mapped I/O space. The memory-mapped I/O address range has a granularity and alignment of 1MB. The maximum memory-mapped I/O address range is 4GB.

The memory-mapped I/O address range is defined by a 16-bit memory-mapped I/O base address register at configuration offset 20h and by a 16-bit memory-mapped I/O limit address register at offset 22h. The top 12 bits of each of these registers correspond to bits [31:20] of the memory address. The low 4 bits are hardwired to 0. The lowest 20 bits of the memory-mapped I/O base address are assumed to be 0 0000h, which results in a natural alignment to a 1MB boundary. The lowest 20 bits of the memory-mapped I/O limit address are assumed to be F FFFFh, which results in an alignment to the top of a 1MB block.

Note: The initial state of the memory-mapped I/O base address register is 0000 0000h. The initial state of the memory-mapped I/O limit address register is 000F FFFFh. Note that the initial states of these registers define a memory-mapped I/O range at the bottom 1MB block of memory. Write these registers with their appropriate values before setting either the memory enable bit or the master enable bit in the command register in configuration space.

To turn off the memory-mapped I/O address range, write the memory-mapped I/O base address register with a value greater than that of the memory-mapped I/O limit address register.

5.3.2 Prefetchable Memory Base and Limit Address Registers

Locations accessed in the prefetchable memory address range must have true memory-like behavior and must not exhibit side effects when read. This means that extra reads to a prefetchable memory location must have no side effects. PI7C7100 pre-fetches for all types of memory read commands in this address space.

The prefetchable memory base address and prefetchable memory limit address registers define an address range that PI7C7100 uses to determine when to forward memory commands. PI7C7100 forwards a memory transaction from the primary to the secondary interface if the transaction address falls within the prefetchable memory address range. PI7C7100 ignores memory transactions initiated on the secondary interface that fall into this address range. PI7C7100 does not respond to any transactions that fall outside this address range on the primary interface and forwards those transactions upstream from the secondary interface (provided that they do not fall into the memory-mapped I/O range or are not forwarded by the VGA mechanism).

The prefetchable memory range supports 64-bit addressing and provides additional registers to define the upper 32 bits of the memory address range, the prefetchable memory base address upper 32 bits register, and the prefetchable memory limit address upper 32 bits register. For address comparison, a single address cycle (32-bit address) prefetchable memory transaction is treated like a 64-bit address transaction where the upper 32 bits of the address are equal to 0. This upper

32-bit value of 0 is compared to the prefetchable memory base address upper 32 bits register and the prefetchable memory limit address upper 32 bits register. The prefetchable memory base address upper 32 bits register must be 0 to pass any single address cycle transactions downstream.

Prefetchable memory address range has a granularity and alignment of 1MB. Maximum memory address range is 4GB when 32-bit addressing is being used.

Prefetchable memory address range is defined by a 16-bit prefetchable memory base address register at configuration offset 24h and by a 16-bit prefetchable memory limit address register at offset 26h. The top 12 bits of each of these registers correspond to bits [31:20] of the memory address. The lowest 4 bits are hardwired to 1h. The lowest 20 bits of the prefetchable memory base address are assumed to be 0 0000h, which results in a natural alignment to a 1MB boundary. The lowest 20 bits of the prefetchable memory limit address are assumed to be FFFFFh, which results in an alignment to the top of a 1MB block.

Note: The initial state of the prefetchable memory base address register is 0000 0000h. The initial state of the prefetchable memory limit address register is 000F FFFFh. Note that the initial states of these registers define a prefetchable memory range at the bottom 1MB block of memory. Write these registers with their appropriate values before setting either the memory enable bit or the master enable bit in the command register in configuration space.

To turn off the prefetchable memory address range, write the prefetchable memory base address register with a value greater than that of the prefetchable memory limit address register. The entire base value must be greater than the entire limit value, meaning that the upper 32 bits must be considered. Therefore, to disable the address range, the upper 32 bits registers can both be set to the same value, while the lower base register is set greater than the lower limit register. Otherwise, the upper 32-bit base must be greater than the upper 32-bit limit.

5.4 VGA Support

PI7C7100 provides two modes for VGA support:

- VGA mode, supporting VGA-compatible addressing
- VGA snoop mode, supporting VGA palette forwarding

5.4.1 VGA Mode

When a VGA-compatible device exists downstream from PI7C7100, set the VGA mode bit in the bridge control register in configuration space to enable VGA mode. When PI7C7100 is operating in VGA mode, it forwards downstream those transactions addressing the VGA frame buffer memory and VGA I/O registers, regardless of the values of the base and limit address registers. PI7C7100 ignores transactions initiated on the secondary interface addressing these locations.

The VGA frame buffer consists of the following memory address range:

000A 0000h–000B FFFFh

Read transactions to frame buffer memory are treated as non-prefetchable. PI7C7100 requests only a single data transfer from the target, and read byte enable bits are forwarded to the target bus.

The VGA I/O addresses are in the range of 3B0h–3BBh and 3C0h–3DFh I/O. These I/O addresses are aliases every 1KB throughout the first 64KB of I/O space. This means that address bits <15:10> are not decoded and can be any value, while address bits [31:16] must be all 0s. VGA BIOS addresses starting at C0000h are not decoded in VGA mode.

5.4.2 VGA Snoop Mode

PI7C7100 provides VGA snoop mode, allowing for VGA palette write transactions to be forwarded downstream. This mode is used when a graphics device downstream from PI7C7100 needs to snoop or respond to VGA palette write transactions. To enable the mode, set the VGA snoop bit in the command register in configuration space. Note that PI7C7100 claims VGA palette write transactions by asserting DEVSEL# in VGA snoop mode.

When VGA snoop bit is set, PI7C7100 forwards downstream transactions within the 3C6h, 3C8h and 3C9h I/O addresses space. Note that these addresses are also forwarded as part of the VGA compatibility mode previously described. Again, address bits <15:10> are not decoded, while address bits <31:16> must be equal to 0, which means that these addresses are aliases every 1KB throughout the first 64KB of I/O space.

Note: If both the VGA mode bit and the VGA snoop bit are set, PI7C7100 behaves in the same way as if only the VGA mode bit were set.

6. Transaction Ordering

To maintain data coherency and consistency, PI7C7100 complies with the ordering rules set forth in the PCI Local Bus Specification, Revision 2.1, for transactions crossing the bridge. This chapter describes the ordering rules that control transaction forwarding across PI7C7100.

6.1 Transactions Governed by Ordering Rules

Ordering relationships are established for the following classes of transactions crossing PI7C7100:

- **Posted write transactions, comprised of memory write and memory write and invalidate transactions.** Posted write transactions complete at the source before they complete at the destination; that is, data is written into intermediate data buffers before it reaches the target.
- **Delayed write request transactions, comprised of I/O write and configuration write transactions.** Delayed write requests are terminated by target retry on the initiator bus and are queued in the delayed transaction queue. A delayed write transaction must complete on the target bus before it completes on the initiator bus.
- **Delayed write completion transactions, comprised of I/O write and configuration write transactions.** Delayed write completion transactions complete on the target bus, and the target response is queued in the buffers. A delayed write completion transaction proceeds in the direction opposite that of the original delayed write request; that is, a delayed write completion transaction proceeds from the target bus to the initiator bus.
- **Delayed read request transactions, comprised of all memory read, I/O read, and configuration read transactions.** Delayed read requests are terminated by target retry on the initiator bus and are queued in the delayed transaction queue.
- **Delayed read completion transactions, comprised of all memory read, I/O read, & configuration read transactions.** Delayed read completion transactions complete on the target bus, and the read data is queued in the read data buffers. A delayed read completion transaction proceeds in the direction opposite that of the original delayed read request; that is, a delayed read completion transaction proceeds from the target bus to the initiator bus.

PI7C7100 does not combine or merge write transactions:

- PI7C7100 does not combine separate write transactions into a single write transaction—this optimization is best implemented in the originating master.
- PI7C7100 does not merge bytes on separate masked write transactions to the same DWORD address—this optimization is also best implemented in the originating master.
- PI7C7100 does not collapse sequential write transactions to the same address into a single write transaction—the PCI Local Bus Specification does not permit this combining of transactions.

6.2 General Ordering Guidelines

Independent transactions on primary and secondary buses have a relationship only when those transactions cross PI7C7100.

The following general ordering guidelines govern transactions crossing PI7C7100:

- The ordering relationship of a transaction with respect to other transactions is determined when the transaction completes, that is, when a transaction ends with a termination other than target retry.
- Requests terminated with target retry can be accepted and completed in any order with respect to other transactions that have been terminated with target retry. If the order of completion of delayed requests is important, the initiator should not start a second delayed transaction until the first one has been completed. If more than one delayed transaction is initiated, the initiator should repeat all delayed transaction requests, using some fairness algorithm. Repeating a delayed transaction cannot be contingent on completion of another delayed transaction. Otherwise, a deadlock can occur.

- Write transactions flowing in one direction have no ordering requirements with respect to write transactions flowing in the other direction. PI7C7100 can accept posted write transactions on both interfaces at the same time, as well as initiate posted write transactions on both interfaces at the same time.
- The acceptance of a posted memory write transaction as a target can never be contingent on the completion of a non-locked, non-posted transaction as a master. This is true for PI7C7100 and must also be true for other bus agents. Otherwise, a deadlock can occur.
- PI7C7100 accepts posted write transactions, regardless of the state of completion of any delayed transactions being forwarded across PI7C7100.

6.3 Ordering Rules

Table 6–1 shows the ordering relationships of all the transactions and refers by number to the ordering rules that follow.

Table 6-1. Summary of Transaction Ordering

Pass	Posted Write	Delayed Read Request	Delayed Write Request	Delayed Read Completion	Delayed Write Completion
Posted write	N[1]	Y[5]	Y[5]	Y[5]	Y[5]
Delayed read request	N[2]	N	N	Y	Y
Delayed write request	N[4]	N	N	Y	Y
Delayed read completion	N[3]	Y	Y	N	N
Delayed write completion	Y	Y	Y	N	N

Note: The superscript accompanying some of the table entries refers to any applicable ordering rule listed in this section. Many entries are not governed by these ordering rules; therefore, the implementation can choose whether or not the transactions pass each other.

The entries without superscripts reflect the PI7C7100's implementation choices.

The following ordering rules describe the transaction relationships. Each ordering rule is followed by an explanation, and the ordering rules are referred to by number in Table 6–1. These ordering rules apply to posted write transactions, delayed write and read requests, and delayed write and read completion transactions crossing PI7C7100 in the same direction. Note that delayed completion transactions cross PI7C7100 in the direction opposite that of the corresponding delayed requests.

1. Posted write transactions must complete on the target bus in the order in which they were received on the initiator bus. The subsequent posted write transaction can be setting a flag that covers the data in the first posted write transaction; if the second transaction were to complete before the first transaction, a device checking the flag could subsequently consume stale data.

2. A delayed read request traveling in the same direction as a previously queued posted write transaction must push the posted write data ahead of it. The posted write transaction must complete on the target bus before the delayed read request can be attempted on the target bus. The read transaction can be to the same location as the write data, so if the read transaction were to pass the write transaction, it would return stale data.

3. A delayed read completion must ''pull'' ahead of previously queued posted write data traveling in the same direction. In this case, the read data is traveling in the same direction as the write data, and the initiator of the read transaction is on the same side of PI7C7100 as the target of the write transaction. The posted write transaction must complete to the target before the read data is returned to the initiator. The read transaction can be a reading to a status register of the initiator of the posted write data and therefore should not complete until the write transaction is complete.

4. Delayed write requests cannot pass previously queued posted write data. For posted memory write transactions, the delayed write transaction can set a flag that covers the data in the posted write transaction. If the delayed write request were to complete before the earlier posted write transaction, a device checking the flag could subsequently consume stale data.

5. Posted write transactions must be given opportunities to pass delayed read and write requests and completions. Otherwise, deadlocks may occur when some bridges which support delayed transactions and other bridges which do not support delayed transactions are being used in the same system. A fairness algorithm is used to arbitrate between the posted write queue and the delayed transaction queue.

6.4 Data Synchronization

Data synchronization refers to the relationship between interrupt signaling and data delivery. The PCI Local Bus Specification, Revision 2.1, provides the following alternative methods for synchronizing data and interrupts:

- The device signaling the interrupt performs a read of the data just written (software).
- The device driver performs a read operation to any register in the interrupting device before accessing data written by the device (software).
- System hardware guarantees that write buffers are flushed before interrupts are forwarded.

PI7C7100 does not have a hardware mechanism to guarantee data synchronization for posted write transactions. Therefore, all posted write transactions must be followed by a read operation, either from the device to the location just written (or some other location along the same path), or from the device driver to one of the device registers.

7. Error Handling

PI7C7100 checks, forwards, and generates parity on both the primary and secondary interfaces. To maintain transparency, PI7C7100 always 3es to forward the existing parity condition on one bus to the other bus, along with address and data. PI7C100 always attempts to be transparent when reporting errors, but this is not always possible, given the presence of posted data and delayed transactions.

To support error reporting on the PCI bus, PI7C7100 implements the following:
- PERR# and SERR# signals on both the primary and secondary interfaces
- Primary status and secondary status registers
- The device-specific P_SERR# event disable register

This chapter provides detailed information about how PI7C7100 handles errors. It also describes error status reporting and error operation disabling.

7.1 Address Parity Errors

PI7C7100 checks address parity for all transactions on both buses, for all address and all bus commands. When PI7C7100 detects an address parity error on the primary interface, the following events occur:

- If the parity error response bit is set in the command register, PI7C7100 does not claim the transaction with P_DEVSEL#; this may allow the transaction to terminate in a master abort.
  If parity error response bit is not set, PI7C7100 proceeds normally and accepts the transaction if it is directed to or across PI7C7100.
- PI7C7100 sets the detected parity error bit in the status register.
- PI7C7100 asserts P_SERR# and sets signaled system error bit in the status register, if both the following conditions are met:
  - The SERR# enable bit is set in the command register.
  - The parity error response bit is set in the command register.

When PI7C7100 detects an address parity error on the secondary interface, the following events occur:

- If the parity error response bit is set in the bridge control register, PI7C7100 does not claim the transaction with S1_DEVSEL# or S2_DEVSEL#; this may allow the transaction to terminate in a master abort. If parity error response bit is not set, PI7C7100 proceeds normally and accepts transaction if it is directed to or across PI7C7100.
- PI7C7100 sets the detected parity error bit in the secondary status register.
- PI7C7100 asserts P_SERR# and sets signaled system error bit in status register, if both of the following conditions are met:
  - The SERR# enable bit is set in the command register.
  - The parity error response bit is set in the bridge control register.

7.2 Data Parity Errors

When forwarding transactions, PI7C7100 attempts to pass the data parity condition from one interface to the other unchanged, whenever possible, to allow the master and target devices to handle the error condition.

The following sections describe, for each type of transaction, the sequence of events that occurs when a parity error is detected and the way in which the parity condition is forwarded across PI7C7100.

7.2.1 Configuration Write Transactions to Configuration Space

When PI7C7100 detects a data parity error during a Type 0 configuration write transaction to PI7C7100 configuration space, the following events occur:

- If the parity error response bit is set in the command register, PI7C7100 asserts P_TRDY# and writes the data to the configuration register. PI7C7100 also asserts P_PERR#. If the parity error response bit is not set, PI7C7100 does not assert P_PERR#.
- PI7C7100 sets the detected parity error bit in the status register, regardless of the state of the parity error response bit.

7.2.2 Read Transactions

When PI7C7100 detects a parity error during a read transaction, the target drives data and data parity, and the initiator checks parity and conditionally asserts PERR#.

For downstream transactions, when PI7C7100 detects a read data parity error on the secondary bus, the following events occur:

- PI7C7100 asserts S_PERR# two cycles following the data transfer, if the secondary interface parity error response bit is set in the bridge control register.
- PI7C7100 sets the detected parity error bit in the secondary status register.
- PI7C7100 sets the data parity detected bit in the secondary status register, if the secondary interface parity error response bit is set in the bridge control register.
- PI7C7100 forwards the bad parity with the data back to the initiator on the primary bus.
  If the data with the bad parity is pre-fetched and is not read by the initiator on the primary bus, the data is discarded and the data with bad parity is not returned to the initiator.
- PI7C7100 completes the transaction normally.

For upstream transactions, when PI7C7100 detects a read data parity error on the primary bus, the following events occur:

- PI7C7100 asserts P_PERR# two cycles following the data transfer, if the primary interface parity error response bit is set in the command register.
- PI7C7100 sets the detected parity error bit in the primary status register.
- PI7C7100 sets the data parity detected bit in the primary status register, if the primary interface parity-error-response bit is set in the command register.
- PI7C7100 forwards the bad parity with the data back to the initiator on the secondary bus.
  If the data with the bad parity is pre-fetched and is not read by the initiator on the secondary bus, the data is discarded and the data with bad parity is not returned to the initiator.
- PI7C7100 completes the transaction normally.
  PI7C7100 returns to the initiator the data and parity that was received from the target. When the initiator detects a parity error on this read data and is enabled to report it, the initiator asserts PERR# two cycles after the data transfer occurs. It is assumed that the initiator takes responsibility for handling a parity error condition; therefore, when PI7C7100 detects PERR# asserted while returning read data to the initiator, PI7C7100 does not take any further action and completes the transaction normally.

7.2.3 Delayed Write Transactions

When PI7C7100 detects a data parity error during a delayed write transaction, the initiator drives data and data parity, and the target checks parity and conditionally asserts PERR#.

For delayed write transactions, a parity error can occur at the following times:

- During the original delayed write request transaction
- When the initiator repeats the delayed write request transaction
- When PI7C7100 completes the delayed write transaction to the target

When a delayed write transaction is normally queued, the address, command, address parity, data, byte enable bits, and data parity are all captured and a target retry is returned to the initiator. When PI7C7100 detects a parity error on the write data for the initial delayed write request transaction, the following events occur:

- If the parity-error-response bit corresponding to the initiator bus is set, PI7C7100 asserts TRDY# to the initiator and the transaction is not queued. If multiple data phases are requested, STOP# is also asserted to cause a target disconnect. Two cycles after the data transfer, PI7C7100 also asserts PERR#.

  If the parity-error-response bit is not set, PI7C7100 returns a target retry. It queues the transaction as usual. PI7C7100 does not assert PERR#. In this case, the initiator repeats the transaction.
- PI7C7100 sets the detected-parity-error bit in the status register corresponding to the initiator bus, regardless of the state of the parity-error-response bit.

Note: If parity checking is turned off and data parity errors have occurred for queued or subsequent delayed write transactions on the initiator bus, it is possible that the initiator's re-attempts of the write transaction may not match the original queued delayed write information contained in the delayed transaction queue. In this case, a master timeout condition may occur, possibly resulting in a system error (P_SERR# assertion).

For downstream transactions, when PI7C7100 is delivering data to the target on the secondary bus and S_PERR# is asserted by the target, the following events occur:

- PI7C7100 sets the secondary interface data parity detected bit in the secondary status register, if the secondary parity error response bit is set in the bridge control register.
- PI7C7100 captures the parity error condition to forward it back to the initiator on the primary bus.

Similarly, for upstream transactions, when PI7C7100 is delivering data to the target on the primary bus and P_PERR# is asserted by the target, the following events occur:

- PI7C7100 sets the primary interface data-parity-detected bit in the status register, if the primary parity-error-response bit is set in the command register.
- PI7C7100 captures the parity error condition to forward it back to the initiator on the secondary bus.

A delayed write transaction is completed on the initiator bus when the initiator repeats the write transaction with the same address, command, data, and byte enable bits as the delayed write command that is at the head of the posted data queue. Note that the parity bit is not compared when determining whether the transaction matches those in the delayed transaction queues.

Two cases must be considered:

- When parity error is detected on the initiator bus on a subsequent re-attempt of the transaction and was not detected on the target bus
- When parity error is forwarded back from the target bus

For downstream delayed write transactions, when the parity error is detected on the initiator bus and PI7C7100 has write status to return, the following events occur:

- PI7C7100 first asserts P_TRDY# and then asserts P_PERR# two cycles later, if the primary interface parity-error-response bit is set in the command register.
- PI7C7100 sets the primary interface parity-error-detected bit in the status register.
- Because there was not an exact data and parity match, the write status is not returned and the transaction remains in the queue.

Similarly, for upstream delayed write transactions, when the parity error is detected on the initiator bus and PI7C7100 has write status to return, the following events occur:

- PI7C7100 first asserts S1_TRDY# or S2_TRDY# and then asserts S_PERR# two cycles later, if the secondary interface parity-error-response bit is set in the bridge control register (offset 3Ch).
- PI7C7100 sets the secondary interface parity-error-detected bit in the secondary status register.
- Because there was not an exact data and parity match, the write status is not returned and the transaction remains in the queue.

For downstream transactions, where the parity error is being passed back from the target bus and the parity error condition was not originally detected on the initiator bus, the following events occur:

- PI7C7100 asserts P_PERR# two cycles after the data transfer, if the following are both true:
  - The parity-error-response bit is set in the command register of the primary interface.
  - The parity-error-response bit is set in the bridge control register of the secondary interface.
- PI7C7100 completes the transaction normally.

For upstream transactions, when the parity error is being passed back from the target bus and the parity error condition was not originally detected on the initiator bus, the following events occur:

- PI7C7100 asserts S_PERR# two cycles after the data transfer, if the following are both true:
  - The parity error response bit is set in the command register of the primary interface.
  - The parity error response bit is set in the bridge control register of the secondary interface.
- PI7C7100 completes the transaction normally.

7.2.4 Posted Write Transactions

During downstream posted write transactions, when PI7C7100 responds as a target, it detects a data parity error on the initiator (primary) bus, the following events occur:

- PI7C7100 asserts P_PERR# two cycles after the data transfer, if the parity error response bit is set in the command register of primary interface.
- PI7C7100 sets the parity error detected bit in the status register of the primary interface.
- PI7C7100 captures and forwards the bad parity condition to the secondary bus.
- PI7C7100 completes the transaction normally.

Similarly, during upstream posted write transactions, when PI7C7100 responds as a target, it detects a data parity error on the initiator (secondary) bus, the following events occur:

- PI7C7100 asserts S_PERR# two cycles after the data transfer, if the parity error response bit is set in the bridge control register of the secondary interface.
- PI7C7100 sets the parity error detected bit in the status register of the secondary interface.
- PI7C7100 captures and forwards the bad parity condition to the primary bus.
- PI7C7100 completes the transaction normally.

During downstream write transactions, when a data parity error is reported on the target (secondary) bus by the target's assertion of S_PERR#, the following events occur:

- PI7C7100 sets the data parity detected bit in the status register of secondary interface, if the parity error response bit is set in the bridge control register of the secondary interface.
- PI7C7100 asserts P_SERR# and sets the signaled system error bit in the status register, if all the following conditions are met:
  - The SERR# enable bit is set in the command register.
  - The posted write parity error bit of P_SERR# event disable register is not set.
  - The parity error response bit is set in the bridge control register of the secondary interface.
  - The parity error response bit is set in the command register of the primary interface.
  - PI7C7100 has not detected the parity error on the primary (initiator) bus which the parity error is not forwarded from the primary bus to the secondary bus.

During upstream write transactions, when a data parity error is reported on the target (primary) bus by the target's assertion of P_PERR#, the following events occur:

- PI7C7100 sets the data parity detected bit in the status register, if the parity error response bit is set in the command register of the primary interface.
- PI7C7100 asserts P_SERR# and sets the signaled system error bit in the status register, if all the following conditions are met:
  - The SERR# enable bit is set in the command register.
  - The parity error response bit is set in the bridge control register of the secondary interface.
  - The parity error response bit is set in the command register of the primary interface.
  - PI7C7100 has not detected the parity error on the secondary (initiator) bus which the parity error is not forwarded from the secondary bus to the primary bus.

Assertion of P_SERR# is used to signal the parity error condition when the initiator does not know that the error occurred.

Because the data has already been delivered with no errors, there is no other way to signal this information back to the initiator.

If the parity error has forwarded from the initiating bus to the target bus, P_SERR# will not be asserted.

7.3 Data Parity Error Reporting Summary

In the previous sections, the responses of PI7C7100 to data parity errors are presented according to the type of transaction in progress. This section organizes the responses of PI7C7100 to data parity errors according to the status bits that PI7C7100 sets and the signals that it asserts.

Table 7–1 shows setting the detected parity error bit in the status register, corresponding to the primary interface. This bit is set when PI7C7100 detects a parity error on the primary interface.

Table 7–1 Setting the Primary Interface Detected Parity Error Bit

Primary detected parity error bit	Transaction Type	Direction	Bus where error was detected	Primary/Secondary parity error response bits
0	Read	Downstream	Primary	x/x[1]
0	Read	Downstream	Secondary	x/x
1	Read	Upstream	Primary	x/x
0	Read	Upstream	Secondary	x/x
1	Posted write	Downstream	Primary	x/x
0	Posted write	Downstream	Secondary	x/x
0	Posted write	Upstream	Primary	x/x
0	Posted write	Upstream	Secondary	x/x
1	Delayed write	Downstream	Primary	x/x
0	Delayed write	Downstream	Secondary	x/x
0	Delayed write	Upstream	Primary	x/x
0	Delayed write	Upstream	Secondary	x/x

[1]x =don’t care

[1]x =don’t care

Table 7–2 shows setting the detected parity error bit in the secondary status register, corresponding to the secondary interface. This bit is set when PI7C7100 detects a parity error on the secondary interface.

Table 7–2. Setting Secondary Interface Detected Parity Error Bit

Secondary detected parity error bit	Transaction Type	Direction	Bus where error was detected	Primary/Secondary parity error response bits
0	Read	Downstream	Primary	x/x[1]
1	Read	Downstream	Secondary	x/x
0	Read	Upstream	Primary	x/x
0	Read	Upstream	Secondary	x/x
0	Posted write	Downstream	Primary	x/x
0	Posted write	Downstream	Secondary	x/x
0	Posted write	Upstream	Primary	x/x
1	Posted write	Upstream	Secondary	x/x
0	Delayed write	Downstream	Primary	x/x
0	Delayed write	Downstream	Secondary	x/x
0	Delayed write	Upstream	Primary	x/x
1	Delayed write	Upstream	Secondary	x/x

Table 7–3 shows setting data parity detected bit in the primary interface's status register. This bit is set under the following conditions:

- PI7C7100 must be a master on the primary bus.
- The parity error response bit in the command register, corresponding to the primary interface, must be set.
- The P_PERR# signal is detected asserted or a parity error is detected on the primary bus.

Table 7–3. Setting Primary Interface Data Parity Detected Bit

Primary data parity bit	Transaction Type	Direction	Bus where error was detected	Primary/Secondary parity error response bits
0	Read	Downstream	Primary	x/x[1]
0	Read	Downstream	Secondary	x/x
1	Read	Upstream	Primary	1/x
0	Read	Upstream	Secondary	x/x
0	Posted write	Downstream	Primary	x/x
0	Posted write	Downstream	Secondary	x/x
1	Posted write	Upstream	Primary	1/x
0	Posted write	Upstream	Secondary	x/x
0	Delayed write	Downstream	Primary	x/x
0	Delayed write	Downstream	Secondary	x/x
1	Delayed write	Upstream	Primary	1/x
0	Delayed write	Upstream	Secondary	x/x

[1]x =don't care

Table 7–4 shows setting the data parity detected bit in the status register of secondary interface. This bit is set under the following conditions:

- The PI7C7100 must be a master on the secondary bus.
- The parity error response bit must be set in the bridge control register of secondary interface.
- The S_PERR# signal is detected asserted or a parity error is detected on the secondary bus.

Table 7–4. Setting Secondary Interface Data Parity Detected Bit

Secondary data parity detected bit	Transaction Type	Direction	Bus where error was detected	Primary/Secondary parity error response bits
0	Read	Downstream	Primary	x/x[1]
1	Read	Downstream	Secondary	x/1
0	Read	Upstream	Primary	x/x
0	Read	Upstream	Secondary	x/x
0	Posted write	Downstream	Primary	x/x
1	Posted write	Downstream	Secondary	x/1
0	Posted write	Upstream	Primary	x/x
0	Posted write	Upstream	Secondary	x/x
0	Delayed write	Downstream	Primary	x/x
1	Delayed write	Downstream	Secondary	x/1
0	Delayed write	Upstream	Primary	x/x
0	Delayed write	Upstream	Secondary	x/x

[1]x =don't care

Table 7–5 shows assertion of P_PERR#. This signal is set under the following conditions:

- PI7C7100 is either the target of a write transaction or the initiator of a read transaction on the primary bus.
- The parity-error-response bit must be set in the command register of primary interface.
- PI7C7100 detects a data parity error on the primary bus or detects S_PERR# asserted during the completion phase of a downstream delayed write transaction on the target (secondary) bus.

Table 7–5. Assertion of P_PERR#

P_PERR#	Transaction Type	Direction	Bus where error was detected	Primary/Secondary parity error response bits
1 (de-asserted)	Read	Downstream	Primary	x/x[1]
1	Read	Downstream	Secondary	x/x
0 (asserted)	Read	Upstream	Primary	1/x
1	Read	Upstream	Secondary	x/x
0	Posted write	Downstream	Primary	1/x
1	Posted write	Downstream	Secondary	x/x
1	Posted write	Upstream	Primary	x/x
1	Posted write	Upstream	Secondary	x/x
0	Delayed write	Downstream	Primary	1/x
0[2]	Delayed write	Downstream	Secondary	1/1
1	Delayed write	Upstream	Primary	x/x
1	Delayed write	Upstream	Secondary	x/x

[1]x =don't care
[2]The parity error was detected on the target (secondary) bus but not on the initiator (primary) bus.

Table 7–6 shows assertion of S_PERR# that is set under the following conditions:

- PI7C7100 is either the target of a write transaction or the initiator of a read transaction on the secondary bus.
- The parity error response bit must be set in the bridge control register of secondary interface.
- PI7C7100 detects a data parity error on the secondary bus or detects P_PERR# asserted during the completion phase of an upstream delayed write transaction on the target (primary) bus.

Table 7–6. Assertion of S_PERR#

S_PERR#	Transaction Type	Direction	Bus where error was detected	Primary/Secondary parity error response bits
1 (de-asserted)	Read	Downstream	Primary	x/x[1]
0 (asserted)	Read	Downstream	Secondary	x/1
1	Read	Upstream	Primary	x/x
1	Read	Upstream	Secondary	x/x
1	Posted write	Downstream	Primary	x/x
1	Posted write	Downstream	Secondary	x/x
1	Posted write	Upstream	Primary	x/x
0	Posted write	Upstream	Secondary	x/1
1	Delayed write	Downstream	Primary	x/x
1	Delayed write	Downstream	Secondary	x/x
0[2]	Delayed write	Upstream	Primary	1/1
0	Delayed write	Upstream	Secondary	x/1

[1]x =don't care
[2]The parity error was detected on the target (secondary) bus but not on the initiator (primary) bus.

Table 7–7 shows assertion of P_SERR#. This signal is set under the following conditions:

- PI7C7100 has detected P_PERR# asserted on an upstream posted write transaction or S_PERR# asserted on a downstream posted write transaction.
- PI7C7100 did not detect the parity error as a target of the posted write transaction.
- The parity error response bit on the command register and the parity error response bit on the bridge control register must both be set.
- The SERR# enable bit must be set in the command register.

Table 7–7. Assertion of P_SERR# for Data Parity Errors

P_SERR#	Transaction Type	Direction	Bus where error was detected	Primary/Secondary parity error response bits
1 (de-asserted)	Read	Downstream	Primary	x/x[1]
1	Read	Downstream	Secondary	x/x
1	Read	Upstream	Primary	x/x
1	Read	Upstream	Secondary	x/x
1	Posted write	Downstream	Primary	x/x
0[2] (asserted)	Posted write	Downstream	Secondary	1/1
0[3]	Posted write	Upstream	Primary	1/1
1	Posted write	Upstream	Secondary	x/x
1	Delayed write	Downstream	Primary	x/x
1	Delayed write	Downstream	Secondary	x/x
1	Delayed write	Upstream	Primary	x/x
1	Delayed write	Upstream	Secondary	x/x

[1]x =don't care
[2]The parity error was detected on the target (secondary) bus but not on the initiator (primary) bus.
[3]The parity error was detected on the target (primary) bus but not on the initiator (secondary) bus.

7.4 System Error (SERR#) Reporting

PI7C7100 uses the P_SERR# signal to report conditionally a number of system error conditions in addition to the special case parity error conditions described in Section 7.2.3.

Whenever assertion of P_SERR# is discussed in this document, it is assumed that the following conditions apply:

- For PI7C7100 to assert P_SERR# for any reason, the SERR# enable bit must be set in the command register.
- Whenever PI7C7100 asserts P_SERR#, PI7C7100 must also set the signaled system error bit in the status register.

In compliance with the PCI-to-PCI Bridge Architecture Specification, PI7C7100 asserts P_SERR# when it detects the secondary SERR# input, S_SERR#, asserted and the SERR# forward enable bit is set in the bridge control register. In addition, PI7C7100 also sets the received system error bit in the secondary status register.

PI7C7100 also conditionally asserts P_SERR# for any of the following reasons:

- Target abort detected during posted write transaction
- Master abort detected during posted write transaction
- Posted write data discarded after 2^{24}(default) attempts to deliver (2^{24} target re3es received)
- Parity error reported on target bus during posted write transaction (see previous section)
- Delayed write data discarded after 2^{24}(default) attempts to deliver (2^{24} target re3es received)
- Delayed read data cannot be transferred from target after 2^{24}(default) attempts (2^{24} target re3es received)
- Master timeout on delayed transaction

The device-specific P_SERR# status register reports the reason for the assertion of P_SERR#.

Most of these events have additional device-specific disable bits in the P_SERR# event disable register that make it possible to mask out P_SERR# assertion for specific events. The master timeout condition has a SERR# enable bit for that event in the bridge control register and therefore does not have a device-specific disable bit.

8. Exclusive Access

This chapter describes the use of the LOCK# signal to implement exclusive access to a target for transactions that cross PI7C7100.

8.1 Concurrent Locks

The primary and secondary bus lock mechanisms operate concurrently except when a locked transaction crosses PI7C7100. A primary master can lock a primary target without affecting the status of the lock on the secondary bus, and vice versa. This means that a primary master can lock a primary target at the same time that a secondary master locks a secondary target.

8.2 Acquiring Exclusive Access across PI7C7100

For any PCI bus, before acquiring access to the LOCK# signal and starting a series of locked transactions, the initiator must first check that both of the following conditions are met:

- The PCI bus must be idle.
- The LOCK# signal must be de-asserted.

The initiator leaves the LOCK# signal de-asserted during the address phase and asserts LOCK# one clock cycle later. Once a data transfer is completed from the target, the target lock has been achieved.

Locked transactions can cross PI7C7100 in the downstream and upstream directions, from the primary bus to the secondary bus and vice versa.

When the target resides on another PCI bus, the master must acquire not only the lock on its own PCI bus but also the lock on every bus between its bus and the target's bus. When PI7C7100 detects on the primary bus, an initial locked transaction intended for a target on the secondary bus, PI7C7100 samples the address, transaction type, byte enable bits, and parity, as described in Section 4.6.4. It also samples the lock signal. If there is a lock established between 2 ports or the target bus is already locked by another master, then the current lock cycle is re3ed without forward. Because a target retry is signaled to the initiator, the initiator must relinquish the lock on the primary bus, and therefore the lock is not yet established.

The first locked transaction must be a read transaction. Subsequent locked transactions can be read or write transactions. Posted memory write transactions that are a part of the locked transaction sequence are still posted. Memory read transactions that are a part of the locked transaction sequence are not pre-fetched.

When the locked delayed read request is queued, PI7C7100 does not queue any more transactions until the locked sequence is finished. PI7C7100 signals a target retry to all transactions initiated subsequent to the locked read transaction that are intended for targets on the other side of PI7C7100. PI7C7100 allows any transactions queued before the locked transaction to complete before initiating the locked transaction.

When the locked delayed read request transaction moves to the head of the delayed transaction queue, PI7C7100 initiates the transaction as a locked read transaction by de-asserting LOCK# on the target bus during the first address phase, and by asserting LOCK# one cycle later. If LOCK# is already asserted (used by another initiator), PI7C7100 waits to request access to the secondary bus until LOCK# is de-asserted when the target bus is idle. Note that the existing lock on the target bus could not have crossed PI7C7100. Otherwise, the pending queued locked transaction would not have been queued. When PI7C7100 is able to complete a data transfer with the locked read transaction, the lock is established on the secondary bus.

When the initiator repeats the locked read transaction on the primary bus with the same address, transaction type, and byte enable bits, PI7C7100 transfers the read data back to the initiator, and the lock is then also established on the primary bus.

For PI7C7100 to recognize and respond to the initiator, the initiator's subsequent attempts of the read transaction must use the locked transaction sequence (de-assert LOCK# during address phase, and assert LOCK# one cycle later). If the LOCK# sequence is not used in subsequent attempts, a master timeout condition may result. When a master timeout condition occurs, SERR# is conditionally asserted (see Section 7.4), the read data and queued read transaction are discarded, and the LOCK# signal is de-asserted on the target bus.

Once the intended target has been locked, any subsequent locked transactions initiated on the initiator bus that are forwarded by PI7C7100 are driven as locked transactions on the target bus.

When PI7C7100 receives a target abort or a master abort in response to the delayed locked read transaction, this status is passed back to the initiator, and no locks are established on either the target or the initiator bus. PI7C7100 resumes forwarding unlocked transactions in both directions.

8.3 Ending Exclusive Access

After the lock has been acquired on both initiator and target buses, PI7C7100 must maintain the lock on the target bus for any subsequent locked transactions until the initiator relinquishes the lock.

The only time a target-retry causes the lock to be relinquished is on the first transaction of a locked sequence. On subsequent transactions in the sequence, the target retry has no effect on the status of the lock signal.

An established target lock is maintained until the initiator relinquishes the lock. PI7C7100 does not know whether the current transaction is the last one in a sequence of locked transactions until the initiator de-asserts the LOCK# signal at end of the transaction.

When the last locked transaction is a delayed transaction, PI7C7100 has already completed the transaction on the secondary bus. In this example, as soon as PI7C7100 detects that the initiator has relinquished the LOCK# signal by sampling it in the de-asserted state while FRAME# is de-asserted, PI7C7100 de-asserts the LOCK# signal on the target bus as soon as possible. Because of this behavior, LOCK# may not be de-asserted until several cycles after the last locked transaction has been completed on the target bus. As soon as PI7C7100 has de-asserted LOCK# to indicate the end of a sequence of locked transactions, it resumes forwarding unlocked transactions.

When the last locked transaction is a posted write transaction, PI7C7100 de-asserts LOCK# on the target bus at the end of the transaction because the lock was relinquished at the end of the write transaction on the initiator bus.

When PI7C7100 receives a target abort or a master abort in response to a locked delayed transaction, PI7C7100 returns a target abort or a master abort when the initiator repeats the locked transaction. The initiator must then de-assert LOCK# at the end of the transaction. PI7C7100 sets the appropriate status bits, flagging the abnormal target termination condition (see Section 4.8). Normal forwarding of unlocked posted and delayed transactions is resumed.

When PI7C7100 receives a target abort or a master abort in response to a locked posted write transaction, PI7C7100 cannot pass back that status to the initiator. PI7C7100 asserts SERR# on the initiator bus when a target abort or a master abort is received during a locked posted write transaction, if the SERR# enable bit is set in the command register. Signal SERR# is asserted for the master abort condition if the master abort mode bit is set in the bridge control register (see Section 7.4).

9. PCI Bus Arbitration

PI7C7100 must arbitrate for use of the primary bus when forwarding upstream transactions. Also, it must arbitrate for use of the secondary bus when forwarding downstream transactions. The arbiter for the primary bus resides external to PI7C7100, typically on the motherboard. For the secondary PCI bus, PI7C7100 implements an internal arbiter. This arbiter can be disabled, and an external arbiter can be used instead. This chapter describes primary and secondary bus arbitration.

9.1 Primary PCI Bus Arbitration

PI7C7100 implements a request output pin, P_REQ#, and a grant input pin, P_GNT#, for primary PCI bus arbitration. PI7C7100 asserts P_REQ# when forwarding transactions upstream; that is, it acts as initiator on the primary PCI bus. As long as at least one pending transaction resides in the queues in the upstream direction, either posted write data or delayed transaction requests, PI7C7100 keeps P_REQ# asserted. However, if a target retry, target disconnect, or a target abort is received in response to a transaction initiated by PI7C7100 on the primary PCI bus, PI7C7100 de-asserts P_REQ# for two PCI clock cycles.

For all cycles through the bridge, P_REQ# is not asserted until the transaction request has been completely queued.

When P_GNT# is asserted LOW by the primary bus arbiter after PI7C7100 has asserted P_REQ#, PI7C7100 initiates a transaction on the primary bus during the next PCI clock cycle. When P_GNT# is asserted to PI7C7100 when P_REQ# is not asserted, PI7C7100 parks P_AD, P_CBE, and P_PAR by driving them to valid logic levels. When the primary bus is parked at PI7C7100 and PI7C7100 has a transaction to initiate on the primary bus, PI7C7100 starts the transaction if P_GNT# was asserted during the previous cycle.

9.2 Secondary PCI Bus Arbitration

PI7C7100 implements an internal secondary PCI bus arbiter. This arbiter supports two sets of eight external masters in addition to PI7C7100. The internal arbiter can be disabled, and an external arbiter can be used instead for secondary bus arbitration.

9.2.1 Secondary Bus Arbitration Using the Internal Arbiter

To use the internal arbiter, the secondary bus arbiter enable pin, S_CFN#, must be tied LOW. PI7C7100 has two sets of eight secondary bus request input pins, S1_REQ#[7:0], S2_REQ#[7:0], and two sets of eight secondary bus output grant pins, S1_GNT#[7:0], S2_GNT#[7:0], to support external secondary bus masters. The secondary bus request and grant signals are connected internally to the arbiter and are not brought out to external pins when S_CFN# is HIGH.

The secondary arbiter supports a 2-sets programmable 2-level rotating algorithm with each set taking care of 8 requests/ grants. Each set of masters can be assigned to a high priority group and a low priority group. The low priority group as a whole represents one entry in the high priority group; that is, if the high priority group consists of n masters, then in at least every n+1 transactions the highest priority is assigned to the low priority group. Priority rotates evenly among the low priority group. Therefore, members of the high priority group can be serviced n transactions out of n+1, while one member of the low priority group is serviced once every n+1 transactions. Figure 9–1 shows an example of an internal arbiter where four masters, including PI7C7100, are in the high priority group, and five masters are in the low priority group. Using this example, if all requests are always asserted, the highest priority rotates among the masters in the following fashion (high priority members are given in italics, low priority members, in boldface type):
B, m0, m1, *m2*, **m3**, *B, m0, m1, m2*, **m4**, *B, m0, m1, m2*, **m5**, *B, m0, m1, m2*, **m6**, *B, m0, m1, m2*, **m7** and so on.

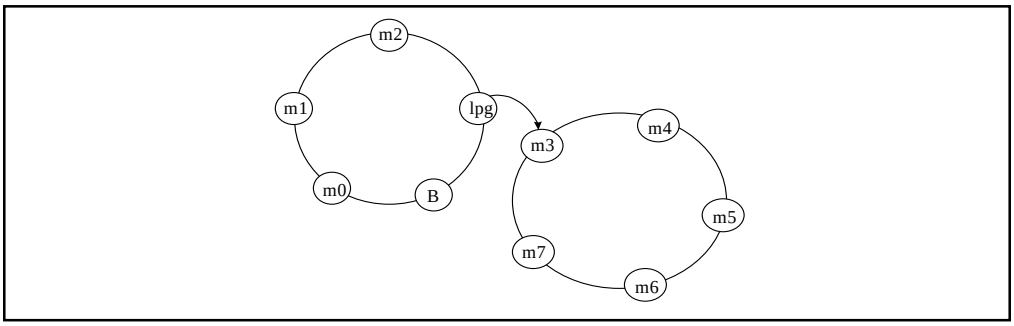


Figure 9-1. Secondary Arbiter Example

Each bus master, including PI7C7100, can be configured to be in either the low priority group or the high priority group by setting the corresponding priority bit in the arbiter-control register. The arbiter-control register is located at offset 40h. Each master has a corresponding bit. If the bit is set to 1, the master is assigned to the high priority group. If the bit is set to 0, the master is assigned to the low priority group. If all the masters are assigned to one group, the algorithm defaults to a straight rotating priority among all the masters. After reset, all external masters are assigned to the low priority group, and PI7C7100 is assigned to the high priority group. PI7C7100 receives highest priority on the target bus every other transaction, and priority rotates evenly among the other masters.

Priorities are re-evaluated every time S1_FRAME# or S2_FRAME# is asserted at the start of each new transaction on the secondary PCI bus. From this point until the time that the next transaction starts, the arbiter asserts the grant signal corresponding to the highest priority request that is asserted. If a grant for a particular request is asserted, and a higher priority request subsequently asserts, the arbiter de-asserts the asserted grant signal and asserts the grant corresponding to the new higher priority request on the next PCI clock cycle. When priorities are re-evaluated, the highest priority is assigned to the next highest priority master relative to the master that initiated the previous transaction. The master that initiated the last transaction now has the lowest priority in its group.

If PI7C7100 detects that an initiator has failed to assert S1_FRAME# or S2_FRAME# after 16 cycles of both grant assertion and a secondary idle bus condition, the arbiter de-asserts the grant. That master does not receive any more grants until it de-asserts its request for at least one PCI clock cycle.

To prevent bus contention, if the secondary PCI bus is idle, the arbiter never asserts one grant signal in the same PCI cycle in which it de-asserts another. It de-asserts one grant and asserts the next grant, no earlier than one PCI clock cycle later. If the secondary PCI bus is busy, that is, either S1_FRAME# (S2_FRAME#) or S1_IRDY# (S2_IRDY#) is asserted, the arbiter can de-assert one grant and assert another grant during the same PCI clock cycle.

9.2.2 Secondary Bus Arbitration Using an External Arbiter

The internal arbiter is disabled when the secondary bus central function control pin, S_CFN#, is tied high. An external arbiter must then be used.

When S_CFN# is tied high, PI7C7100 reconfigures four pins (two per port) to be external request and grant pins. The S1_GNT#[0] and S2_GNT#[0] pins are reconfigured to be the external request pins because they are output. The S1_REQ#[0] and S2_REQ#[0] pins are reconfigured to be the external grant pins because they are input. When an external arbiter is used, PI7C7100 uses the S1_GNT#[0] or S2_GNT#[0] pin to request the secondary bus. When the reconfigured S1_REQ#[0] or S2_REQ#[0] pin is asserted low after PI7C7100 has asserted S1_GNT#[0] or S2_GNT#[0]. PI7C7100 initiates a transaction on the secondary bus one cycle later. If grant is asserted and PI7C7100 has not asserted the request, PI7C7100 parks AD, CBE and PAR pins by driving them to valid logic levels.

The unused secondary bus grant outputs, S1_GNT#[7:1] and S2_GNT#[7:1] are driven high. The unused secondary bus request inputs, S1_REQ#[7:1] and S2_REQ#[7:1], should be pulled high.

9.2.3 Bus Parking

Bus parking refers to driving the AD[31:0], CBE[3:0]#, and PAR lines to a known value while the bus is idle. In general, the device implementing the bus arbiter is responsible for parking the bus or assigning another device to park the bus. A device parks the bus when the bus is idle, its bus grant is asserted, and the device's request is not asserted. The AD and CBE signals should be driven first, with the PAR signal driven one cycle later.

PI7C7100 parks the primary bus only when P_GNT# is asserted, P_REQ# is de-asserted, and the primary PCI bus is idle. When P_GNT# is de-asserted, PI7C7100 3-states the P_AD, P_CBE, and P_PAR signals on the next PCI clock cycle. If PI7C7100 is parking the primary PCI bus and wants to initiate a transaction on that bus, then PI7C7100 can start the transaction on the next PCI clock cycle by asserting P_FRAME# if P_GNT# is still asserted.

If the internal secondary bus arbiter is enabled, the secondary bus is always parked at the last master that used the PCI bus. That is, PI7C7100 keeps the secondary bus grant asserted to a particular master until a new secondary bus request comes along. After reset, PI7C7100 parks the secondary bus at itself until transactions start occurring on the secondary bus. If the internal arbiter is disabled, PI7C7100 parks the secondary bus only when the reconfigured grant signal, S_REQ#<0>, is asserted and the secondary bus is idle.

10. Clocks

This chapter provides information about the clocks.

10.1 Primary Clock Inputs

PI7C7100 implements a primary clock input for the PCI interface. The primary interface is synchronized to the primary clock input, P_CLK, and the secondary interface is synchronized to the secondary clock. The secondary clock is derived internally from the primary clock, P_CLK, through an internal PLL.

PI7C7100 operates at a maximum frequency of 33 MHz.

10.2 Secondary Clock Outputs

PI7C7100 has 16 secondary clock outputs, S_CLKOUT[15:0] that can be used as clock inputs for up to sixteen external secondary bus devices. The S_CLKOUT[15:0] outputs are derived from P_CLK. The secondary clock edges are delayed from P_CLK edges by a minimum of 0ns.

This is the rule for using secondary clocks:

- Each secondary clock output is limited to no more than one load.

11. Reset

This chapter describes the primary interface, secondary interface, and chip reset mechanisms.

11.1 Primary Interface Reset

PI7C7100 has a reset input, P_RESET#. When P_RESET# is asserted, the following events occur:

- PI7C7100 immediately 3-states all primary and secondary PCI interface signals.
- PI7C7100 performs a chip reset.
- Registers that have default values are reset.

P_RESET# asserting and de-asserting edges can be asynchronous to P_CLK and S_CLK.

11.2 Secondary Interface Reset

PI7C7100 is responsible for driving the secondary bus reset signals, S1_RESET# and S2_RESET#.

PI7C7100 asserts S1_RESET# or S2_RESET# when any of the following conditions is met:

- Signal P_RESET# is asserted.
  Signal S1_RESET# or S2_RESET# remains asserted as long as P_RESET# is asserted and does not de-assert until P_RESET# is de-asserted.
- The secondary reset bit in the bridge control register is set.
  Signal S1_RESET# or S2_RESET# remains asserted until a configuration write operation clears the secondary reset bit.
- S1_RESET# or S2_RESET# pin is asserted.
  When S1_RESET# or S2_RESET# is asserted, the following events occur:
  PI7C7100 immediately 3-states all the secondary PCI interface signals associated with the Secondary S1 or S2 port.
  The S1_RESET# or S2_RESET# in asserting and de-asserting edges can be asynchronous to P_CLK.
- The chip reset bit in the diagnostic control register is set.
  Signal S1_RESET# or S2_RESET# remains asserted until a configuration write operation clears the secondary reset bit and the secondary clock serial mask has been shifted in.

When S1_RESET# or S2_RESET# is asserted, all secondary PCI interface control signals, including the secondary grant outputs, are immediately 3-stated. Signals S1_AD, S1_CBE[3:0]#, S1_PAR (S2_AD, S2_CBE[3:0]#, S2_PAR) are driven low for the duration of S1_RESET# (S2_RESET#) assertion. All posted write and delayed transaction data buffers are reset. Therefore, any transactions residing inside the buffers at the time of secondary reset are discarded.

When S1_RESET# or S2_RESET# is asserted by means of the secondary reset bit, PI7C7100 remains accessible during secondary interface reset and continues to respond to accesses to its configuration space from the primary interface.

11.3 Chip Reset

The chip reset bit in the diagnostic control register can be used to reset PI7C7100 and the secondary buses. All registers, and chip state machines are reset and all signals are 3-stated when the chip reset is set. In addition, S1_RESET# or S2_RESET# is asserted, and the secondary reset bit is automatically set. Signal S1_RESET# or S2_RESET# remains asserted until a configuration write operation clears the secondary reset bit.

As soon as chip reset completes, within 20 PCI clock cycles after completion of the configuration write operation that sets the chip reset bit, the chip reset bit automatically clears and the chip is ready for configuration.

During chip reset, PI7C7100 is inaccessible.

12. Supported Commands

The PCI command set is given below for the primary and secondary interfaces.

12.1 Primary Interface

P_CBE[3:0]#	Command	Action
0000	Interrupt Acknowledge	Ignore.
0001	Special Cycle	Do not claim. Ignore.
0010	I/O Read	1. If address is within pass through I/O range: claim and pass through. 2. If address points to I/O mapped bridge internal register: claim and permit access to register, do not pass through. 3. Otherwise, do not pass through and do not claim for internal access.
0011	I/O Write	Same as I/O read.
0100	Reserved	- - - - -
0101	Reserved	- - - - -
0110	Memory Read	1. If address is within pass through memory range: claim and pass through. 2. If address is within pass through memory mapped I/O range: claim and pass through. 3. If address points to memory mapped bridge internal register: claim and permit access to register, do not pass through. 4. Otherwise, do not pass through and do not claim for internal access.
0111	Memory Write	Same as Memory Read
1000	Reserved	- - - - -
1001	Reserved	- - - - -
1010	Configuration Read	**I. Type 0 configuration read:** If the bridge's IDSEL line is asserted, perform function decode and claim if target function is implemented, otherwise, ignore. If claimed, permit access to target function's configuration registers. Do not pass through under any circumstances. **II. Type 1 configuration read:** 1. If the target bus is the bridge's secondary bus: claim and pass through as a type 0 configuration read. 2. If the target bus is a subordinate bus that exists behind the bridge (but not equal to the secondary bus): claim and pass through as a type 1 configuration read. 3. Otherwise, ignore.

12.1 Primary Interface

P_CBE[3:0]#	Command	Action
1011	Configuration Write	**I. Type 0 configuration write: same as configuration read.** **II. Type 1 configuration write(not special cycle request):** 1. If the target bus is the bridge's secondary bus: claim and pass through as a type 0 configuration write 2. If the target bus is a subordinate bus that exists behind the bridge (but not equal to the secondary bus): claim and pass through unchanged as a type 1 configuration write. 3. Otherwise, ignore. **III. Configuration write as special cycle request (device = 1Fh, function = 7h):** 1. If the target bus is the bridge's secondary bus: claim and pass through as a special cycle 2. If the target bus is a subordinate bus that exists behind the bridge (but not equal to the secondary bus): claim and pass through unchanged as a type 1 configuration write. 3. Otherwise, ignore
1100	Memory Read Multiple	Same as Memory Read
1101	Dual Address Cycle	Not Supported
1110	Memory Read Line	Same as Memory Read
1111	Memory Write & Invalidate	Same as Memory Read

12.2 Secondary Interface

S1_CBE[3:0]# S2_CBE[3:0]#	Command	Action
0000	Interrupt Acknowledge	Ignore.
0001	Special Cycle	Do not claim. Ignore.
0010	I/O Read	Same as primary interface.
0011	I/O Write	Same as I/O read.
0100	Reserved	- - - - -
0101	Reserved	- - - - -
0110	Memory Read	Same as primary interface.
0111	Memory Write	Same as Memory Read.
1000	Reserved	- - - - -
1001	Reserved	- - - - -
1010	Configuration Read	Ignore.
1011	Configuration Write	**I. Type 0 configuration write:** Ignore. **II. Type 1 configuration write (not special cycle request):** Ignore. **III. Configuration write as special cycle request (device = 1Fh, function = 7h):** 1. If the target bus is the bridge's primary bus: claim and pass through as a special cycle. 2. If the target bus is neither the primary bus nor is it in range of buses defined by the bridge's secondary and subordinate bus registers: claim and pass through unchanged as a type 1 configuration write. 3. If the target bus is not the bridge's primary bus: but is in range of buses defined by the bridge's secondary and subordinate bus registers: Ignore.
1100	Memory Read Multiple	Same as Memory Read
1101	Dual Address Cycle	Not Supported
1110	Memory Read Line	Same as Memory Read
1111	Memory Write & Invalidate	Same as Memory Read

13. Configuration Registers

As PI7C7100 supports two secondary interfaces, it has two sets of configuration registers which are almost identical and accessed through different function numbers. The description below is for one set only.

PCI configuration defines a 64-byte space (configuration header) to define various at3butes of the PCI-to-PCI Bridge as shown below. All of the registers in bold type are required by the PCI specification and are implemented in this bridge. The others are available for use as control registers for the device. There are two configuration registers: Configuration Register 1 and Configuration Register 2 corresponding to Secondary bus 1 and Secondary bus 2 interfaces respectively. Also, the configuration for the primary interface is implemented through the Configuration Register 1.

13.1 Configuration Register 1

31-24	23-16	15-8	7-0	Address
Device ID		**Vendor ID**		00h
Status		**Command**		04h
Class Code			**Revision ID**	08h
Reserved	**Header Type**	**Primary Latency Timer**	**Cache Line Size**	0Ch
Reserved				10h-14h
Secondary Latency Timer	**Subordinate Bus Number**	**Secondary Bus Number**	**Primary Bus Number**	18h
Secondary Status		**I/O Limit**	**I/O Base**	1Ch
Memory Limit		**Memory Base**		20h
Prefetchable Memory Limit		Prefetchable Memory Base		24h
Reserved				28h-2Ch
I/O Limit Upper 16 Bits		I/O Base Upper 16 Bits		30h
Subsystem ID		Subsystem Vendor ID		34h
Reserved				38h
Bridge Control		Interrupt Pin	Reserved	3Ch
Arbiter Control		Diagnostic Control	Chip Control	40h
Primary Prefetchable Memory Limit		Primary Prefetchable Memory Base		44h
Reserved				48h-60h
Reserved			P_SERR# Event Disable	64h
Reserved	Reserved	Secondary Clock Control		68h
Reserved				6Ch
Non-Posted Memory Limit		Non-Posted Memory Base		70h
Master Timeout Counter		Port Option		74h
Retry Counter				78h
Sampling Timer				7Ch
Secondary Successful I/O read count				80h
Secondary Successful I/O write count				84h
Secondary Successful memory read count				88h
Secondary Successful memory write count				8Ch
Primary Successful I/O read count				90h
Primary Successful I/O write count				94h
Primary Successful memory read count				98h
Primary Successful memory write count				9Ch
Reserved				A0h-FFh

13.2 Configuration Register 2

31-24	23-16	15-8	7-0	Address
Device ID		**Vendor ID**		00h
Status		**Command**		04h
Class Code			**Revision ID**	08h
Reserved	**Header Type**	**Primary Latency Timer**	**Cache Line Size**	0Ch
Reserved				10h-14h
Secondary Latency Timer	**Subordinate Bus Number**	**Secondary Bus Number**	**Primary Bus Number**	18h
Secondary Status		**I/O Limit**	**I/O Base**	1Ch
Memory Limit		**Memory Base**		20h
Prefetchable Memory Limit		Prefetchable Memory Base		24h
Reserved				28h-2Ch
I/O Limit Upper 16 Bits		I/O Base Upper 16 Bits		30h
Subsystem ID		Subsystem Vendor ID		34h
Reserved				38h
Bridge Control		Interrupt Pin	Reserved	3Ch
Arbiter Control		Diagnostic Control	Chip Control	40h
Primary Prefetchable Memory Limit		Primary Prefetchable Memory Base		44h
Reserved				48h-60h
Reserved				64h
Reserved	Reserved	Secondary Clock Control		68h
Reserved				6Ch
Non-Posted Memory Limit		Non-Posted Memory Base		70h
Reserved		Reserved		74h
Reserved				78h
Sampling Timer				7Ch
Secondary Successful I/O read count				80h
Secondary Successful I/O write count				84h
Secondary Successful memory read count				88h
Secondary Successful memory write count				8Ch
Reserved				90h
Reserved				94h
Reserved				98h
Reserved				9Ch
Reserved				A0h-FFh

13.2.1 Config Register 1 or 2: Vendor ID Register (read only, bit 15-0; offset 00h)

Pericom ID is 12D8h.

13.2.2 Config Register 1: Device ID Register (read only, bit 31-16; offset 00h)

Hardwired to 1B59h (S1)

13.2.3 Config Register 2: Device ID Register (read only, bit 31-16; offset 00h)

Hardwired to 1B5Ah (S2)

13.2.4 Configuration Register 1: Command Register (bit 15-0; offset 04h)

Bit	Function	Type	Description
15-10	Reserved	R/O	Reset to '000000'
9	Fast Back to Back Enable	R/W	Controls bridge's ability to generate fast back-to-back transactions to different devices on the primary interface. **0 = no fast back to back transaction** 1 = enable fast back to back transaction **Reset to 0**
8	SERR# Enable	R/W	Controls the enable for the P_SERR# pin. **0=disable the P_SERR# driver** 1 = enable the P_SERR# driver **Reset to 0**
7	Wait Cycle Control	R/O	No data stepping supported. **Reset to 0**
6	Parity Error Enable	R/W	Controls bridge's response to parity errors. **0 = ignore any parity errors** 1 = normal parity checking performed **Reset to 0**
5	VGA Palette Snoop Enable	R/W	Controls bridge's response to VGA compatible palette accesses. **0 = ignore VGA palette accesses on the primary interface** 1 = enable response to VGA palette writes on the primary interface (I/O address AD[9:0] = 3C6h, 3C8h and 3C9h) **Reset to 0**
4	Memory Write and Invalidate Enable	R/O	Memory Write and Invalidate not supported **Reset to 0**
3	Special Cycle Enable	R/O	No special cycle implementation **Reset to 0**
2	Bus Master Enable	R/W	Controls bridge's ability to operate as a master on the primary interface. **0 = do not initiate transaction on the primary interface and disable response to memory or I/O transactions on secondary interface** 1 = enable the bridge to operate as a master on the primary interface **Reset to 0**
1	Memory Space Enable	R/W	Controls bridge's response to memory accesses on the primary interface. **0 = ignore all memory transaction** 1 = enable response to memory transaction **Reset to 0**
0	I/O Space Enable	R/W	Controls bridge's response to I/O accesses on the primary interface. **0 = ignore I/O transaction** 1 = enable response to I/O transaction **Reset to 0**

Note: R/W - Read/Write, R/O - Read Only, R/WC - Read/ Write 1 to clear

13.2.5 Configuration Register 2: Command Register (bit 15-0; offset 04h)

Bit	Function	Type	Description
15-10	Reserved	R/O	**Reset to '000000'**
9	Reserved	R/W	**Reset to 0**
8	Reserved	R/W	**Reset to 0**
7	Wait Cycle Control	R/O	No data stepping supported. **Reset to 0**
6	Reserved	R/W	**Reset to 0**
5	VGA Palette Snoop Enable	R/W	Controls bridge's response to VGA compatible palette accesses. **0 = ignore VGA palette accesses on the primary interface** 1 = enable response to VGA palette writes on the primary interface (I/O address AD[9:0] = 3C6h, 3C8h and 3C9h) **Reset to 0**
4	Memory Write and Invalidate Enable	R/O	Memory Write and Invalidate not supported. **Reset to 0**
3	Special Cycle Enable	R/O	No special cycle implementation. **Reset to 0**
2	Bus Master Enable	R/W	Controls bridge's ability to operate as a master on the primary interface. **0 = do not initiate transaction on the primary interface and disable response to memory or I/O transactions on secondary interface** 1 = enable the bridge to operate as a master on the primary interface **Reset to 0**
1	Memory Space Enable	R/W	Controls bridge's response to memory accesses on the primary interface. **0 = ignore all memory transaction** 1 = enable response to memory transaction **Reset to 0**
0	I/O Space Enable	R/W	Controls bridge's response to I/O accesses on the primary interface. **0 = ignore I/O transaction** 1 = enable response to I/O transaction **Reset to 0**

Note: R/W - Read/Write, R/O - Read Only, R/WC - Read/ Write 1 to clear

13.2.6 Configuration Register 1 or 2: Status Register (for primary bus, bits 31-16; offset 04h)

Bit	Function	Type	Description
31	Detected Parity Error	R/WC	Should be set whenever a parity error is detected regardless of the state of bit 6 of the command register. **Reset to 0**
30	Signaled System Error	R/WC	Should be set whenever P_SERR# is asserted. **Reset to 0**
29	Received Master Abort	R/WC	Set to '1' (by a master) when transactions are terminated with Master Abort. **Reset to 0**
28	Received Target Abort	R/WC	Set to '1' (by a master device) when transactions are terminated with Target Abort. **Reset to 0**
27	Signaled Target Abort	R/WC	Should be set (by a target device) whenever a Target Abort cycle occurs. **Reset to 0**
26-25	DEVSEL Timing	R/O	Medium DEVSEL# timing. **Reset to '01'**
24	Data Parity Error Detected	R/WC	It is set when the following conditions are met: 1. P_PERR# is asserted 2. Bit 6 of Command Register is set **Reset to 0**
23	Fast Back to Back Capable	R/O	Fast back-to-back write capable on primary side. **Reset to 1**
22	Reserved	R/O	**Reset to 0**
21	Reserved	R/O	**Reset to 1**
20	Capabilities List	R/O	Capabilities List is not supported. **Reset to 0**
19-16	Reserved	R/O	**Reset to 0**

Note: R/W - Read/Write; R/O - Read Only; R/WC - Read/Write1 to clear.

13.2.7 Config Register 1 or 2: Revision ID Register (read only, bit 7-0; offset 08h)

Hardwired to 01h

13.2.8 Config Register 1 or 2: Class Code Register (read only, bit 31-8; offset 08h)

Hardwired to 060400h

13.2.9 Config Register 1 or 2: Cache Line Size Register (read/write, bit 7-0; offset 0Ch)

This register is used when terminating memory write and invalidate transactions and when pre-fetching.

Only cache line sizes (in units of 4-byte) which are power of two are valid (only one bit can be set in this register; only 00h, 01h, 02h, 04h, 08h, 10h are valid values). Reset to 00h

13.2.10 Config Register 1: Primary Latency Timer Register (read/write, bit 15-8; offset 0Ch)

This register sets the value for Master Latency Timer which starts counting when master asserts FRAME#. Reset to 00h

13.2.11 Config Register 2: Primary Latency Timer Register (read/write, bit 15-8; offset 0Ch)

This register is implemented but not being used internally. Reset to 00h

13.2.12 Config Register 1: Header Type Register (read only, bit 23-16; offset 0Ch)

Hardwired to 81h for function 0 (multiple function PCI-to-PCI bridge, for secondary bus S1)

13.2.13 Config Register 2: Header Type Register (read only, bit 23-16; offset 0Ch)

Hardwired to 01h for function 1 (single function PCI-to-PCI bridge, for secondary bus S2)

13.2.14 Config Register 1: Primary Bus Number Register (read/write, bit 7-0; offset 18h)

Programmed with the number of the PCI bus to which the primary bridge interface is connected.
This value is set by software during configuration. Reset to 00h

13.2.15 Config Register 2: Primary Bus Number Register (read/write, bit 7-0; offset 18h)

This register is implemented but not being used internally. Reset to 00h

13.2.16 Config Register 1 or 2: Secondary Bus Number Register (read/write, bit 15-8; offset 18h)

Programmed with the number of the PCI bridge secondary bus interface. This value is set by software during configuration. Reset to 00h

13.2.17 Config Register 1 or 2: Subordinate Bus Number Register (read/write, bit 23-16; offset 18h)

Programmed with the number of the PCI bus with the highest number that is subordinate to the bridge.
This value is set by software during configuration. Reset to 00h

13.2.18 Config Register 1 or 2: Secondary Latency Timer (read/write, bit 31-24; offset 18h)

This register is programmed in units of PCI bus clocks.The latency timer checks for master accesses on the secondary bus interfaces that remain unclaimed by any target. Reset to 00h

13.2.19 Config Register 1 or 2: I/O Base Register (read/write, bit 7-0; offset 1Ch)

This register defines the bottom address of the I/O address range for the bridge. The upper four bits define the bottom address range used by the chip to determine when to forward I/O transactions from one interface to the other. These 4 bits correspond to address bits [15:12] and are write-able. The upper 16 bits corresponding to address bits [31:16] are defined in the I/O base upper 16 bits address register. The address bits [11:0] are assumed to be 000h. The lower four bits (3:0) of this register set to '0001' (read-only) to indicate 32-bit I/O addressing. Reset to 00h

13.2.20 Config Register 1 or 2: I/O Limit Register (read/write, bit 15-8; offset 1Ch)

This register defines the top address of the I/O address range for the bridge. The upper four bits define the top address range used by the chip to determine when to forward I/O transactions from one interface to the other. These 4 bits correspond to address bits [15:12] and are write-able. The upper 16 bits corresponding to address bits [31:16] are defined in the I/O limit upper 16 bits address register. The address bits [11:0] are assumed to be FFFh. The lower four bits (3:0) of this register set to '0001' (read-only) to indicate 32-bit I/O addressing. Reset to 00h.

13.2.21 Configuration Register 1 or 2: Secondary Status Register (bits 31-16; offset 1Ch)

Bit	Function	Type	Description
31	Detected Parity Error	R/WC	Should be set whenever a parity error is detected regardless of the state of bit 6 of the command register. **Reset to 0**
30	Signaled System Error	R/WC	Should be set whenever S1_SERR# or S2_SERR# is detected. Should be a '0' after reset. **Reset to 0**
29	Received Master Abort	R/WC	Set to '1' (by a master) when transactions are terminated with Master Abort. **Reset to 0**
28	Received Target Abort	R/WC	Set to '1' (by a master device) when transactions are terminated with Target Abort. **Reset to 0**
27	Signaled Target Abort	R/WC	Should be set (by a target device) whenever a Target Abort cycle occurs. Should be '0' after reset. **Reset to 0**
26-25	DEVSEL timing	R/O	Medium DEVSEL# timing. **Reset to '01'**
24	Data Parity Error Detected	R/WC	It is set when the following conditions are met: 1. S1_PERR# or S2_PERR# is asserted 2. Bit 6 of Command Register is set **Reset to 0**
23	Fast Back-to-Back Capable	R/O	Fast back-to-back write capable on secondary buses. **Reset to 1**
22	Reserved	R/O	**Reset to 0**
21	Reserved	R/O	**Reset to 0**
20-16	Reserved	R/O	**Reset to '00000'**

Note: R/W - Read/Write, R/O - Read Only, R/WC - Read/ Write 1 to clear

13.2.22 Config Register 1 or 2: Memory Base Register (read/write, bit 15-0; offset 20h)

This register defines the base address of the memory-mapped address range for forwarding the cycle through the bridge. The upper twelve bits corresponding to address bits [31:20] are read/write. The twelve bits are reset to 000h. The lower 20 address bits (19:0) are assumed to be 00000h.

13.2.23 Config Register 1 or 2: Memory Limit Register (read/write, bit 31:16; offset 20h)

This register defines the upper limit address of the memory-mapped address range for forwarding the cycle through the bridge. Upper twelve bits corresponding to address bit [31:20] are read/write. Upper twelve bits are reset to 0000h. Lower 20 address bits (19:0) are assumed to be FFFFFh.

13.2.24 Config Register 1 or 2: Prefetchable Memory Base Register (read/write, bit 15-0;offset 24h)

This register defines the base address of the prefetchable memory-mapped address range for forwarding the cycle through the bridge. The upper twelve bits corresponding to address bits [31:20] are read/write. The upper twelve bits are reset to 000h. The lower four bits are read only and are set to 0. The lower 20 address bits (19:0) are assumed to be 00000h.

13.2.25 Config Register 1 or 2: Prefetchable Memory Limit Register (read/write, bit 31-16; offset 24h)

This register defines the upper limit address of the memory-mapped address range for forwarding the cycle through the bridge. The upper twelve bits correspond to address bit [31:20] are read/write. The upper twelve bits are reset to 000h. The lower four bits are read only and are set to 0. The lower 20 address bits (19:0) are assumed to be FFFFFh.

13.2.26 Config Register 1 or 2: I/O Base Address Upper 16 Bits Register (read/write, bit 15-0; offset 30h)

This register defines the upper 16 bits of a 32-bit base I/O address range used for forwarding the cycle through the bridge.

Reset to 0000h.

13.2.27 Config Register 1 or 2: I/O Limit Address Upper 16 Bits Register (read/write, bit 31-16; offset 30h)

This register defines the upper 16 bits of a 32-bit limit I/O address range used for forwarding the cycle through the bridge.

Reset to 0000h.

13.2.28 Config Register 1 or 2: Subsystem Vendor ID (read/write, bit 15-0; offset 34h)

A 16-bit register for add-on cards to distinguish from one another. Reset to 0000h.

13.2.29 Config Register 1 or 2: Subsystem ID (read/write, bit 31-16; offset 34h)

A 16-bit register for add-on cards to distinguish from one another. Reset to 0000h.

13.2.30 Config Register 1 or 2: Interrupt Pin Register (read only, bit 15-8; offset 3Ch)

The register reads as 00h to indicate that PI7C7100 does not use any interrupt pins.

13.2.31 Configuration Register 1 or 2: Bridge Control Register (bits 31-16; offset 3Ch)

Bit	Function	Type	Description
31-28	Reserved	R/O	**Reset to '0000'**
27	Reserved	R/W	**Reset to 0**
26	Master Timeout Status	R/WC	Set to '1' when either primary master or secondary master timeout. **Reset to 0**
25	Reserved	R/W	**Reset to 0**
24	Reserved	R/W	**Reset to 0**
23	Fast Back-to-Back Enable	R/W	Controls bridge's ability to generate fast back-to-back transactions to **different** devices on the secondary interface. **0 = no fast back-to-back transaction** 1 = enable fast back-to-back transacton **Reset to 0**
22	Secondary Interface Reset	R/W	Forces the assertion of S1_RESET# or S2_RESET# signal pin on the secondary interface. **0 = do not force the assertion of S1_RESET# or S2_RESET# pin** 1 = force the assertion of S1_RESET# or S2_RESET# pin **Reset to 0**
21	Master Abort Mode	R/W	Controls bridge's behavior responding to master aborts on secondary interface. **0 = do not report master aborts (return FFFF_FFFFh on read and discard data on write)** 1 = report master aborts by signaling target abort if possible by the assertion of P_SERR# if enabled **Reset to 0**
20	Reserved	R/O	**Reset to 0**
19	VGA Enable	R/W	Controls the bridge's response to VGA compatible addresses. **0 = do not forward VGA compatible memory and I/O addresses from primary to secondary** 1 = forward VGA compatible memory and I/O address from primary to secondary regardless of other settings **Reset to 0**
18	ISA Enable	R/W	Controls bridge's response to ISA I/O address which is limited to the first 64K. **0 = forward all I/O addresses in the range defined by the I/O Base and I/O Limit registers**, 1 = block forwarding of ISA I/O addresses in the range defined by the I/O Base and I/O Limit registers that are in the first 64K of I/O space that address the last 768 bytes in each 1 Kbytes block. Secondary I/O transactions are forwarded upstream if the address falls within the last 768 bytes in each 1 Kbyte block **Reset to 0**
17	S1_SERR# or S2_SERR# Enable		Controls the forwarding of S1_SERR# or S2_SERR# to the primary interface. **0 = disable the forwarding S1_SERR# or S2_SERR# to primary** 1 = enable the forwarding of S1_SERR# or S2_SERR# to primary interface. **Reset to 0**
16	Parity Error Response Enable		Controls the bridge's response to parity errors on the secondary interface. **0 = ignore address and data parity errors on the secondary interface.** 1 = enable parity error reporting and detection on the secondary interface. **Reset to 0**

Note: R/W - Read/Write, R/O - Read Only, R/WC - Read/ Write 1 to clear.

13.2.32 Configuration Register 1 or 2: Diagnostic/Chip Control Register (bit 15-0, offset 40h)

Bit	Function	Type	Description
15-11	Reserved	R/O	**Reset to '00000'**
10-9	Test Mode	R/W	Controls testability of chip's internal counters. When 00, all bits of counter are exercised. When 01, byte 1 of counter is exercised. When 10, byte 2 of counter is exercised. When 11, byte 3 of counter is exercised. **Reset to 0**
8-5	Reserved	R/O	**Reset to '0000'**
4	Reserved	R/W	**Reset to 0**
3-2	Reserved	R/O	**Reset to '00'**
1	Reserved	R/W	**Reset to 0**
0	Reserved	R/O	**Reset to 0**

13.2.33 Configuration Register 1 or 2: Arbiter Control Register (bit 31-16, offset 40h)

Bit	Function	Type	Description
31:28	Reserved	R/O	**Reset to '0000'**
27	Hybrid	R/W	Mixed arbitration for masters from secondary bus 1 and 2. 0 = separate arbitration for S1_REQ[7:0]# and S2_REQ[7:0]# 1 = S1_REQ[3:0]# are mixed with S2_REQ[3:0]# for arbitration. Only one arbiter is used. **Reset to 0**
26	Reserved	R/W	**Reset to 0**
25	Priority of Secondary Port	R/W	Defines whether the secondary port of PI7C7100 is in high priority group or the low priority group. 0 = low priority group **1 = high priority group** **Reset to 1**
24	Reserved	R/O	**Reset to 0**
23-16	Arbiter Control	R/W	Each bit controls whether a secondary-bus master is assigned to the high priority group or the low priority group. Bit [7:0] correspond to request inputs S1_REQ[7:0]# or S2_REQ[7:0]#. **Reset to '00000000'**

Note: R/W - Read/Write, R/O - Read Only, R/WC - Read/ Write 1 to clear.

13.2.34 Config Register 1: Primary Prefetchable Memory Base Register (Read/Write, bit 15-0; offset 44h)

This register defines the base address of the primary prefetchable memory-mapped address range for forwarding the cycle through the bridge. The upper twelve bits corresponding to address bits [31:20] are read/write. The upper twelve bits are reset to 0000h. The lower four bits are read only and are set to 0h. The lower 20 address bits (19:0) are assumed to be 00000h.

13.2.35 Config Register 2: Primary Prefetchable Memory Base Register (Read/Write, bit 15-0; offset 44h)

This register is implemented but not being used internally. The upper twelve bits corresponding to address bits [31:20] are read/write. The upper twelve bits are reset to 0000h. The lower four bits are read only and are set to 0h.

13.2.36 Config Register 1: Primary Prefetchable Memory Limit Register (Read/Write, bit 31-16; offset 44h)

This register defines the upper limit address of the primary prefetchable memory-mapped address range for forwarding the cycle through the bridge. The upper twelve bits corresponding to address bits [31:20] are read/write. The upper twelve bits are reset to 0000h. The lower four bits are read only and are set to 0h. The lower 20 address bits (19:0) are assumed to be FFFFFh.

13.2.37 Config Register 2: Primary Prefetchable Memory Limit Register (Read/Write, bit 31-16; offset 44h)

This register is implemented but not being used internally. The upper twelve bits corresponding to address bits [31:20] are read/write. The upper twelve bits are reset to 0000h. The lower four bits are read only and are set to 0h.

13.2.38 Config Register 1 or 2: P_SERR# Event Disable Register (bit 7-0; offset 64h)

Bit	Function	Type	Description
7	Reserved	R/O	**Reset to 0**
6	Delayed read - no data from target	R/W	Controls ability of PI7C7100 to assert P_SERR# when it is unable to transfer any read data from the target after 2^{24} attempts. P_SERR# is asserted if this event occurs when this bit is 0 and SERR# enable bit in the command register is set. **Reset to 0**
5	Delayed write nondeliver	R/W	Controls ability of PI7C7100 to assert P_SERR# when it is unable to transfer delayed write data after 2^{24} attempts. P_SERR# is asserted if this event occurs when this bit is 0 and SERR# enable bit in the command register is set. **Reset to 0**
4	Master abort on posted write	R/W	Controls ability of PI7C7100 to assert P_SERR# when it receives a master abort when attempting to deliver posted write data. P_SERR# is asserted if this event occurs when this bit is 0 and SERR# enable bit in the command register is set. **Reset to 0**
3	Target abort during posted write	R/W	Controls ability of PI7C7100 to assert P_SERR# when it receives a target abort when attempting to deliver posted write data. P_SERR# is asserted if this event occurs when this bit is 0 and SERR# enable bit in the command register is set. **Reset to 0**
2	Posted write non-delivery	R/W	Controls ability of PI7C7100 to assert P_SERR# when it is unable to deliver posted write data after 2^{24} attempts. P_SERR# is asserted if this event occurs when this bit is 0 and SERR# enable bit in the command register is set. **Reset to 0**
1	Posted write parity error	R/W	Controls ability of PI7C7100 to assert P_SERR# when a parity error is detected on the target bus during a posted write transaction. P_SERR# is asserted if this event occurs when this bit is 0 and SERR# enable bit in the command register is set. **Reset to 0**
0	Reserved	R/O	**Reset to 0**

Note: R/W - Read/Write, R/O - Read Only, R/WC - Read/ Write 1 to clear.

13.2.39 Configuration Register 1: Secondary Clock Control Register (bit 15-0; offset 68h)

Bit	Function	Type	Description
15-14	Clock 7 Disable	R/W	If either bit is 0, S_CLKOUT[7] is enabled When both bits are 1, S_CLKOUT[7] is disabled
13-12	Clock 6 Disable		If either bit is 0, S_CLKOUT[6] is enabled When both bits are 1, S_CLKOUT[6] is disabled
11-10	Clock 5 Disable		If either bit is 0, S_CLKOUT[5] is enabled When both bits are 1, S_CLKOUT[5] is disabled
9-8	Clock 4 Disable		If either bit is 0, S_CLKOUT[4] is enabled When both bits are 1, S_CLKOUT[4] is disabled
7-6	Clock 3 Disable		If either bit is 0, S_CLKOUT[3] is enabled When both bits are 1, S_CLKOUT[3] is disabled
5-4	Clock 2 Disable		If either bit is 0, S_CLKOUT[2] is enabled When both bits are 1, S_CLKOUT[2] is disabled
3-2	Clock 1 Disable		If either bit is 0, S_CLKOUT[1] is enabled When both bits are 1, S_CLKOUT[1] is disabled
1-0	Clock 0 Disable		If either bit is 0, S_CLKOUT[0] is enabled When both bits are 1, S_CLKOUT[0] is disabled

13.2.40 Configuration Register 2: Secondary Clock Control Register (bit 15-0; offset 68h)

Bit	Function	Type	Description
15-14	Clock 7 Disable	R/W	If either bit is 0, S_CLKOUT[15] is enabled When both bits are 1, S_CLKOUT[15] is disabled
13-12	Clock 6 Disable		If either bit is 0, S_CLKOUT[14] is enabled When both bits are 1, S_CLKOUT[14] is disabled
11-10	Clock 5 Disable		If either bit is 0, S_CLKOUT[13] is enabled When both bits are 1, S_CLKOUT[13] is disabled
9-8	Clock 4 Disable		If either bit is 0, S_CLKOUT[12] is enabled When both bits are 1, S_CLKOUT[12] is disabled
7-6	Clock 3 Disable		If either bit is 0, S_CLKOUT[11] is enabled When both bits are 1, S_CLKOUT[11] is disabled
5-4	Clock 2 Disable		If either bit is 0, S_CLKOUT[10] is enabled When both bits are 1, S_CLKOUT[10] is disabled
3-2	Clock 1 Disable		If either bit is 0, S_CLKOUT[9] is enabled When both bits are 1, S_CLKOUT[9] is disabled
1-0	Clock 0 Disable		If either bit is 0, S_CLKOUT[8] is enabled When both bits are 1, S_CLKOUT[8] is disabled

Note: R/W - Read/Write.

13.2.41 Config Register 1 or 2: Non-Posted Memory Base Register (read/write, bit 15-0; offset 70h)

This register defines the base address of the non-posted memory-mapped address range for forwarding the cycle through the bridge. Upper twelve bits corresponding to address bits [31:20] are read/write. Lower 20 bits (19:0) are assumed to be 00000h.

13.2.42 Config Register 1 or 2: Non-Posted Memory Limit Register (read/write, bit 31-16; offset 70h)

This register defines the upper limit address of the non-posted memory-mapped address range for forwarding the cycle through the bridge. Upper twelve bits corresponding to address bits [31:20] are read/write. Lower 20 bits (19:0) are assumed to be FFFFFh.

13.2.43 Configuration Register 1: Port Option Register (bit 15-0; offset74h)

Bit	Function	Type	Description
15-13	Reserved	R/O	**Reset to '000'**
12	Primary Pre Read	R/W	Enable 1 more read for MEMR command on primary 1 = Enable **0 = No change**
11-10	Reserved	R/O	**Reset to '00'**
9	Enable Long Request	R/W	Enable Long request for lock cycle **0 = No change** 1 = Enable
8	Reset DTQUEUE	R/W	Reset Secondary Delayed Transaction Queue **0 = No change** **1 = Reset**
7-6	Reserved	R/O	Reset to '00'
5	ID Write Enable	R/W	Allow write to Vendor ID, Device ID, Subsystem Vendor ID and Subsystem ID in the configuration space. **0 = Write protect** 1 = Write enable **Reset to 0**
4	Secondary MEMW Command Alias Enable	R/W	Controls the bridge's detection mechanism for matching non-posted memory write retry cycle from initiator on secondary interface. **0 = Command has to be exact** 1 = MEMW is equivalent to MEMWI **Reset to 0**
3	Secondary MEMR Command Alias Enable	R/W	Controls the bridge's detection mechanism for matching memory read retry cycle from initiator on secondary interface. **0=Command has to be exact** 1=MEMR is equivalent to MEMRL or MEMRM **Reset to 0**
2	Primary MEMW Command Alias Enable	R/W	Controls the bridge's detection mechanism for matching non-posted memory write retry cycle from initiator on primary interface. **0 = Command has to be exact** 1 = MEMW is equivalent to MEMWI **Reset to 0**
1	Primary MEMR Command Alias Enable	R/W	Controls the bridge's detection mechanism for matching memory read retry cycle from initiator on primary interface. **0 = Command has to be exact** 1 = MEMR is equivalent to MEMRL or MEMRM **Reset to 0**
0	Secondary Pre Read	R/W	Enable 1 more read for MEMR command on secondary. **0 = disable** 1 = enable **Reset to 0**

13.2.44 Configuration Register 2: Port Option Register (bit 15-0; offset74h)

Bit	Function	Type	Description
15-13	Reserved	R/O	**Reset to '000'**
12	Reserved	R/W	**Reset to 0**
11-10	Reserved	R/O	**Reset to '00'**
9-8	Reserved	R/W	**Reset to '00'**
7-6	Reserved	R/O	**Reset to '00'**
5	ID Write Enable	R/W	Allow write to Vendor ID, Device ID, Subsystem Vendor ID, and Subsystem ID in the configuration space. **0 = Write protect** 1 = Write enable **Reset to 0**
4	Secondary MEMW Command Alias Enable	R/W	Controls the bridge's detection mechanism for matching non-posted memory write retry cycle from initiator on secondary interface. **0 = Command has to be exact** 1 = MEMW is equivalent to MEMWI **Reset to 0**
3	Secondary MEMR Command Alias Enable	R/W	Controls the bridge's detection mechanism for matching memory read retry cycle from initiator on secondary interface. **0=Command has to be exact** 1=MEMR is equivalent to MEMRL or MEMRM **Reset to 0**
2	Reserved	R/W	**Reset to 0**
1	Reserved	R/W	**Reset to 0**
0	Secondary Pre Read	R/W	Enable 1 more read for MEMR command on secondary. **0 = disable** 1 = enable **Reset to 0**

13.2.45 Config Register 1 or 2: Master Timeout Counter Register (read/write, bit 31-16; offset 74h)

This register holds the maximum number of PCI clocks that PI7C7100 will wait for initiator to retry the same cycle before reporting timeout. Default is 8000h.

13.2.46 Config Register 1 or 2: Retry Counter Register (read/write, bit 31-0; offset 78h)

This register holds the maximum number of attempts that PI7C7100 will try before reporting retry timeout. Default is 0100_0000h.

13.2.47 Config Register 1 or 2: Sampling Timer Register (read/write, bit 31-0; offset 7Ch)

This register set the duration (in PCI clocks) during which PI7C7100 will record the number of successful transactions for performance evaluation. The recording will start right after this register is programmed and will be cleared after the timer expires. The maximum period is 128 seconds. Reset to 0000_0000h.

13.2.48 Config Register 1 or 2: Successful I/O Read Count Register (read/write, bit 31-0; offset 80h)

This register stores the successful I/O read count on the secondary interface which will be updated when the sampling timer is active. Reset to 0000_0000h.

13.2.49 Config Register 1 or 2: Successful I/O Write Count Register (read/write, bit 31-0; offset 84h)

This register stores the successful I/O write count on the secondary interface which will be updated when the sampling timer is active. Reset to 0000_0000h.

13.2.50 Config Register 1 or 2: Successful Memory Read Count Register (read/write, bit 31-0; offset 88h)

This register stores the successful memory read count on the secondary interface which will be updated when the sampling timer is active. Reset to 0000_0000h.

13.2.51 Config Register 1 or 2: Successful Memory Write Count Register (read/write, bit 31-0; offset 8Ch)

This register stores the successful memory write count on the secondary interface which will be updated when the sampling timer is active. Reset to 0000_0000h.

13.2.52 Config Register 1: Primary Successful I/O Read Count Register (read/write, bit 31-0; offset 90h)

This register stores the successful I/O read count on the primary interface which will be updated when the sampling timer is active. Reset to 0000_0000h.

13.2.53 Config Register 1: Primary Successful I/O Write Count Register (read/write, bit 31-0; offset 94h)

This register stores the successful I/O write count on the primary interface which will be updated when the sampling timer is active. Reset to 0000_0000h.

13.2.54 Config Register 1: Primary Successful Memory Read Count Register (read/write, bit 31-0; offset 98h)

This register stores the successful memory read count on the primary interface which will be updated when the sampling timer is active. Reset to 0000_0000h.

13.2.55 Config Register 1: Primary Successful Memory Write Count Register (read/write, bit 31-0; offset 9Ch)

This register stores the successful memory write count on the primary interface which will be updated when the sampling timer is active. Reset to 0000_0000h.

14. Bridge Behavior

A PCI cycle is initiated by asserting the FRAME# signal. In a bridge, there are a number of possibilities. Those possibilities are summarized in the table below:

14.1 Bridge Actions for Various Cycle Types

Initiator	Target	Response
Master on primary	Target on Primary	PI7C7100 does not respond. It detects this situation by decoding the address as well as monitoring the P_DEVSEL# for other fast and medium devices on the primary port.
Master on primary	Target on secondary	PI7C7100 asserts P_DEVSEL#, terminates the cycle normally if it is able to be posted, otherwise returns with a retry. It then passes the cycle to the appropriate port. When the cycle is complete on the target port, it will wait for the initiator to repeat the same cycle and end with normal termination.
Master on primary	Target not on primary nor secondary port	PI7C7100 does not respond and the cycle will terminate as master abort.
Master on secondary	Target on the same secondary port	PI7C7100 does not respond.
Master on secondary	Target on primary or the other secondary port	PI7C7100 asserts S1_DEVSEL# or S2_DEVSEL#, terminates the cycle normally if it is able to be posted, otherwise returns with a retry. It then passes the cycle to the appropriate port. When cycle is complete on the target port, it will wait for the initiator to repeat the same cycle and end with normal termination.
Master on secondary	Target not on primary nor the other secondary	PI7C7100 does not respond.

A target then has up to three cycles to respond before subtractive decoding is initiated. If the target detects an address hit, it should assert its DEVSEL# signal in the cycle corresponding to the values of bits 9 and 10 in the Configuration Status Register.

Termination of a PCI cycle can occur in a number of ways. Normal termination begins by the initiator (master) de-asserting FRAME# with IRDY# being asserted (or remaining asserted) on the same cycle. The cycle completes when TRDY# and IRDY# are both asserted simultaneously. The target should de-assert TRDY# for one cycle following final assertion (sustained 3-state signal).

14.2 Transaction Ordering

To maintain data coherency and consistency, PI7C7100 complies with the ordering rules put forth in the PCI Local Bus Specification, Rev 2.1. The following table summarizes the ordering relationship of all the transactions through the bridge.

PMW - Posted write (either memory write or memory write & invalidate)

DRR - Delayed read request (all memory read, I/O read & configuration read)

DWR - Delayed write request (I/O write & configuration write)

DRC - Delayed read completion (all memory read, I/O read & configuration read)

DWC - Delayed write completion (I/O write & configuration write)

Cycle type shown on each row is the subsequent cycle after the previous shown on the column.

Can Row pass Column?	PMW Column 1	DRR Column 2	DWR Column 3	DRC Column 4	DWC Column 5
PMW (Row 1)	No	Yes	Yes	Yes	Yes
DRR (Row 2)	No	No	No	Yes	Yes
DWR (Row 3)	No	No	No	Yes	Yes
DRC (Row 4)	No	Yes	Yes	No	No
DWC (Row 5)	Yes	Yes	Yes	No	No

In Row 1 Column 1, PMW cannot pass the previous PMW and that means they must complete on the target bus in the order in which they were received in the initiator bus.

In Row 2 Column 1, DRR cannot pass the previous PMW and that means the previous PMW heading to the same direction must be completed before the DRR can be attempted on the target bus.

In Row 1 Column 2, PMW can pass the previous DRR as long as the DRR reaches the head of the delayed transaction queue.

14.3 Abnormal Termination (Initiated by Bridge Master)

14.3.1 Master Abort

Master abort indicates that when PI7C7100 acts as a master and receives no response (i.e., no target asserts P_DEVSEL# or S1_DEVSEL# or S2_DEVSEL#) from a target, the bridge de-asserts FRAME# and then de-asserts IRDY#.

14.3.2 Parity and Error Reporting

Parity must be checked for all addresses and write data. Parity is defined on the P_PAR, S1_PAR, and S2_PAR signals. Parity should be even (i.e. an even number of '1's) across AD, CBE, and PAR. Parity information on PAR is valid the cycle after AD and CBE are valid. For reads, even parity must be generated using the initiators CBE signals combined with the read data. Again, the PAR signal corresponds to read data from the previous data phase cycle.

14.3.3 Reporting Parity Errors

For all address phases, if a parity error is detected, the error should be reported on the P_SERR# signal by asserting P_SERR# for one cycle and then 3-stating two cycles after the bad address. P_SERR# can only be asserted if bit 6 and 8 in the Command Register are both set to 1. For write data phases, a parity error should be reported by asserting the P_PERR# signal two cycles after the data phase and should remain asserted for one cycle when bit 8 in the Command register is set to a 1. The target reports any type of data parity errors during write cycles, while the master reports data parity errors during read cycles.

Detection of an address parity error will cause the PCI-to-PCI Bridge target to not claim the bus (P_DEVSEL# remains inactive) and the cycle will then terminate with a Master Abort. When the bridge is acting as master, a data parity error during a read cycle results in the bridge master initiating a Master Abort.

14.3.4 Secondary IDSEL mapping

When PI7C7100 detects a Type 1 configuration transaction for a device connected to the secondary, it translates the Type 1 transaction to Type 0 transaction on the downstream interface. Type 1 configuration format uses a 5-bit field at P_AD[15:11] as a device number. This is translated to S1_AD[31:16] or S2_AD[31:16] by PI7C7100.

15. IEEE 1149.1 Compatible JTAG Controller

An IEEE 1149.1 compatible Test Access Port (TAP) controller and associated TAP pins are provided to support boundary scan in PI7C7100 for board-level continuity test and diagnostics. The TAP pins assigned are TCK, TDI, TDO, TMS and TRST#. All digital input, output, input/output pins are tested except TAP pins and clock pin.

The IEEE 1149.1 Test Logic consists of a TAP controller, an instruction register, and a group of test data registers including Bypass, Device Identification and Boundary Scan registers. The TAP controller is a synchronous 16 state machine driven by the Test Clock (TCK) and the Test Mode Select (TMS) pins. An independent power on reset circuit is provided to ensure the machine is in TEST_LOGIC_RESET state at power-up. The JTAG signal lines are not active when the PCI resource is operating PCI bus cycles.

PI7C7100 implements 3 basic instructions: BYPASS, SAMPLE/PRELOAD, EXTEST.

15.1 Boundary Scan Architecture

Boundary-scan test logic consists of a boundary-scan register and support logic. These are accessed through a Test Access Port (TAP). The TAP provides a simple serial interface that allows all processor signal pins to be driven and/or sampled, thereby providing direct control and monitoring of processor pins at the system level.

This mode of operation is valuable for design debugging and fault diagnosis since it permits examination of connections not normally accessible to the test system. The following subsections describe the boundary-scan test logic elements: TAP pins, instruction register, test data registers and TAP controller. Figure 15-1 illustrates how these pieces fit together to form the JTAG unit.

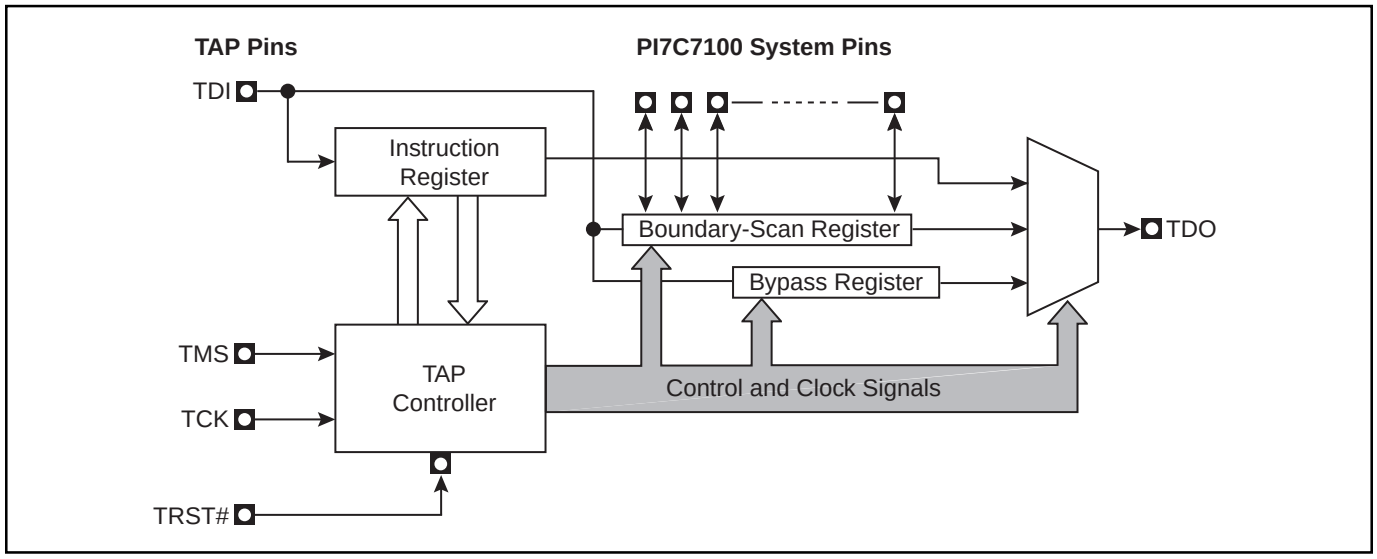


Figure 15-1. Test Access Port Block Diagram

15.1.1 TAP Pins

The PI7C7100's TAP pins form a serial port composed of four input connections (TMS, TCK, TRST# and TDI) and one output connection (TDO). These pins are described in Table 15-1. The TAP pins provide access to the instruction register and the test data registers.

15.1.2 Instruction Register

The Instruction Register (IR) holds instruction codes. These codes are shifted in through the Test Data Input (TDI) pin. The instruction codes are used to select the specific test operation to be performed and the test data register to be accessed.

The instruction register is a parallel-loadable, master/slave-configured 2-bit wide, serial-shift register with latched outputs. Data is shifted into and out of the IR serially through the TDI pin clocked by the rising edge of TCK. The shifted-in instruction becomes active upon latching from the master stage to the slave stage. At that time the IR outputs along with the TAP finite state machine outputs are decoded to select and control the test data register selected by that instruction. Upon latching, all actions caused by any previous instructions terminate.

The instruction determines the test to be performed, the test data register to be accessed, or both. The IR is two bits wide. When the IR is selected, the most significant bit is connected to TDI, and the least significant bit is connected to TDO. The value presented on the TDI pin is shifted into the IR on each rising edge of TCK. The TAP controller captures fixed parallel data (01 binary). When a new instruction is shifted in through TDI, the value 01 (binary) is always shifted out through TDO, least significant bit first. This helps identify instructions in a long chain of serial data from several devices.

Upon activation of the TRST# reset pin, the latched instruction asynchronously changes to the idcode instruction. When the TAP controller moves into the test state other than by reset activation, the opcode changes as TDI shifts, and becomes active on the falling edge of TCK.

15.2 Boundary-Scan Instruction Set

The PI7C7100 supports three mandatory boundary-scan instructions (bypass, sample/preload and extest). The table shown below lists the PI7C7100's boundary-scan instruction codes. The "reserved" code should not be used.

Instruction Code (binary)	Instruction Name	Instruction Code (binary)	Instruction Name
00	extest	10	reserved
01	sample/preload	11	bypass

Table 15-1. TAP Pins

Instruction / Requisite	Opcode (binary)	Description
extest IEEE 1149.1 Required	00	Extest initiates testing of external circuitry, typically board-level interconnects and off chip circuitry. extest connects the boundary-scan register between TDI and TDO. When Extest is selected, all output signal pin values are driven by values shifted into the boundary-scan register and may change only on the falling edge of TCK. Also, when extest is selected, all system input pin states must be loaded into the boundary-scan register on the rising-edge of TCK.
sample/ preload IEEE 1149.1 Required	01	Sample/preload performs two functions: A snapshot of the sample instruction is captured on the rising edge of TCK without interfering with normal operation. The instruction causes boundary-scan register cells associated with outputs to sample the value being driven. • On the falling edge of TCK the data held in the boundary-scan cells is transferred to the slave register cells. Typically the slave latched data is applied to the system outputs via the extest instruction.
idcode IEEE 1149.1 Optional	10	Reserved
bypass IEEE 1149.1 Required	11	Bypass instruction selects the one-bit bypass register between TDI and TDO pins. 0 (binary) is the only instruction that accesses the bypass register. While this instruction is in effect, all other test data registers have no effect on system operation. Test data registers with both test and system functionality perform their system functions when this instruction is selected.

15.3 TAP Test Data Registers

The PI7C7100 contains two test data registers (bypass and boundary-scan). Each test data register selected by the TAP controller is connected serially between TDI and TDO. TDI is connected to the test data register's most significant bit. TDO is connected to the least significant bit. Data is shifted one bit position within the register towards TDO on each rising edge of TCK. While any register is selected, data is transferred from TDI to TDO without inversion. The following sections describe each of the test data registers.

15.4 Bypass Register

The required bypass register, a one-bit shift register, provides the shortest path between TDI and TDO when a bypass instruction is in effect. This allows rapid movement of test data to and from other components on the board. This path can be selected when no test operation is being performed on the PI7C7100.

15.5 Boundary-Scan Register

The boundary-scan register contains a cell for each pin as well as control cells for I/O and the high-impedance pin.

Table 15-2 shows the bit order of the PI7C7100 boundary-scan register. All table cells that contain "Control" select the direction of bidirectional pins or high-impedance output pins. When a "0" is loaded into the control cell, the associated pin(s) are high-impedance or selected as input.

The boundary-scan register is a required set of serial-shiftable register cells, configured in master/slave stages and connected between each of the PI7C7100's pins and on-chip system logic. The VDD, GND, PLL, AGND, AVDD and JTAG pins are NOT in the boundary-scan chain.

The boundary-scan register cells are dedicated logic and do not have any system function. Data may be loaded into the boundary-scan register master cells from the device input pins and output pin-drivers in parallel by the mandatory sample/preload and extest instructions. Parallel loading takes place on the rising edge of TCK.

Data may be scanned into the boundary-scan register serially via the TDI serial input pin, clocked by the rising edge of TCK. When the required data has been loaded into the master-cell stages, it can be driven into the system logic at input pins or onto the output pins on the falling edge of TCK state. Data may also be shifted out of the boundary-scan register by means of the TDO serial output pin at the falling edge of TCK.

15.6 TAP Controller

The TAP (Test Access Port) controller is a 4-state synchronous finite state machine that controls the sequence of test logic operations. The TAP can be controlled via a bus master. The bus master can be either automatic test equipment or a component (i.e., PLD) that interfaces to the TAP. The TAP controller changes state only in response to a rising edge of TCK. The value of the test mode state (TMS) input signal at a rising edge of TCK controls the sequence of state changes. The TAP controller is initialized after power-up by applying a low to the TRST# pin. In addition, the TAP controller can be initialized by applying a high signal level on the TMS input for a minimum of five TCK periods.

For greater detail on the behavior of the TAP controller, test logic in each controller state and the state machine and public instructions, refer to the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture document (available from the IEEE).

Table 15-2. JTAG Boundary Register Order

Order	Pin Names	Type
1	S2_AD[20-31]	control enable
2	S2_AD[21]	output
3	S2_AD[21]	input
4	S2_PERR#	control enable
5	S2_PERR#	output
6	S2_PERR#	input
7	S2_AD[8-19]	control enable
8	S2_AD[16]	output
9	S2_AD[16]	input
10	S2_FRAME#	control enable
11	S2_FRAME#	output
12	S2_FRAME#	input
13	S2_DEVSEL#/S2_TRDY#	control enable
14	S2_DEVSEL#	output
15	S2_DEVSEL#	input
16	S2_AD[19]	output
17	S2_AD[19]	input
18	S2_AD[17]	output
19	S2_AD[17]	input
20	S2_AD[18]	output
21	S2_AD[18]	input
22	S2_AD[20]	output
23	S2_AD[20]	input
24	S2_AD[22]	output
25	S2_AD[22]	input
26	S2_AD[24]	output
27	S2_AD[24]	input
28	S2_AD[23]	output
29	S2_AD[23]	input
30	S2_CBE[0-3]	control enable
31	S2_CBE[3]	output
32	S2_CBE[3]	input
33	S2_AD[25]	output
34	S2_AD[25]	input
35	S2_AD[26]	output
36	S2_AD[26]	input
37	S2_AD[28]	output
38	S2_AD[28]	input
39	S2_AD[27]	output
40	S2_AD[27]	input
41	S2_AD[29]	output
42	S2_AD[29]	input
43	S2_AD[30]	output
44	S2_AD[30]	input
45	S2_AD[31]	output
46	S2_AD[31]	input
47	S2_GNT[0]#	control enable
48	S2_GNT[0]#	output
49	S2_REQ[0]#	input
50	S2_REQ[1]#	input
51	S2_GNT[1]#	output
52	S2_GNT[2]#	output
53	S2_REQ[2]#	input
54	S2_REQ[3]#	input
55	S2_GNT[3]#	output
56	S2_GNT[4]#	output
57	S2_REQ[4]#	input
58	S2_REQ[5]#	input
59	S2_GNT[5]#	output
60	S2_GNT[6]#	output
61	S2_REQ[6]#	input
62	S2_REQ[7]#	input
63	S2_GNT[7]#	output
64	S2_RESET#	output
65	S_CFN#	input
66	S1_EN#	input
67	S2_EN#	input
68	SCAN_TM#	input
69	SCAN_EN	input
70	PLL_TM	input
71	BYPASS	input
72	FLUSH#	input
73	P_RESET#	input
74	P_GNT#	input
75	P_REQ#	control enable
76	P_REQ#	output
77	P_AD[20-31]	control enable
78	P_AD[30]	output
79	P_AD[30]	input
80	P_AD[31]	output
81	P_AD[31]	input
82	P_AD[27]	output
83	P_AD[27]	input
84	P_AD[26]	output
85	P_AD[26]	input
86	P_AD[28]	output
87	P_AD[28]	input
88	P_AD[29]	output
89	P_AD[29]	input
90	P_CBE[0-3]	control enable
91	P_CBE[3]	output
92	P_CBE[3]	input
93	P_AD[24]	output
94	P_AD[24]	input
95	P_AD[25]	output
96	P_AD[25]	input
97	P_AD[23]	output
98	P_AD[23]	input
99	P_AD[22]	output
100	P_AD[22]	input
101	P_IDSEL	input
102	P_AD[21]	output
103	P_AD[21]	input
104	P_AD[20]	output
105	P_AD[20]	input

Table 15-2. JTAG Boundary Register Order (continued)

Order	Pin Names	Type
106	P_AD[8-19]	control enable
107	P_AD[19]	output
108	P_AD[19]	input
109	P_AD[18]	output
110	P_AD[18]	input
111	P_AD[17]	output
112	P_AD[17]	input
113	P_AD[16]	output
114	P_AD[16]	input
115	P_CBE[2]	output
116	P_CBE[2]	input
117	P_FRAME#	control enable
118	P_FRAME#	output
119	P_FRAME#	input
120	P_IRDY#	control enable
121	P_IRDY#	output
122	P_IRDY#	input
123	P_DEVSEL/P_TRDY#	control enable
124	P_TRDY#	output
125	P_TRDY#	input
126	P_DEVSEL#	output
127	P_DEVSEL#	input
128	P_STOP#	control enable
129	P_STOP#	output
130	P_STOP#	input
131	P_PERR#	control enable
132	P_PERR#	output
133	P_PERR#	input
134	P_LOCK#	control enable
135	P_LOCK#	output
136	P_LOCK#	input
137	P_SERR#	control enable
138	P_SERR#	output
139	P_AD[13]	output
140	P_AD[13]	input
141	P_AD[14]	output
142	P_AD[14]	input
143	P_AD[11]	output
144	P_AD[11]	input
145	P_AD[15]	output
146	P_AD[15]	input
147	P_AD[12]	output
148	P_AD[12]	input
149	P_AD[8]	output
150	P_AD[8]	input
151	P_CBE[1]	output
152	P_CBE[1]	input
153	P_AD[9]	output
154	P_AD[9]	input
155	P_AD[0-7]	control enable
156	P_AD[5]	output
157	P_AD[5]	input
158	P_M66EN	input
159	P_AD[6]	output
160	P_AD[6]	input
161	P_AD[2]	output
162	P_AD[2]	input
163	P_PAR	control enable
164	P_PAR	output
165	P_PAR	input
166	P_AD[0]	output
167	P_AD[0]	input
168	P_CBE[0]	output
169	P_CBE[0]	input
170	P_AD[7]	output
171	P_AD[7]	input
172	P_AD[10]	output
173	P_AD[10]	input
174	P_AD[1]	output
175	P_AD[1]	input
176	P_AD[3]	output
177	P_AD[3]	input
178	P_AD[4]	output
179	P_AD[4]	input
180	S1_AD[0-7]	control enable
181	S1_AD[0]	output
182	S1_AD[0]	input
183	S1_AD[1]	output
184	S1_AD[1]	input
185	S1_AD[2]	output
186	S1_AD[2]	input
187	S1_AD[5]	output
188	S1_AD[5]	input
189	S1_AD[3]	output
190	S1_AD[3]	input
191	S1_AD[4]	output
192	S1_AD[4]	input
193	S1_CBE[0-3]	control enable
194	S1_CBE[0]	output
195	S1_CBE[0]	input
196	S1_AD[7]	output
197	S1_AD[7]	input
198	S1_AD[6]	output
199	S1_AD[6]	input
200	S1_AD[8-19]	control enable
201	S1_AD[8]	output
202	S1_AD[8]	input
203	S1_AD[9]	output
204	S1_AD[9]	input
205	S1_AD[10]	output
206	S1_AD[10]	input
207	S1_AD[11]	output
208	S1_AD[11]	input
209	S1_AD[12]	output
210	S1_AD[12]	input

Table 15-2. JTAG Boundary Register Order (continued)

Order	Pin Names	Type
211	S1_AD[14]	output
212	S1_AD[14]	input
213	S1_AD[13]	output
214	S1_AD[13]	input
215	S1_AD[15]	output
216	S1_AD[15]	input
217	S1_SERR#	input
218	S1_PAR	control enable
219	S1_PAR	output
220	S1_PAR	input
221	S1_CBE[1]	output
222	S1_CBE[1]	input
223	S1_DEVSEL#/S1_TRDY#	control enable
224	S1_DEVSEL#	output
225	S1_DEVSEL#	input
226	S1_STOP#	control enable
227	S1_STOP#	output
228	S1_STOP#	input
229	S1_LOCK#	control enable
230	S1_LOCK#	output
231	S1_LOCK#	input
232	S1_PERR#	control enable
233	S1_PERR#	output
234	S1_PERR#	input
235	S1_FRAME#	control enable
236	S1_FRAME#	output
237	S1_FRAME#	input
238	S1_IRDY#	control enable
239	S1_IRDY#	output
240	S1_IRDY#	input
241	S1_TRDY#	output
242	S1_TRDY#	input
243	S1_AD[17]#	output
244	S1_AD[17]#	input
245	S1_AD[16]#	output
246	S1_AD[16]#	input
247	S1_AD[20-31]	control enable
248	S1_AD[20]	output
249	S1_AD[20]	input
250	S1_CBE[2]	output
251	S1_CBE[2]	input
252	S1_AD[19]	output
253	S1_AD[19]	input
254	S1_CBE[3]	output
255	S1_CBE[3]	input
256	S1_AD[23]	output
257	S1_AD[23]	input
258	S1_AD[26]	output
259	S1_AD[26]	input
260	S1_AD[22]	output
261	S1_AD[22]	input
262	S1_AD[25]	output
263	S1_AD[25]	input
264	S1_AD[29]	output
265	S1_AD[29]	input
266	S1_AD[21]	output
267	S1_AD[21]	input
268	S1_AD[28]	output
269	S1_AD[28]	input
270	S1_AD[30]	output
271	S1_AD[30]	input
272	S1_AD[31]	output
273	S1_AD[31]	input
274	S1_AD[27]	output
275	S1_AD[27]	input
276	S1_AD[24]	output
277	S1_AD[24]	input
278	S1_AD[18]	output
279	S1_AD[18]	input
280	S1_GNT[0]#	control enable
281	S1_GNT[0]#	output
282	S1_REQ[0]#	input
283	S1_REQ[1]#	input
284	S1_GNT[1]#	output
285	S1_GNT[2]#	output
286	S1_REQ[2]#	input
287	S1_REQ[3]#	input
288	S1_GNT[3]#	output
289	S1_GNT[4]#	output
290	S1_REQ[4]#	input
291	S1_REQ[5]#	input
292	S1_GNT[5]#	output
293	S1_GNT[6]#	output
294	S1_REQ[6]#	input
295	S1_REQ[7]#	input
296	S1_GNT[7]#	output
297	S1_RESET#	output
298	S2_AD[0-7]	control enable
299	S2_AD[0]	output
300	S2_AD[0]	input
301	S2_AD[1]	output
302	S2_AD[1]	input
303	S2_AD[2]	output
304	S2_AD[2]	input
305	S2_AD[3]	output
306	S2_AD[3]	input
307	S2_AD[4]	output
308	S2_AD[4]	input
309	S2_AD[5]	output
310	S2_AD[5]	input
311	S2_AD[6]	output
312	S2_AD[6]	input
313	S2_AD[7]	output
314	S2_AD[7]	input
315	S2_CBE[0]	output

Table 15-2. JTAG Boundary Register Order (continued)

Order	Pin Names	Type
316	S2_CBE[0]	input
317	S2_AD[8]	output
318	S2_AD[8]	input
319	S2_AD[10]	output
320	S2_AD[10]	input
321	S2_AD[9]	output
322	S2_AD[9]	input
323	S2_AD[11]	output
324	S2_AD[11]	input
325	S_M66EN	input
326	S2_AD[12]	output
327	S2_AD[12]	input
328	S2_AD[14]	output
329	S2_AD[14]	input
330	S2_CBE[1]	output
331	S2_CBE[1]	input
332	S2_AD[15]	output
333	S2_AD[15]	input
334	S2_PAR	control enable

Order	Pin Names	Type
335	S2_PAR	output
336	S2_PAR	input
337	S2_SERR#	input
338	S2_LOCK#	control enable
339	S2_LOCK#	output
340	S2_LOCK#	input
341	S2_TRDY#	output
342	S2_TRDY#	input
343	S2_STOP#	control enable
344	S2_STOP#	output
345	S2_STOP#	input
346	S2_IRDY#	control enable
347	S2_IRDY#	output
348	S2_IRDY#	input
349	S2_CBE[2]	output
350	S2_CBE[2]	input
351	S2_AD[13]	output
352	S2_AD[13]	input

16. Electrical and Timing Specifications

16.1 Maximum Ratings

(Above which the useful life may be impaired. For user guidelines, not tested).

Storage Temperature	–65°C to +150°C
Ambient Temperature with Power applied	0°C to +70°C
Supply Voltage to Ground Potentials (Inputs & AV_{CC}, V_{DD} only)	–0.3V to +3.6V
DC Input Voltage	–0.5V to +3.6V

Note:
Stresses greater than those listed under MAXIMUM RATINGS may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

16.2 3.3V DC Specifications

Symbol	Parameter	Condition	Min.	Max.	Units	Notes
V_{DD}, AV_{CC}	Supply Voltage		3	3.6	V	
V_{ih}	Input HIGH Voltage		0.5 V_{DD}	V_{DD} + 0.5		3
V_{il}	Input LOW Voltage		-0.5	0.3 V_{DD}		
V_{ih}	CMOS Input HIGH Voltage		0.7 V_{DD}	V_{DD} + 0.5		1
V_{il}	CMOS Input LOW Voltage		–0.5	0.3 V_{DD}		
V_{ipu}	Input Pull-up Voltage		0.7 V_{DD}			3
I_{il}	Input Leakage Current	$0 < V_{in} < V_{DD}$		±10	µA	
V_{oh}	Output HIGH Voltage	I_{out} = –500µA	0.9 V_{DD}		V	
V_{ol}	Output LOW Voltage	I_{out} = 1500µA		0.1 V_{DD}		
V_{oh}	CMOS Output HIGH Voltage	I_{out} = –500µA	V_{DD}-0.5			2
V_{ol}	CMOS Output LOW Voltage	I_{out} = 1500µA		0.5		
C_{in}	Input Pin Capacitance			10	pF	3
C_{clk}	CLK Pin Capacitance		5	12		
C_{IDSEL}	IDSEL Pin Capacitance			8		
L_{pin}	Pin Inductance			20	nH	

Notes:

1. CMOS Input pins: S_CFN#, TCK, TMS, TDI, TRST#, SCAN_EN, SCAN_TM#
2. CMOS Output pin: TDO
3. PCI pins: P_AD[31:0], P_CBE[3:0], P_PAR, P_FRAME#, P_IRDY#, P_TRDY#, P_DEVSEL#, P_STOP#, P_LOCK#, PIDSEL#, P_PERR#, P_SERR#, P_REQ#, P_GNT#, P_RESET#, S1_AD[31:0], S2_AD[31:0], S1_CBE[3:0], S2_CBE[3:0], S1_PAR, S2_PAR, S1_FRAME#, S2_FRAME#, S1_IRDY#, S2_IRDY#, S1_TRDY#, S2_TRDY#, S1_DEVSEL#, S2_DEVSEL#, S1_STOP#, S2_STOP#, S1_LOCK#, S2_LOCK#, S1_PERR#, S2_PERR#, S1_SERR#, S2_SERR#, S1_REQ[7:0]#, S2_REQ[7:0]#, S1_GNT[7:0]#, S2_GNT[7:0], S1_RESET#, S2_RESET#, S1_EN, S2_EN, P_FLUSH#.

16.3 3.3V AC Specifications

(For P_AD[31:0], P_CBE[3:0], P_PAR, P_FRAME#, P_IRDY#, P_TRDY#, P_DEVSEL#, P_STOP#, P_LOCK#, PIDSEL#, P_PERR#, P_SERR#, P_REQ#, P_GNT#, P_RESET#, S1_AD[31:0], S2_AD[31:0], S1_CBE[3:0], S2_CBE[3:0], S1_PAR, S2_PAR, S1_FRAME#, S2_FRAME#, S1_IRDY#, S2_IRDY#, S1_TRDY#, S2_TRDY#, S1_DEVSEL#, S2_DEVSEL#, S1_STOP#, S2_STOP#, S1_LOCK#, S2_LOCK#, S1_PERR#, S2_PERR#, S1_SERR#, S2_SERR#, S1_REQ[7:0]#, S2_REQ[7:0]#, S1_GNT[7:0]#, S2_GNT[7:0]#, S1_RESET#, S2_RESET#, S1_EN, S2_EN, P_FLUSH#)

Symbol	Parameter	Condition	Min.	Max.	Units	Notes
$I_{oh(AC)}$	Switching	$0V < V_{out} < 0.3\ V_{DD}$	3		mA	
	Current HIGH	$0.3\ V_{DD} < Vout < 0.9\ V_{DD}$	$0.5\ V_{DD}$		mA	
		$0.7\ V_{DD} < V_{out} < V_{DD}$	-0.5	Eqt'n C		
	(Test Point)	$V_{out} = 0.7\ V_{DD}$	$0.7\ V_{DD}$	$-32\ V_{DD}$	mA	
$I_{ol(AC)}$	Switching	$V_{DD} > V_{out} > 0.6\ V_{DD}$	-0.5		mA	
	Current LOW	$0.6\ V_{DD} > V_{out} > 0.1\ V_{DD}$	$0.7\ V_{DD}$		mA	
		$0.18\ V_{DD} > V_{out} > 0V$		Eqt'n D		
	(Test Point)	$V_{out} = 0.18\ V_{DD}$	$0.9\ V_{DD}$	$38\ V_{DD}$	mA	
I_{cl}	LOW Clamp Current	$-3V < V_{in} < -1V$			mA	
I_{ch}	HIGH Clamp Current	$V_{DD} + 4 > V_{in} > V_{DD} + 1$	V_{DD}-0.5		mA	
$Slew_r$	Output Rise Slew Rate	$0.2\ V_{DD}$ to $0.6\ V_{DD}$ load		4	V/ns	
$Slew_f$	Output Fall Slew Rate	$0.6\ V_{DD}$ to $0.2V_{DD}$ load		4	V/ns	

Notes:
Equation C: $I_{oh} = (98/V_{DD})*(Vout-V_{DD})*(Vout+0.4V_{DD})$ for $V_{DD}>Vout>0.7V_{DD}$
Equation D: $I_{ol} = (256/V_{DD})*Vout*(V_{DD}-Vout)$ for $0V<Vout<0.18V_{DD}$

16.4 Primary and Secondary Buses at 33 MHz Clock Timing

Symbol	Parameter	Condition	Min.	Max.	Units	Notes
TSKEW	SKEW among S_CLKOUT[15:0]		0	1.0	ns	
TDELAY	DELAY between PCLK and S_CLKOUT[15:0]	20pF load	0	10		
TCYCLE	PCLK, S_CLKOUT[15:0] cycle time		30			
THIGH	PCLK, S_CLKOUT[15:0] HIGH time		11			
TLOW	PCLK, S_CLKOUT[15:0] LOW time		11			

17. 256-Pin PBGA Package

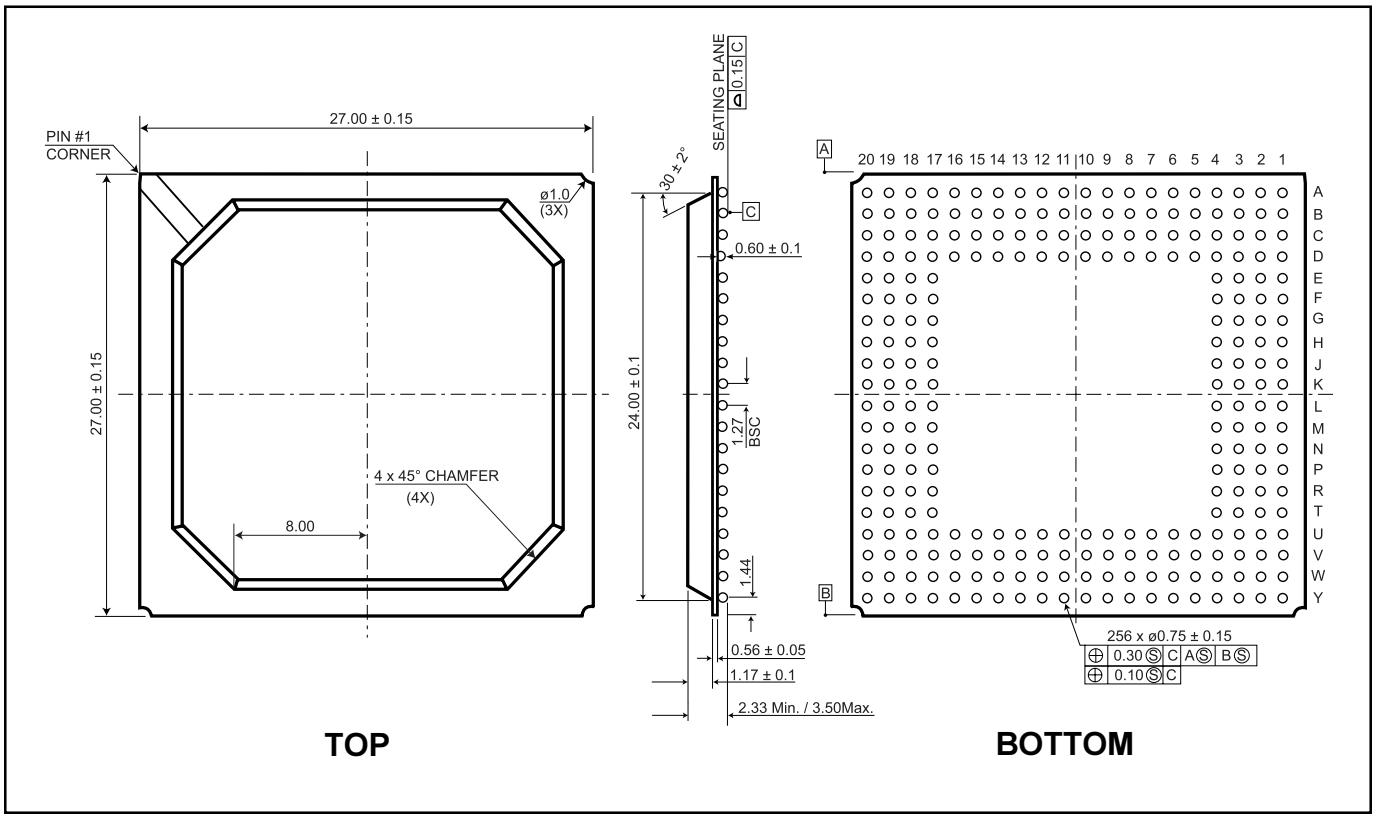


Figure 17-1. 256-Pin PBGA Package

17.1 Part Number Ordering Information

Part	Pin - Package	Temperature
PI7C7100BNA	256 - PBGA	0°C to +70°C

PI7C7100 3-Port PCI Bridge

Appendix A

Timing Diagrams

Figure 1. Configuration Read Transaction

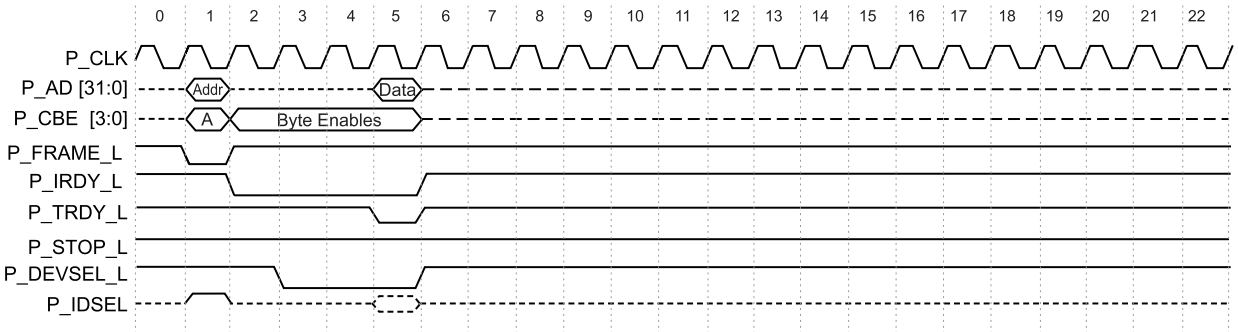


Figure 2. Configuration Write Transaction

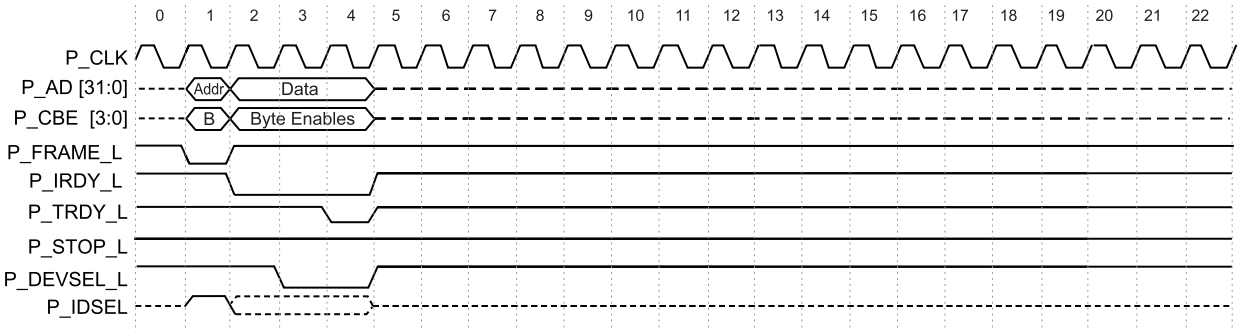


Figure 3. Type 1 to Type 0 Configuration Read Transaction (P --> S)

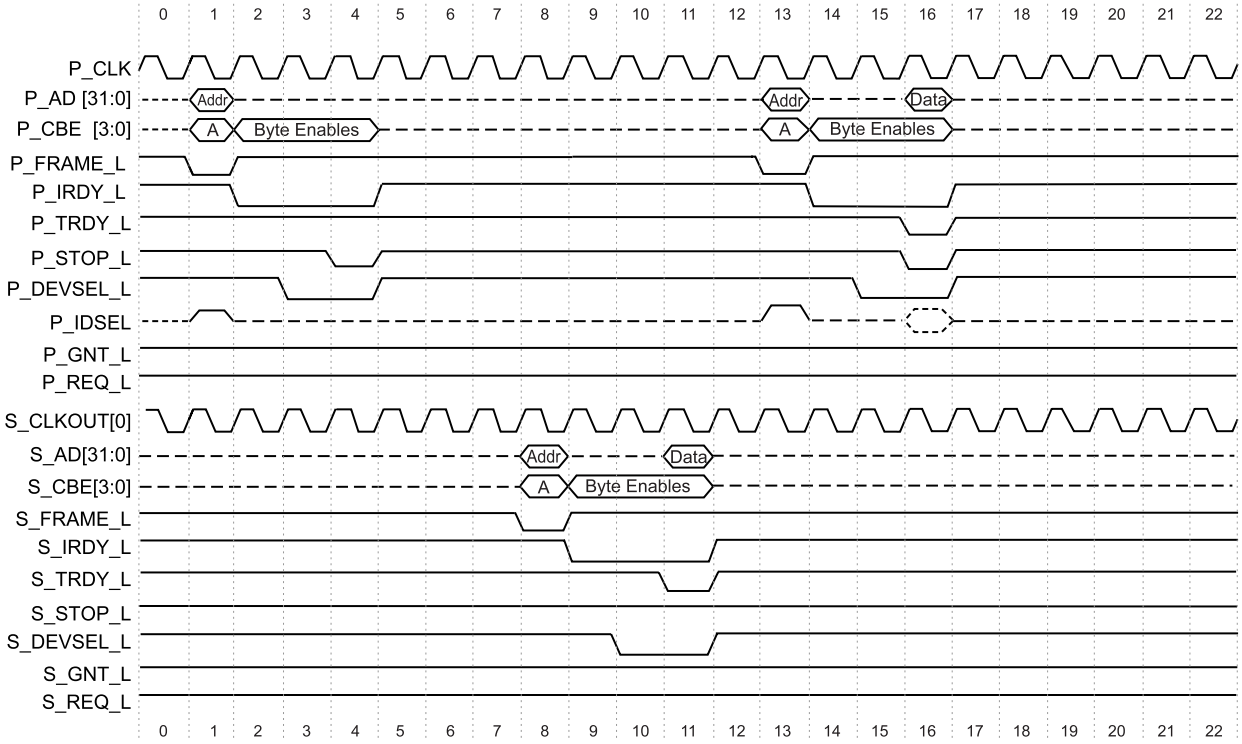


Figure 4. Type 1 to Type 0 Configuration Write Transaction (P --> S)

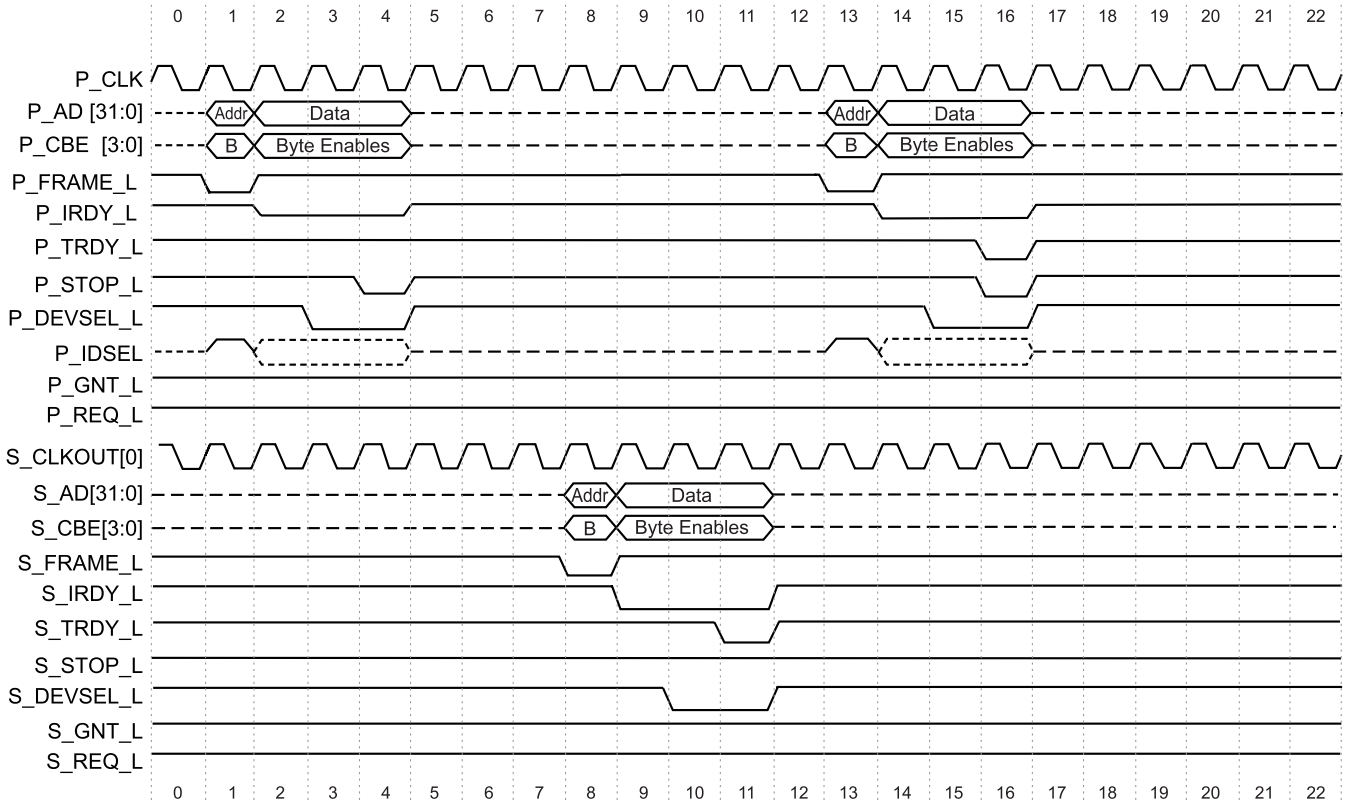


Figure 5. Upstream Type 1 to Special Cycle Transaction (S --> P)

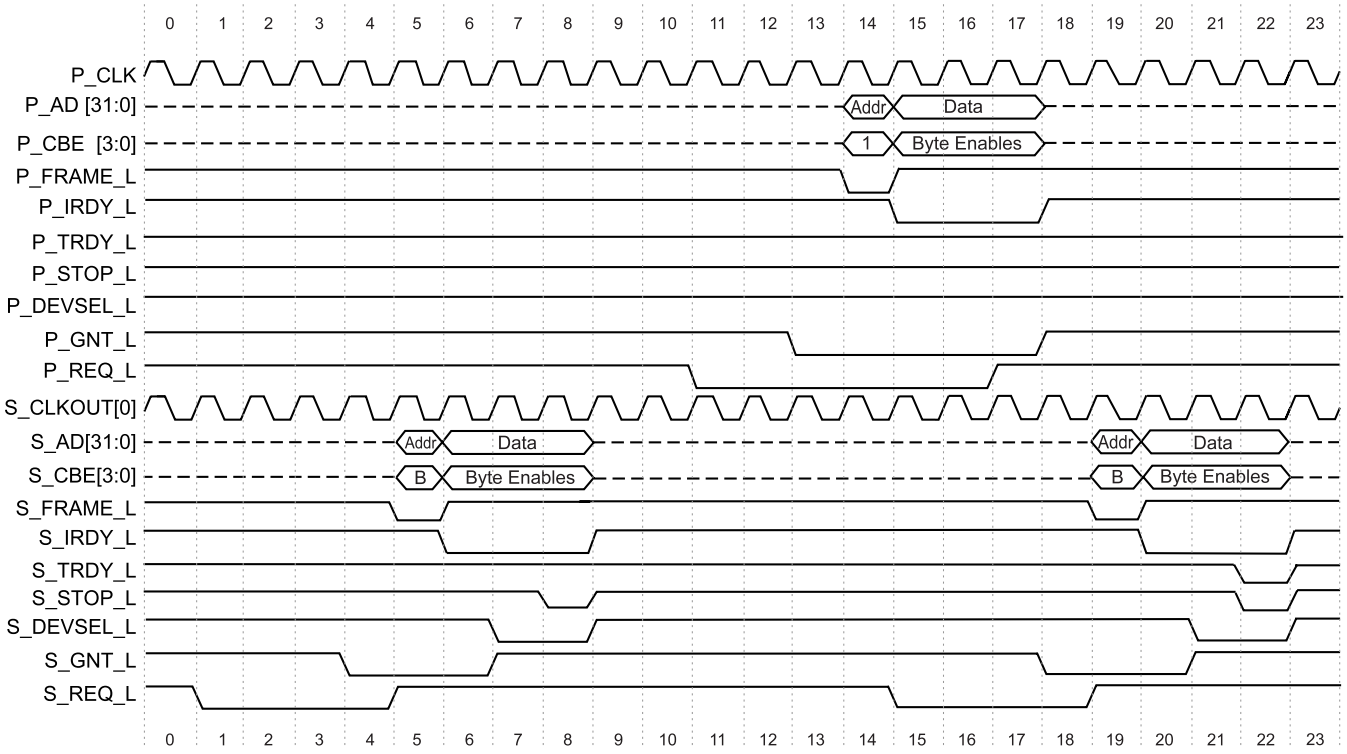


Figure 6. Downstream Type 1 to Special Cycle Transaction (P --> S)

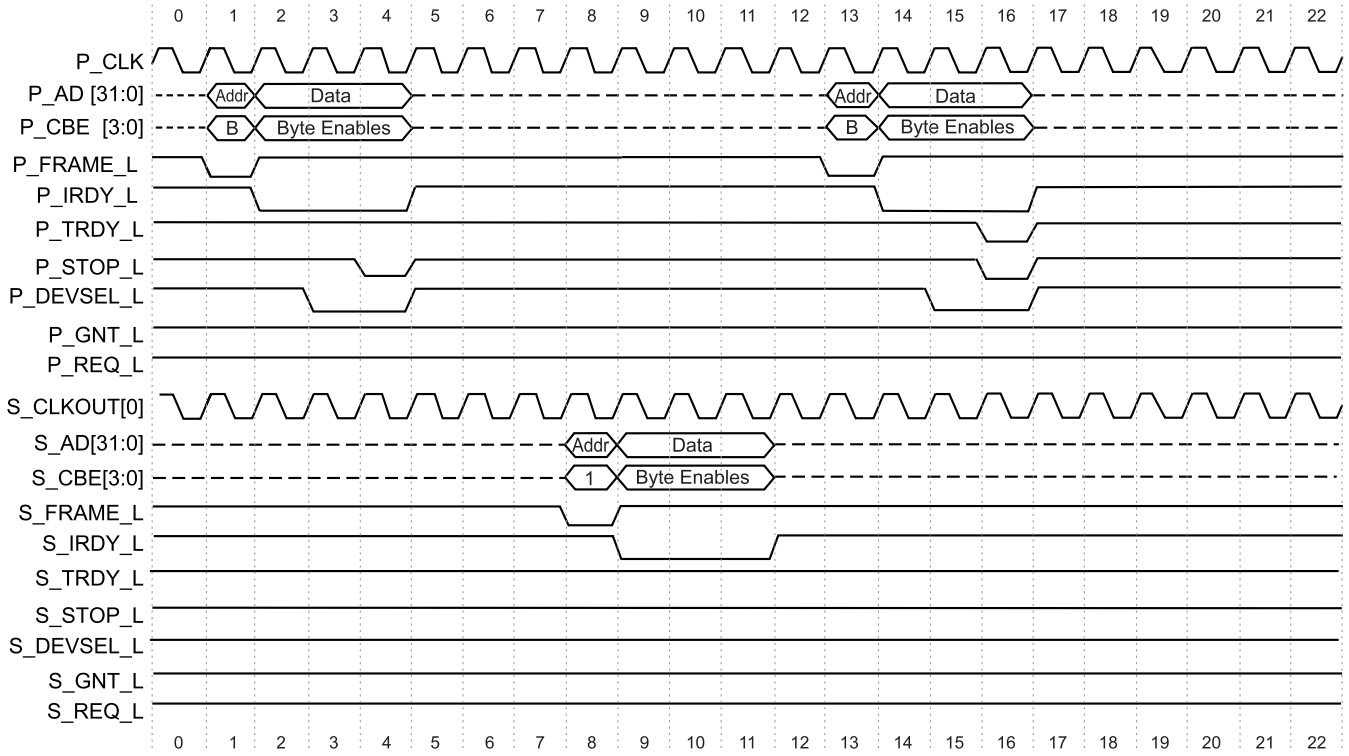


Figure 7. Downstream Type1 to Type1 Configuration Read Transaction (P --> S)

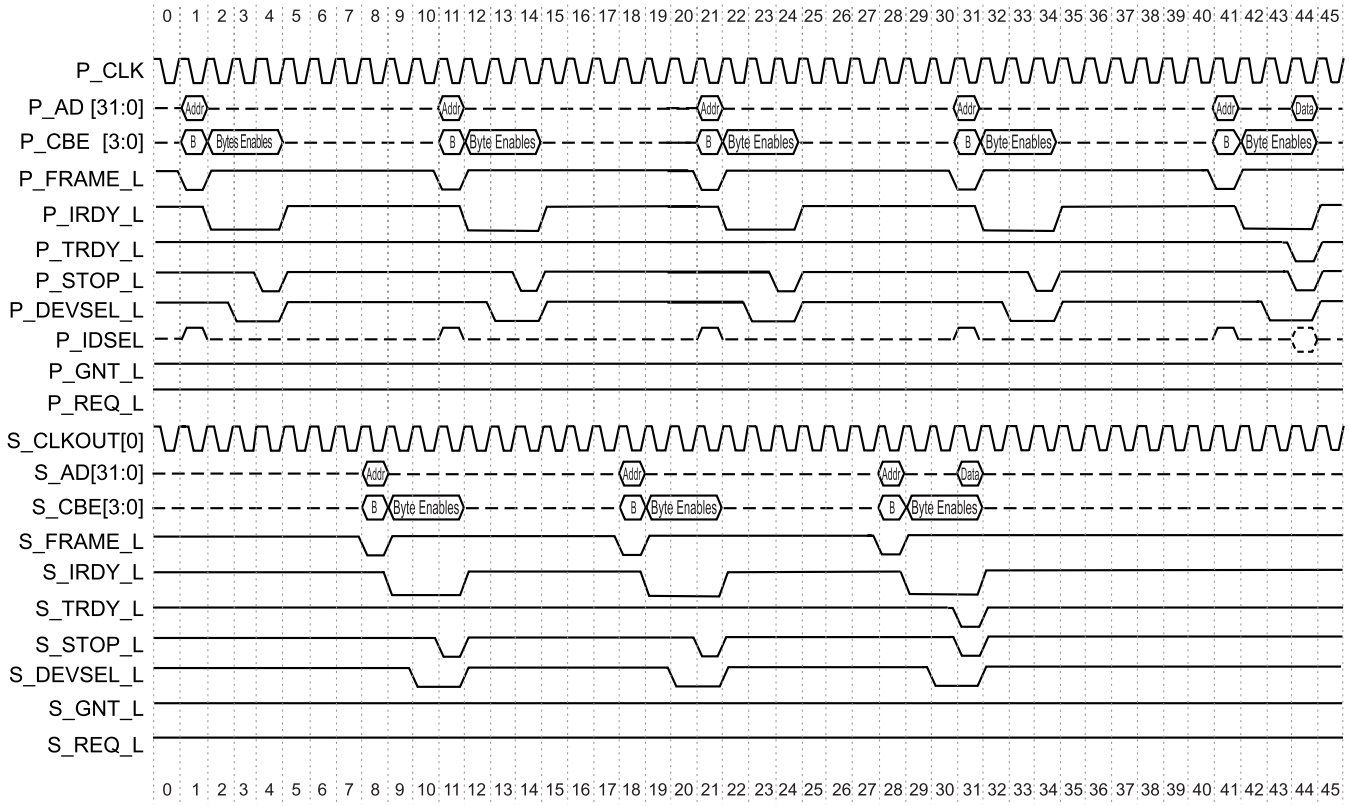


Figure 8. Downstream Type1 to Type1 Configuration Write Transaction (P --> S)

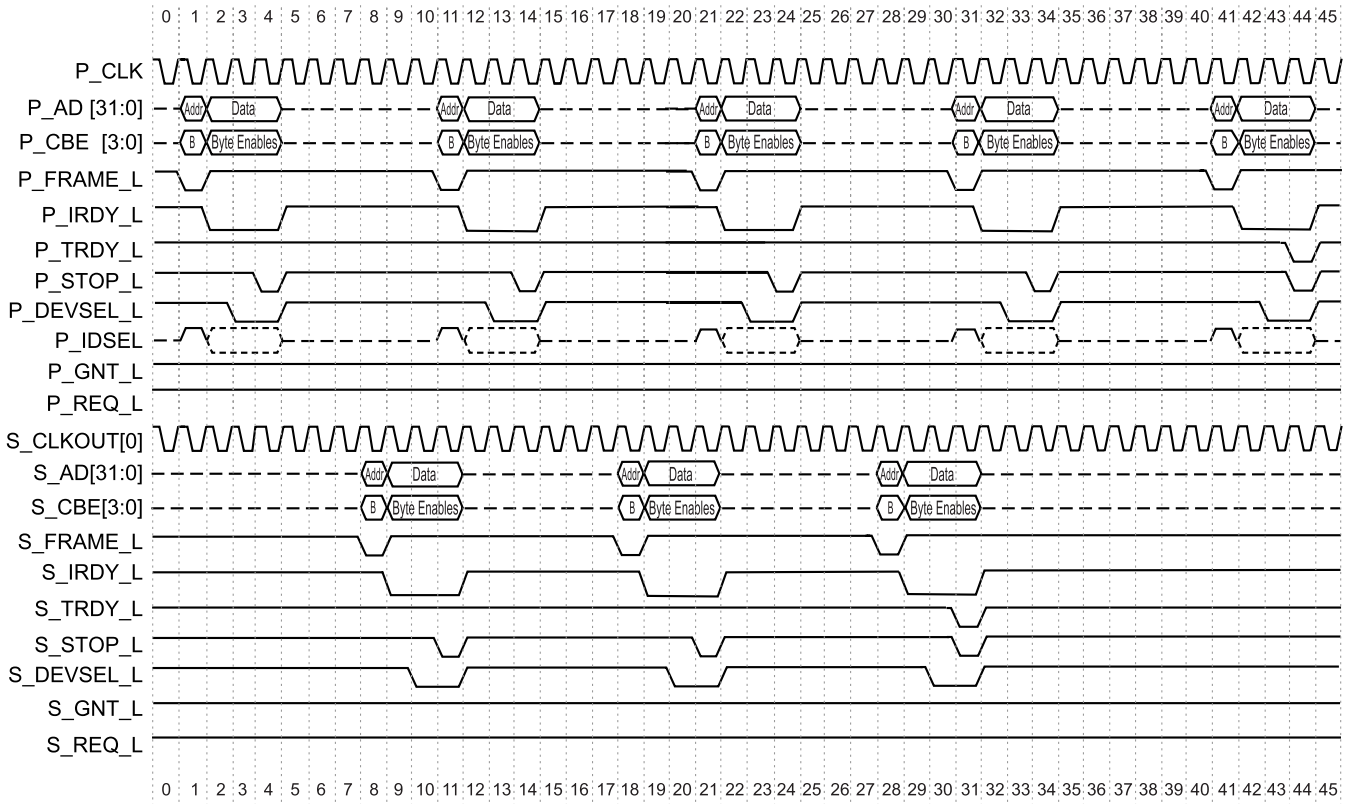


Figure 9. Upstream Delayed Burst Memory Read Transaction (S --> P)

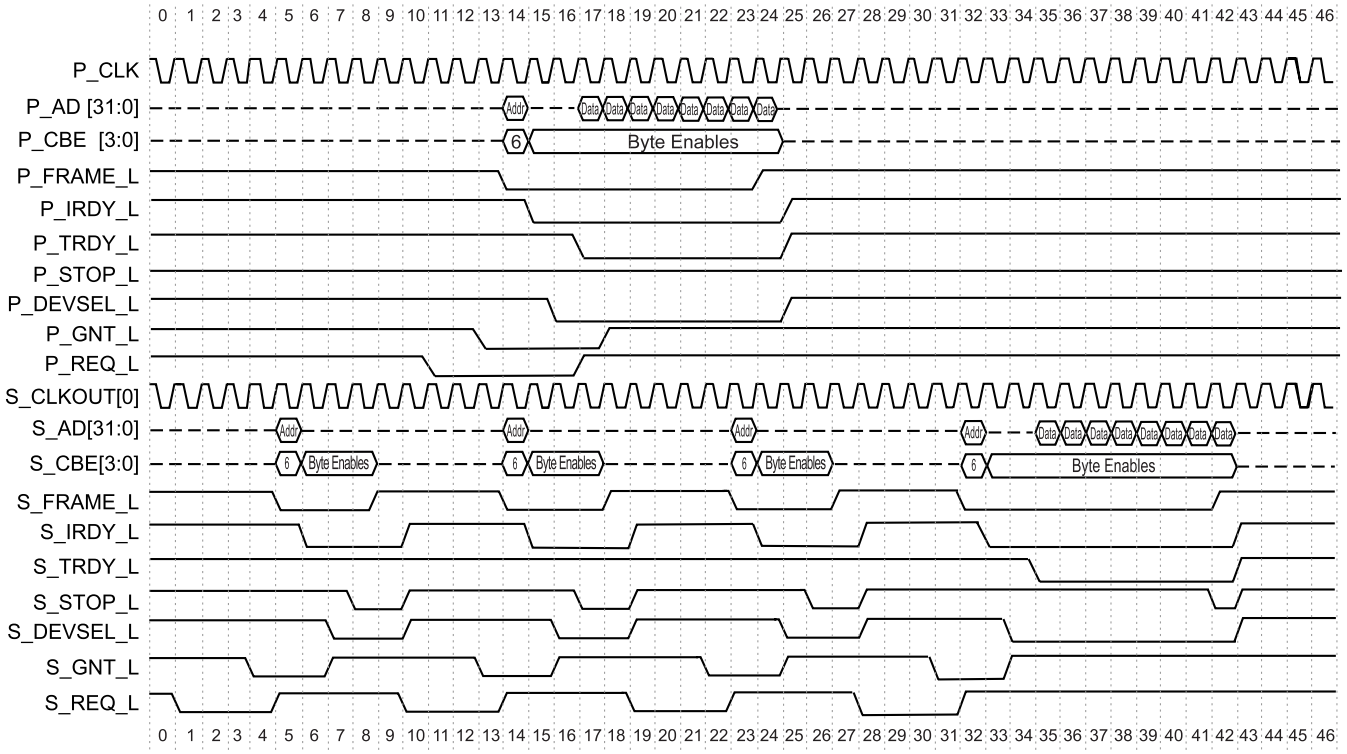


Figure 10. Downstream Delayed Burst Memory Read Transaction (P --> S)

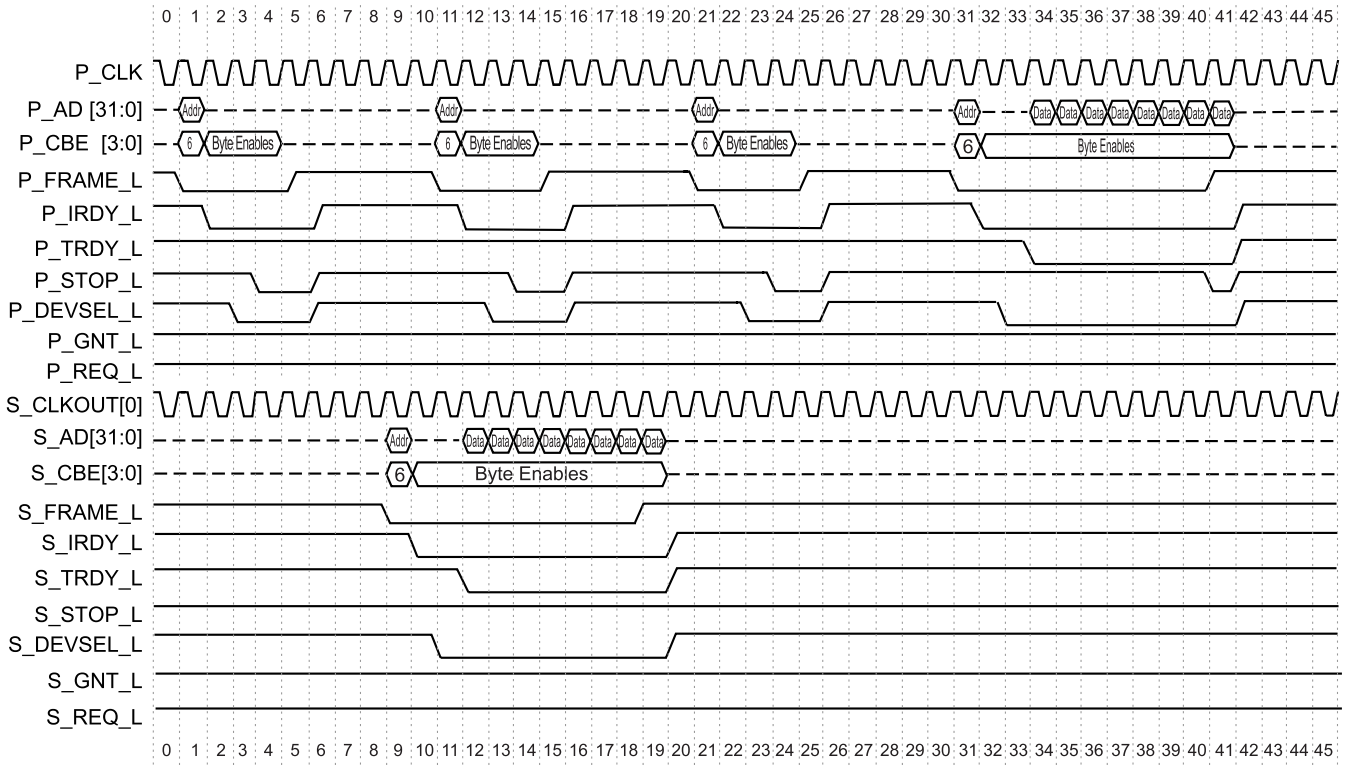


Figure 11. Downstream Delayed Memory Read Transaction (P/33MHz-->S/33MHz)

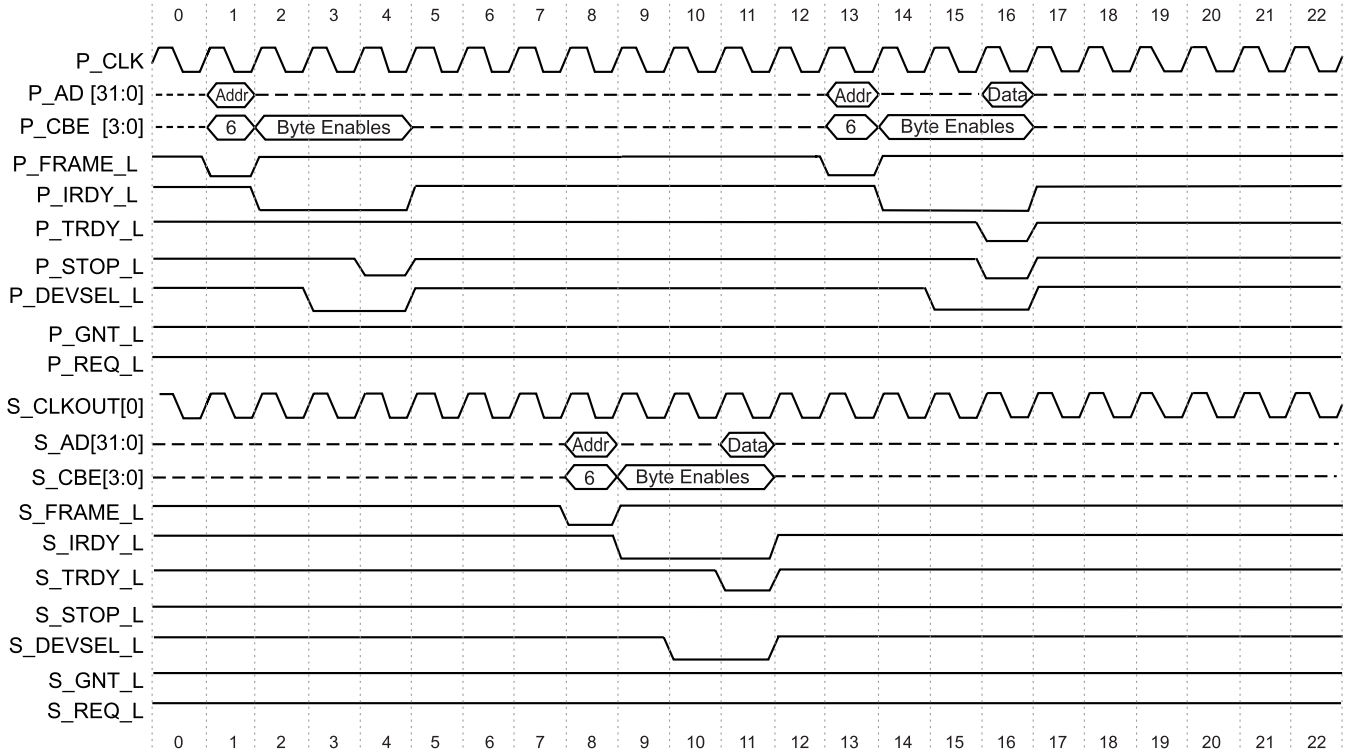


Figure 12. Downstream Delayed Memory Read Transaction (S2/33MHz-->S1/33MHz)

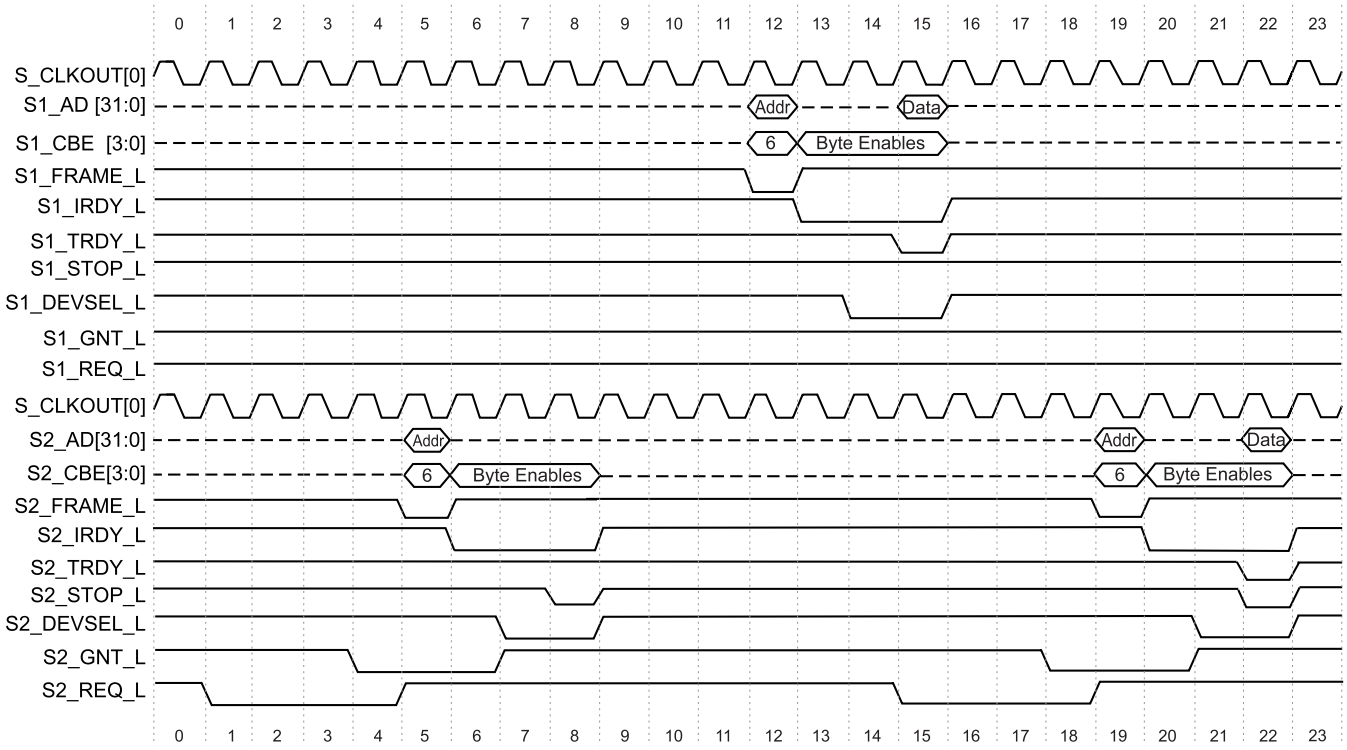


Figure 13. Downstream Delayed Memory Read Transaction (S1/33MHz-->S2/33MHz)

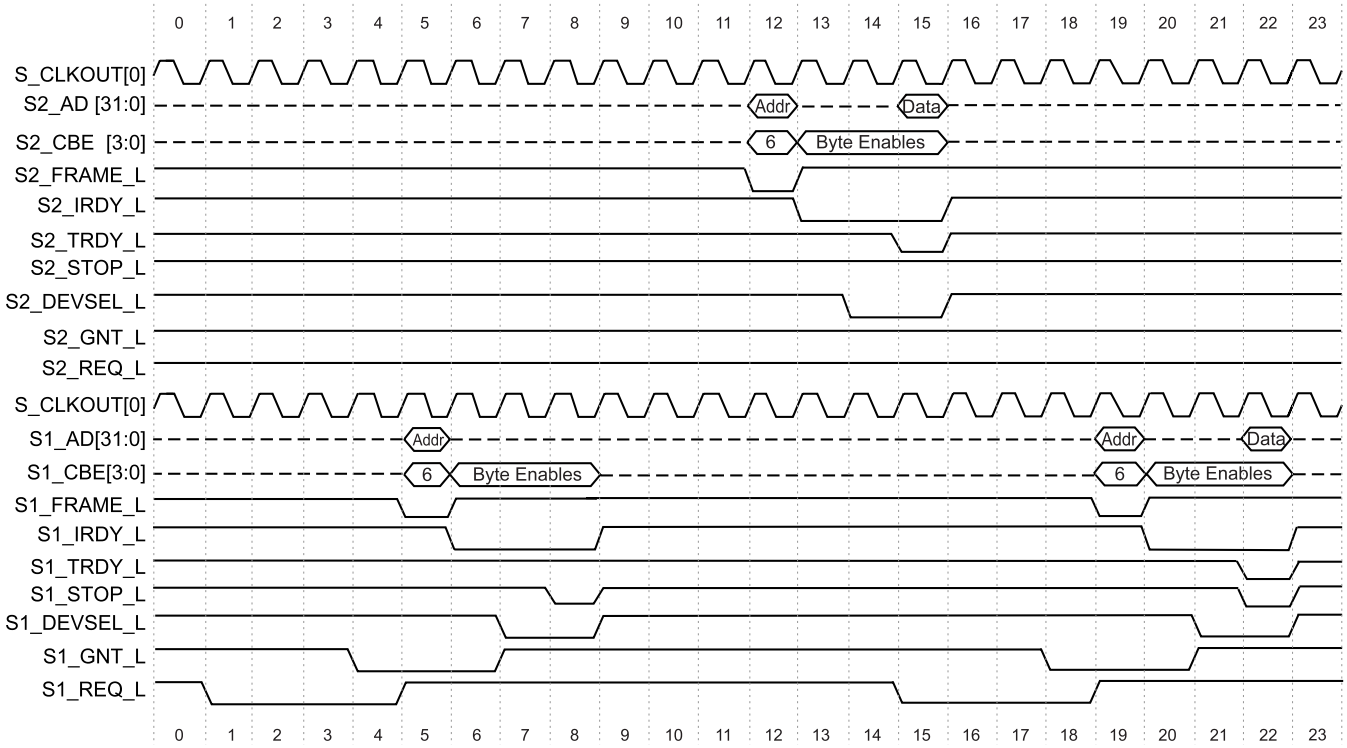


Figure 14. Upstream Delayed Memory Read Transaction (S/33MHz-->P/33MHz)

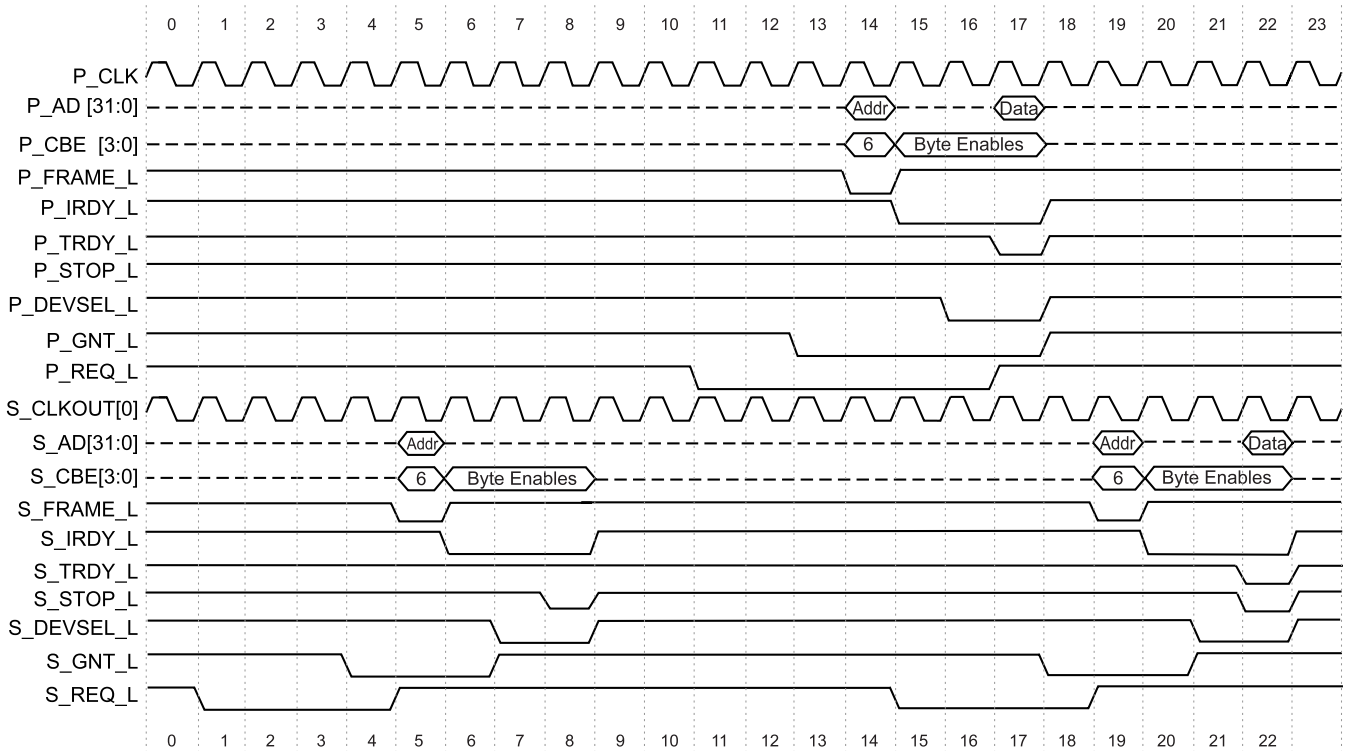


Figure 15. Downstream Posted Memory Write Transaction (P/33MHz-->S/33MHz)

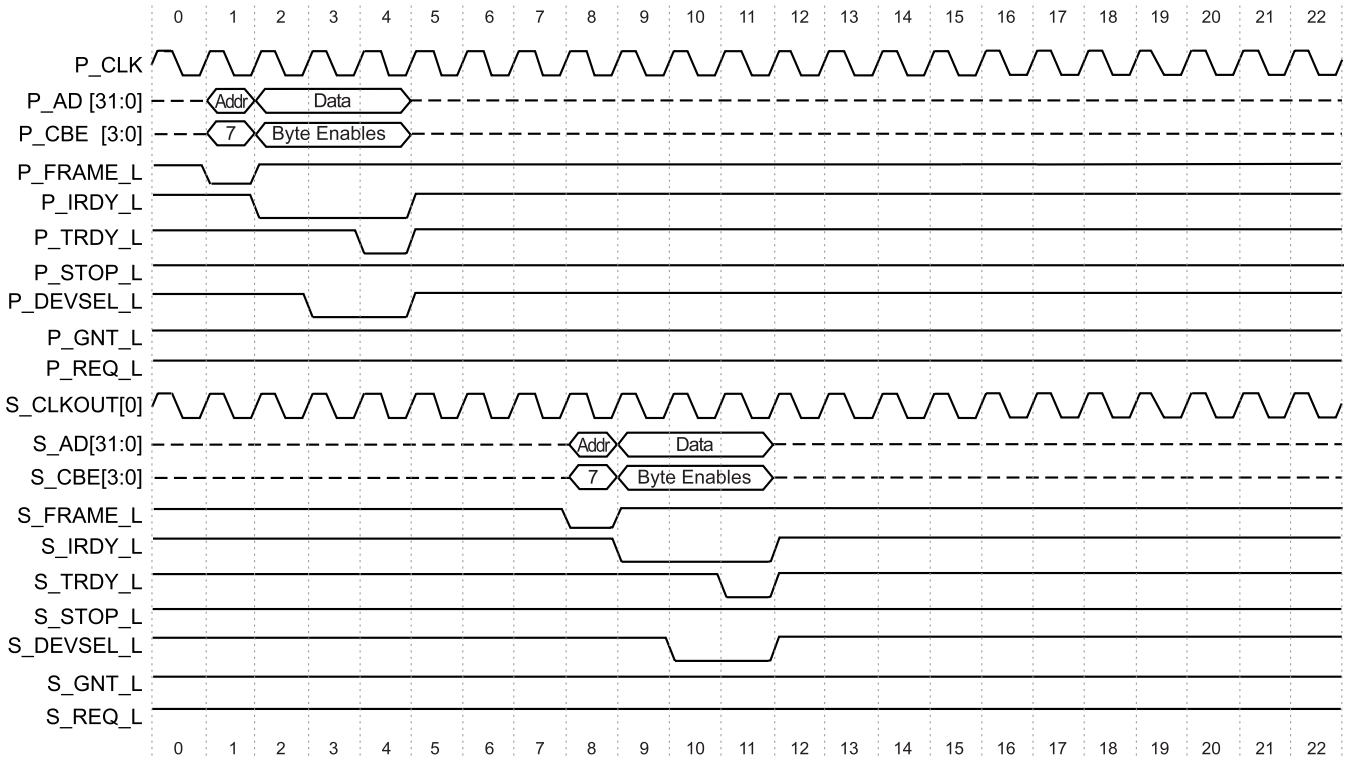


Figure 16. Downstream Posted Memory Write Transaction (S2/33MHz-->S1/33MHz)

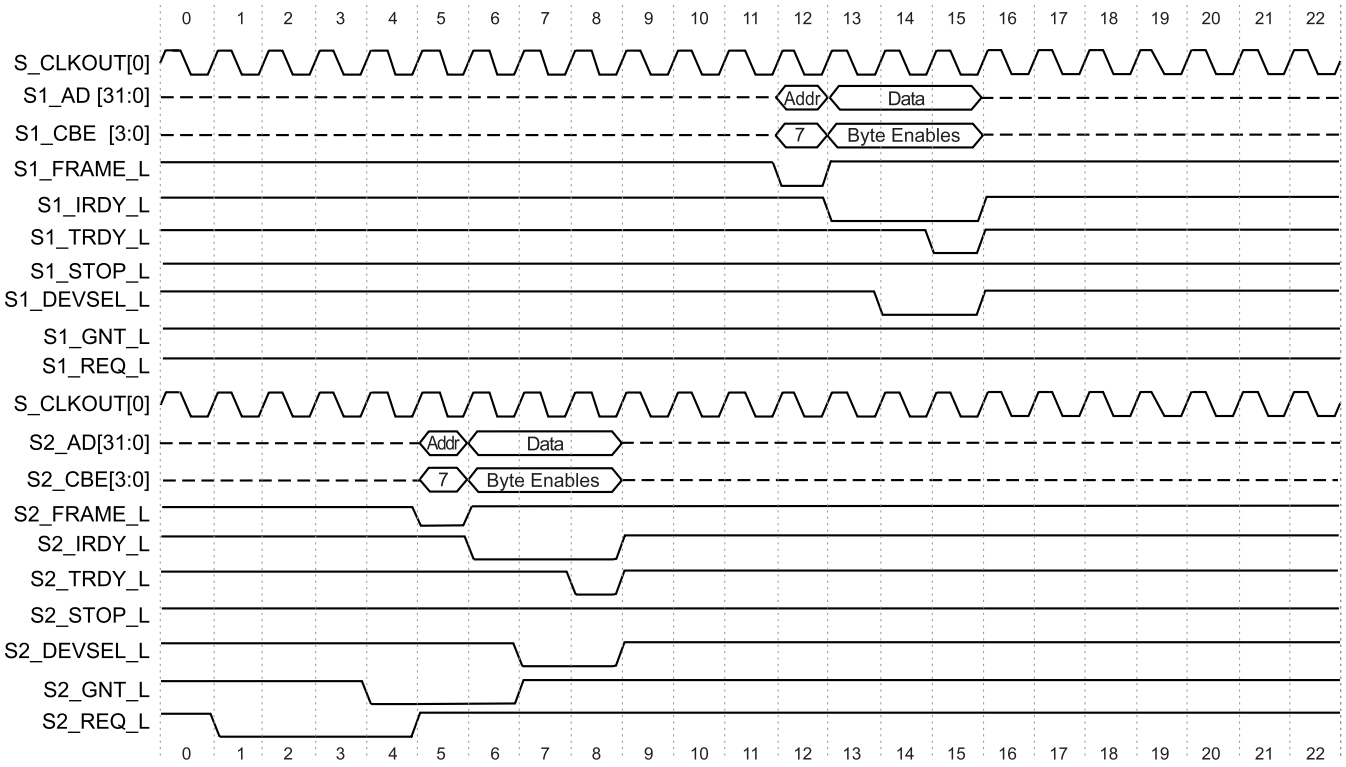


Figure 17. Downstream Posted Memory Write Transaction (S1/33MHz-->S2/33MHz)

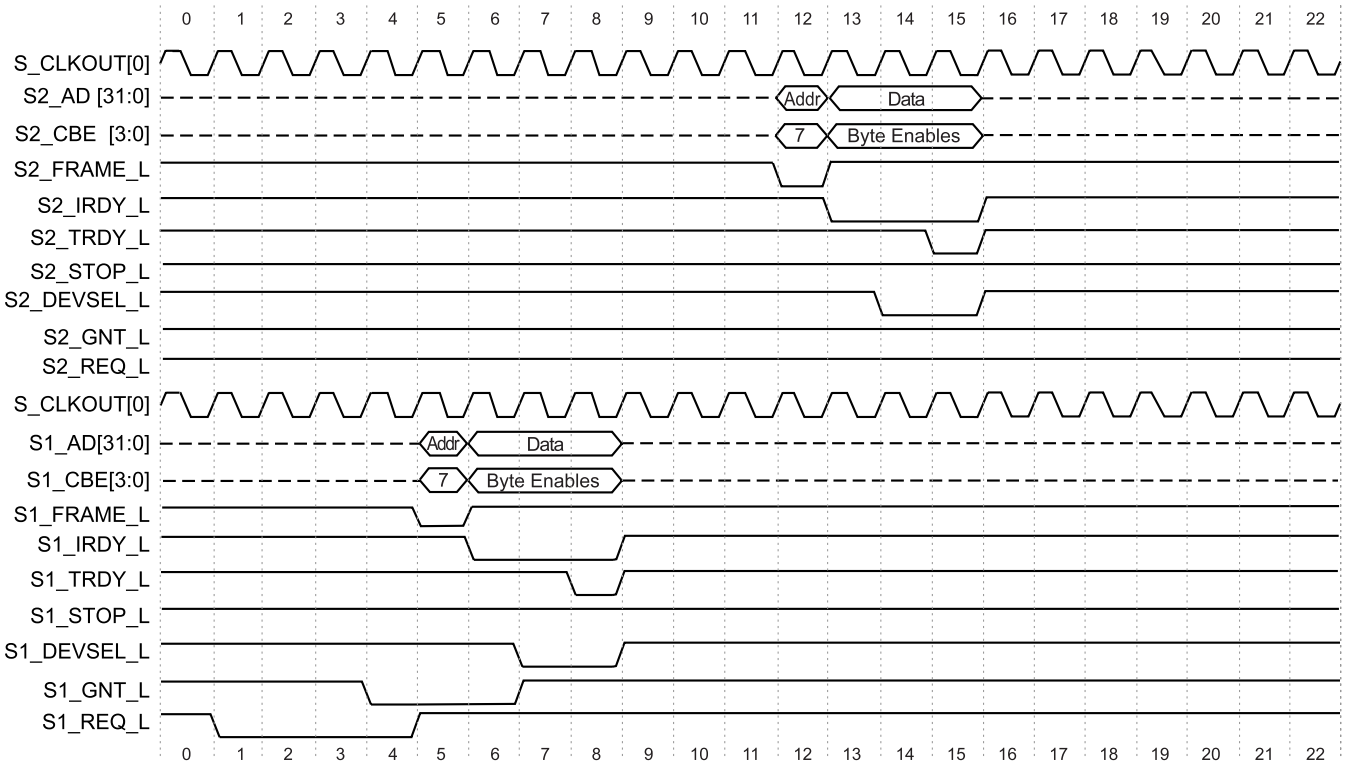


Figure 18. Upstream Posted Memory Write Transaction (S/33MHz-->P/33MHz)

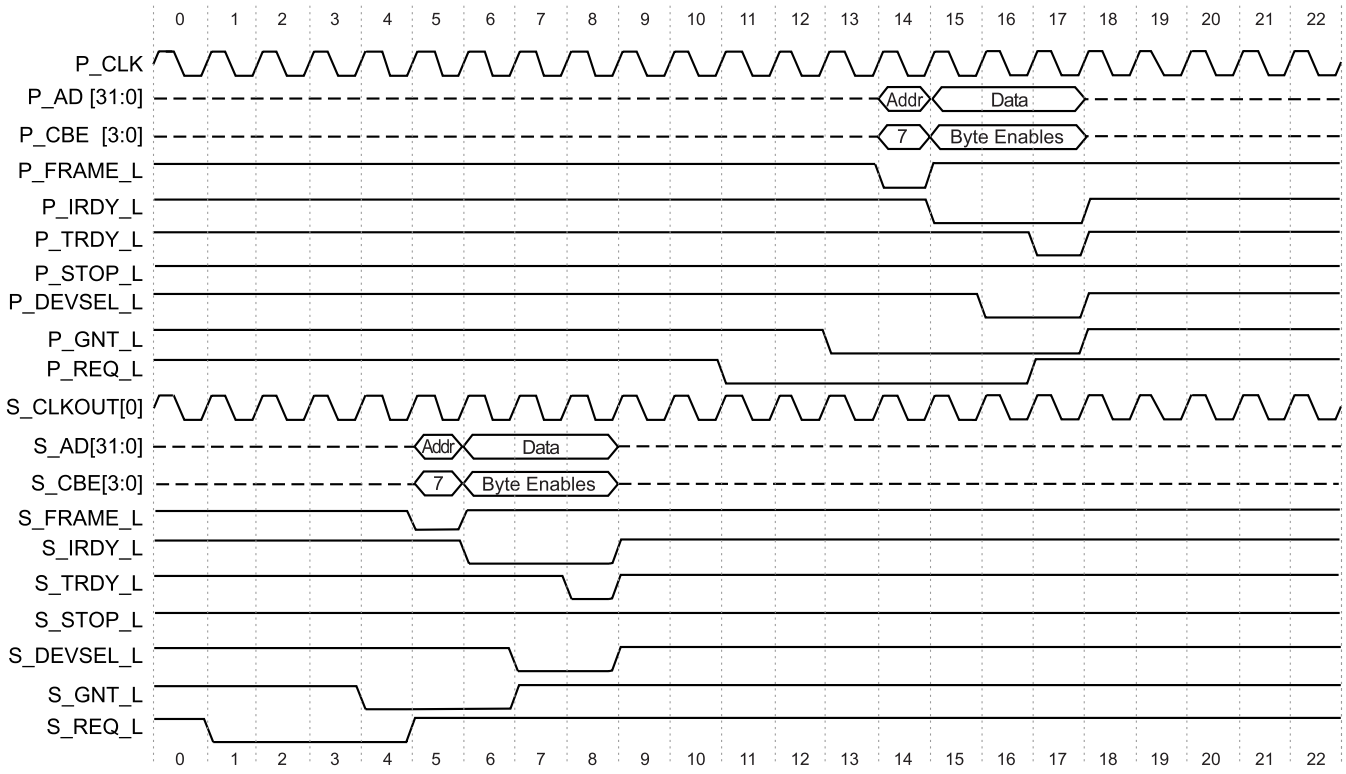


Figure 19. Downstream Flow-Through Posted Memory Write Transaction (P/33MHz-->S/33MHz)

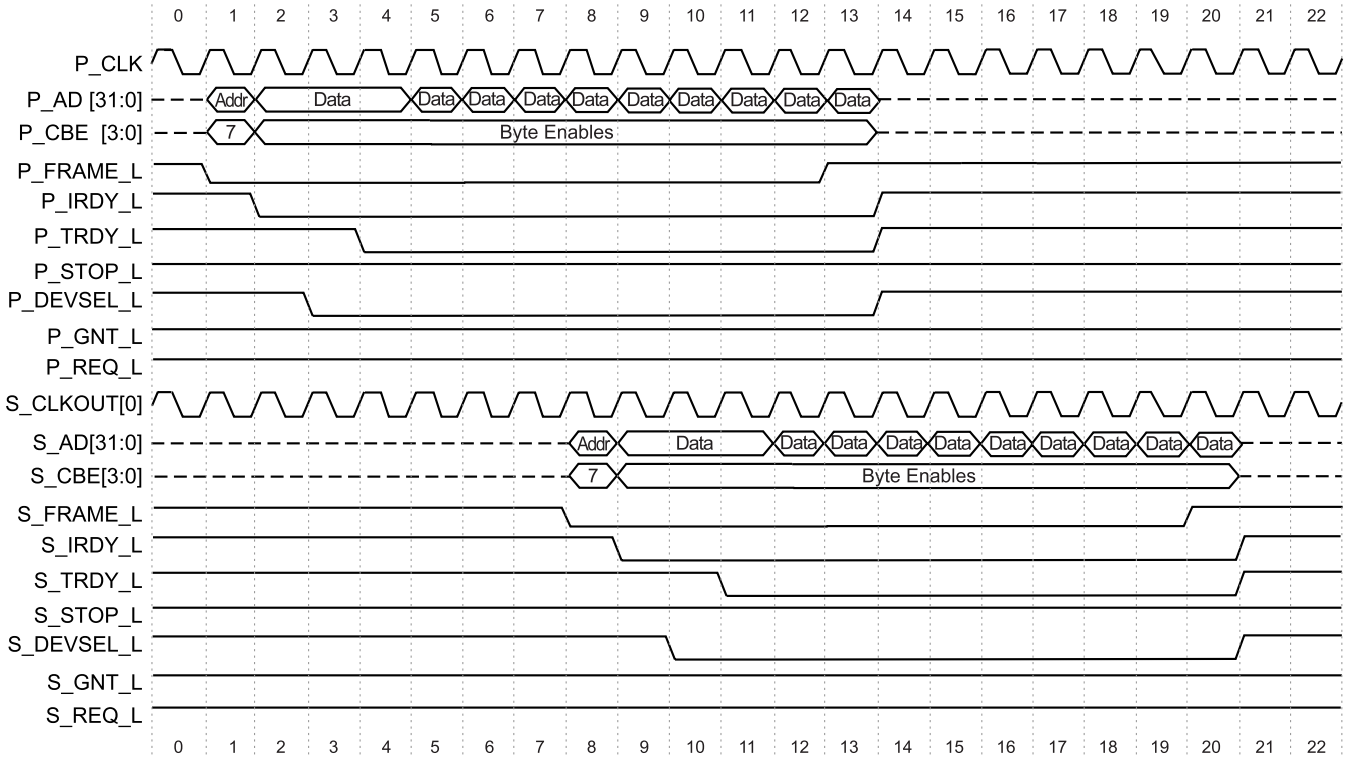


Figure 20. Downstream Flow-Through Posted Memory Write Transaction (S2/33MHz-->S1/33MHz)

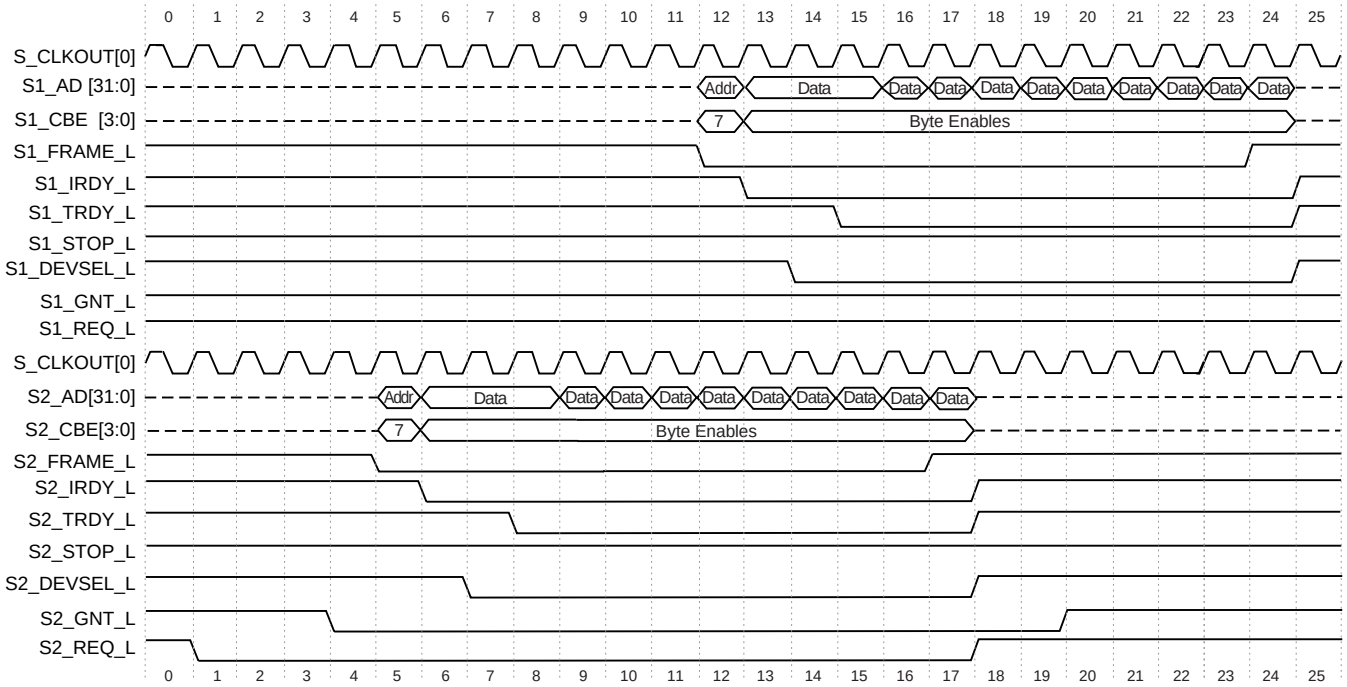


Figure 21. Downstream Flow-Through Posted Memory Write Transaction (S1/33MHz-->S2/33MHz)

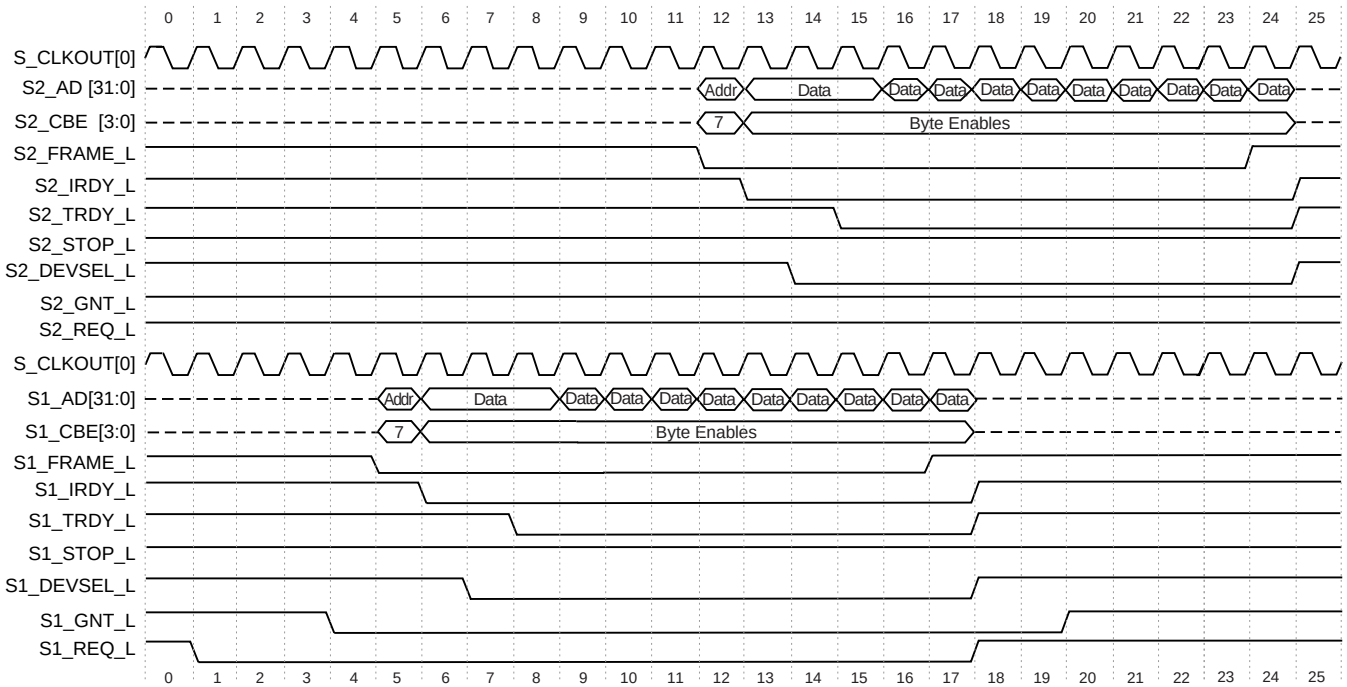


Figure 22. Upstream Flow-Through Posted Memory Write Transaction (S/33MHz-->P/33MHz)

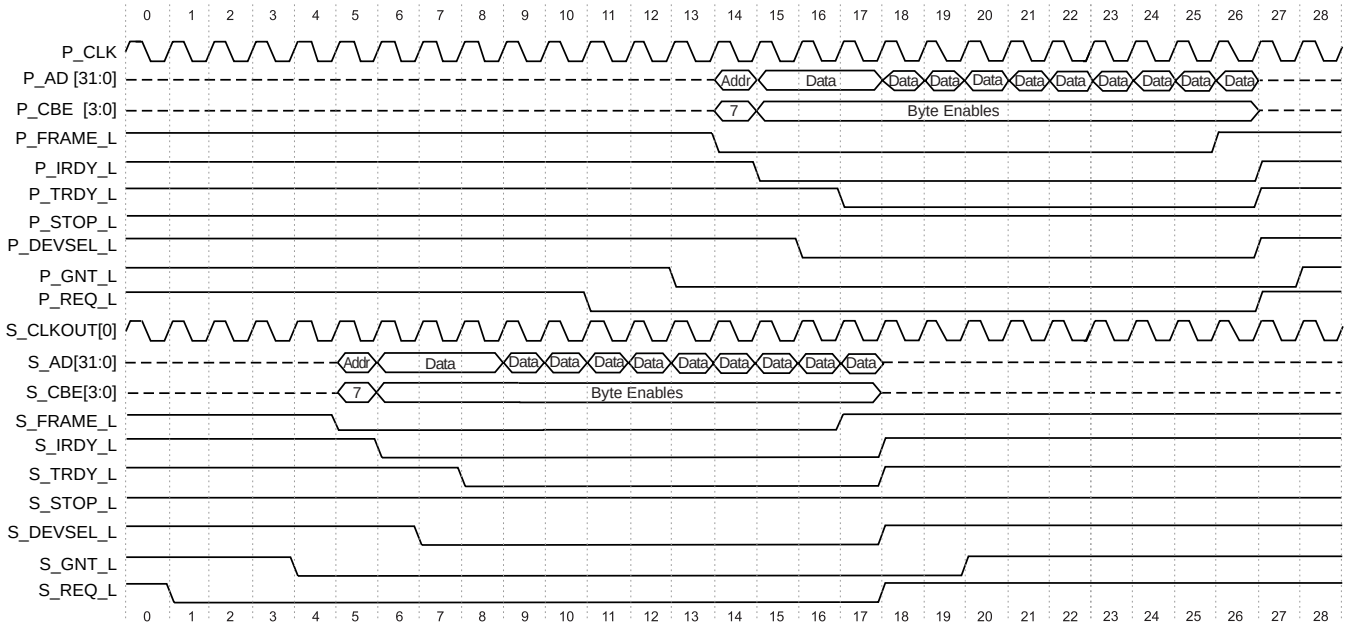


Figure 23. Downstream Delayed I/O Read Transaction (P --> S)

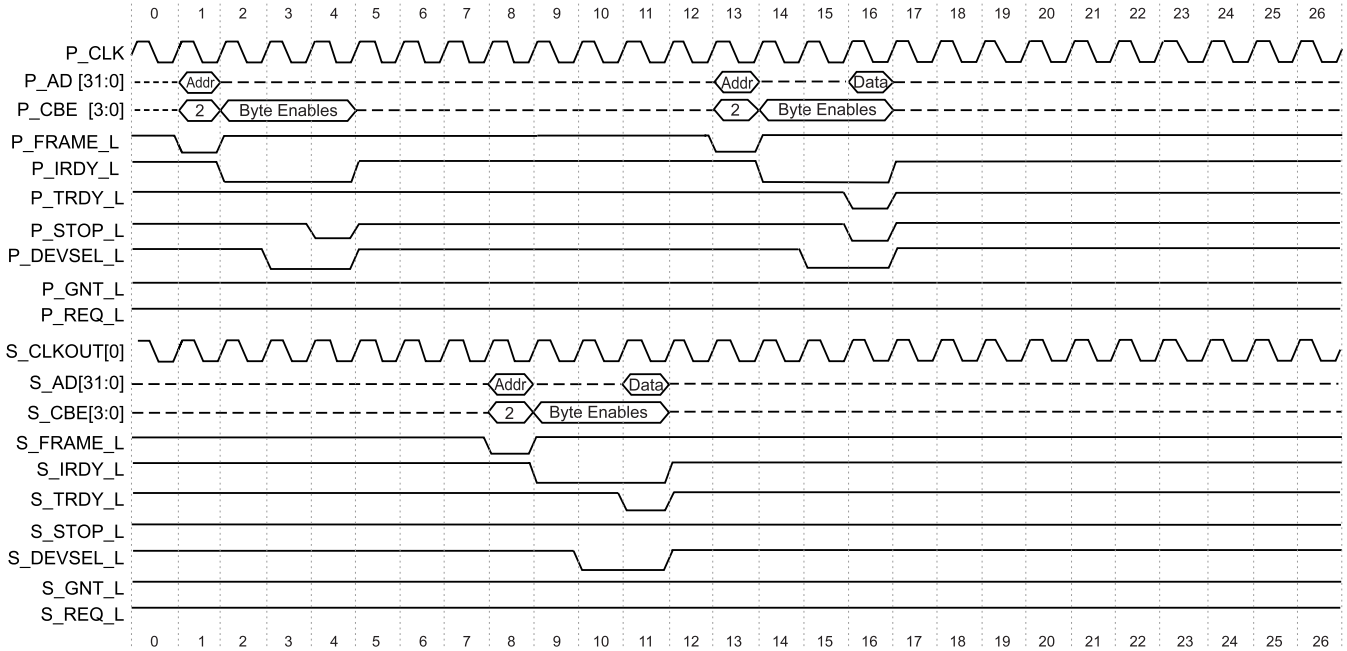


Figure 24. Downstream Delayed I/O Read Transaction (S2/33MHz-->S1/33MHz)

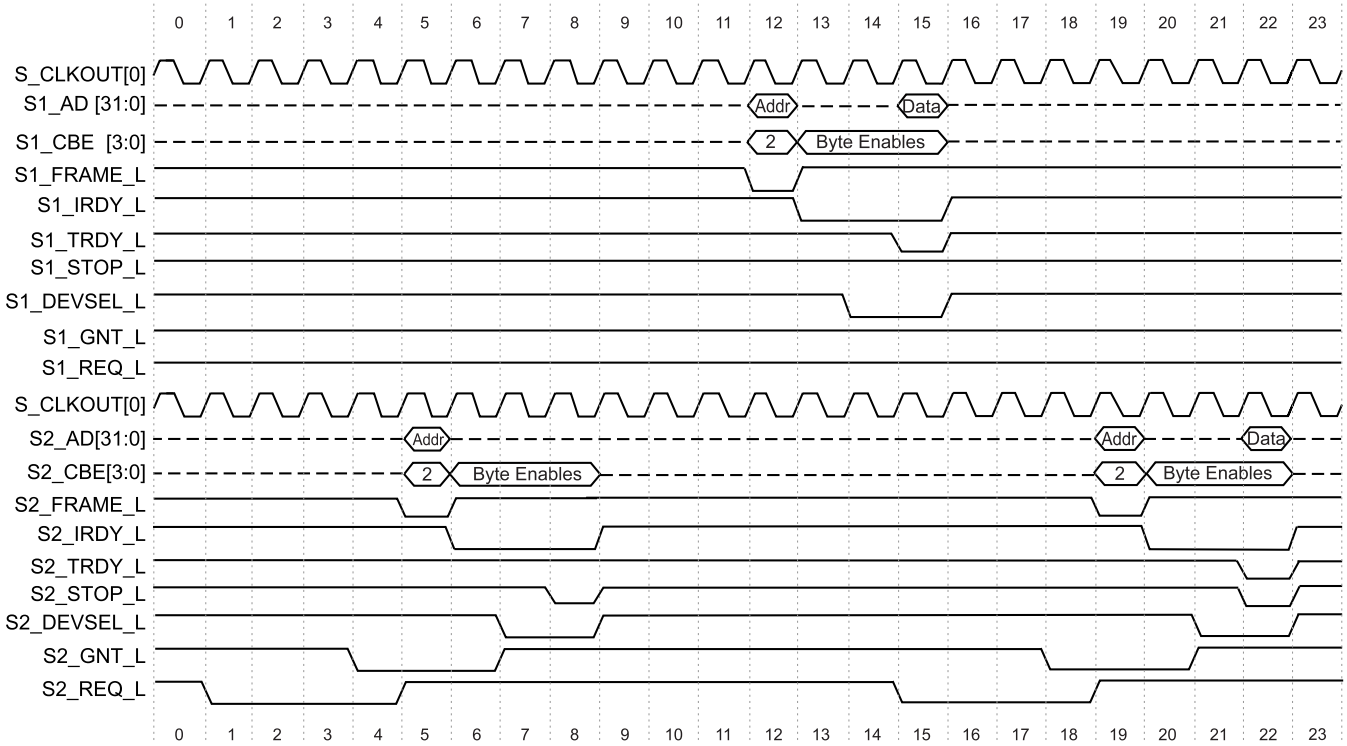


Figure 25. Downstream Delayed I/O Read Transaction (S1/33MHz-->S2/33MHz)

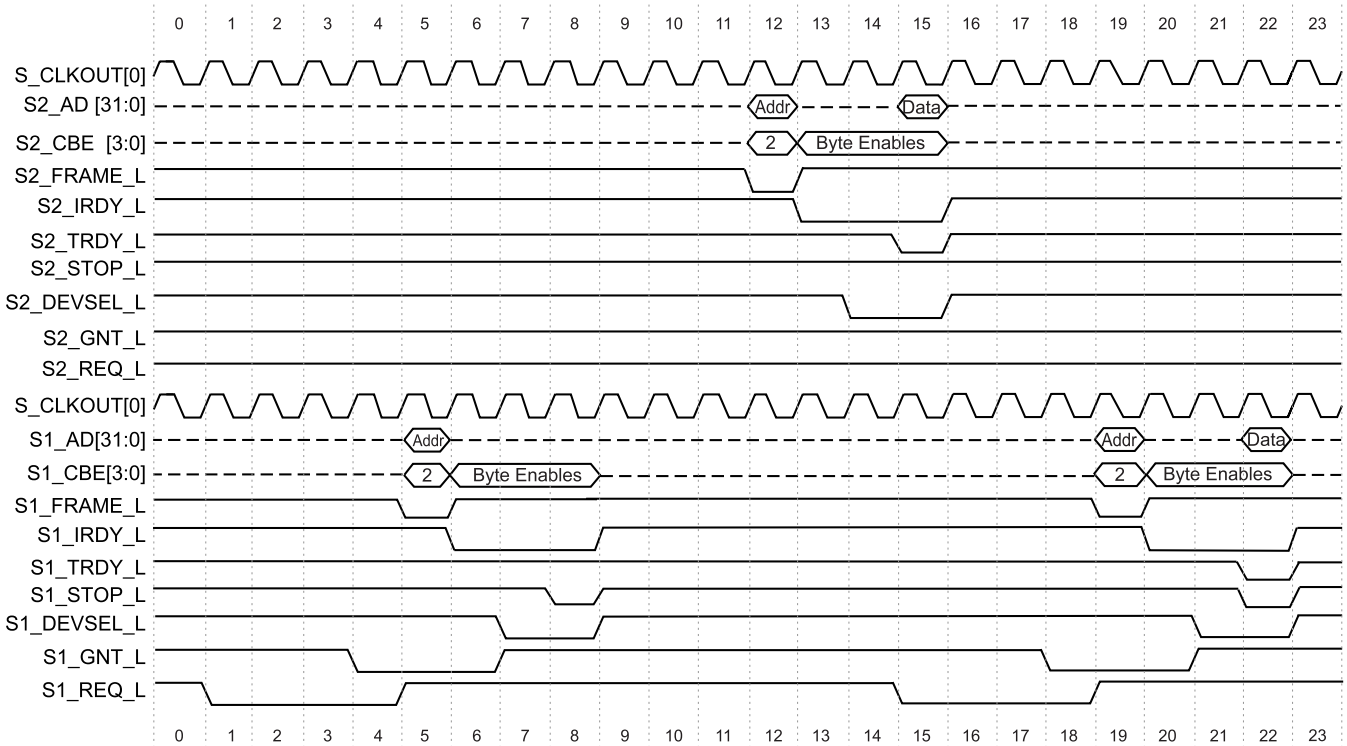


Figure 26. Upstream Delayed I/O Read Transaction (S/33MHz-->P/33MHz)

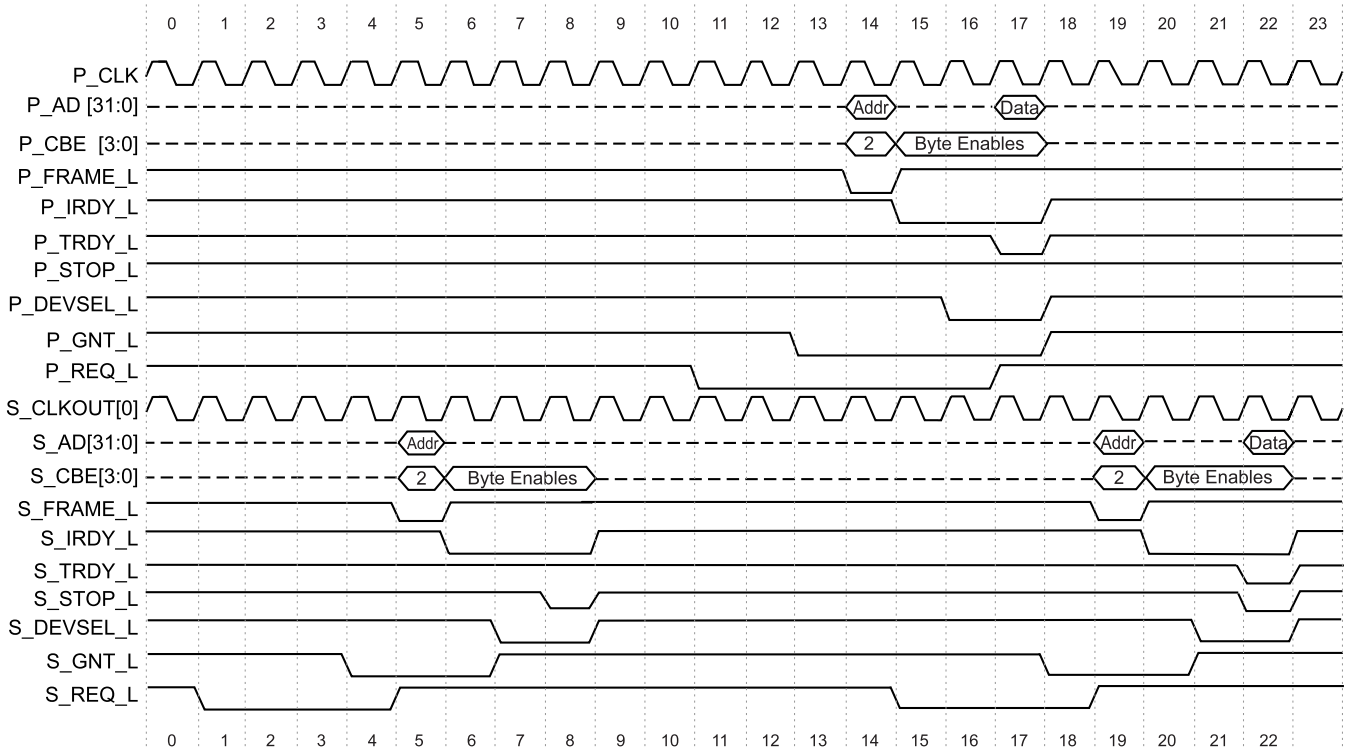


Figure 27. Downstream Delayed I/O Write Transaction (P --> S)

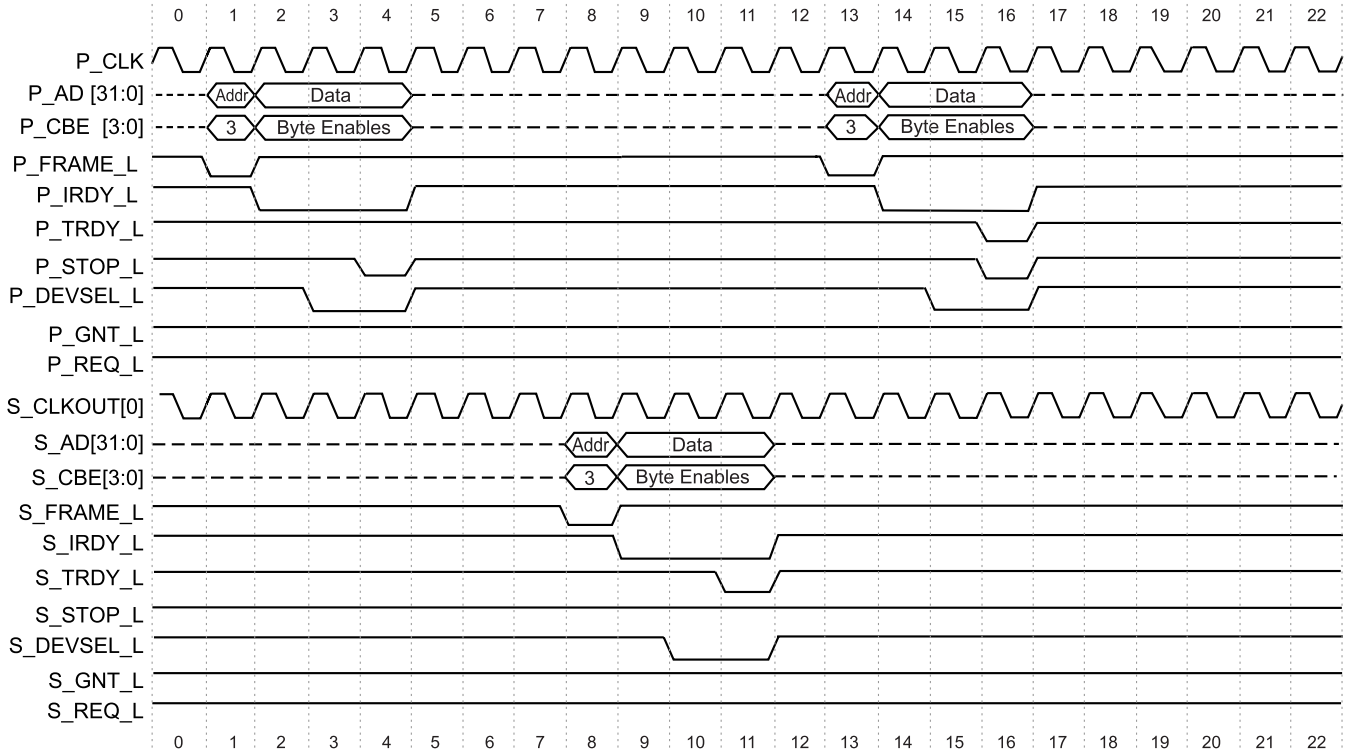


Figure 28. Downstream Delayed I/O Write Transaction (S2/33MHz-->S1/33MHz)

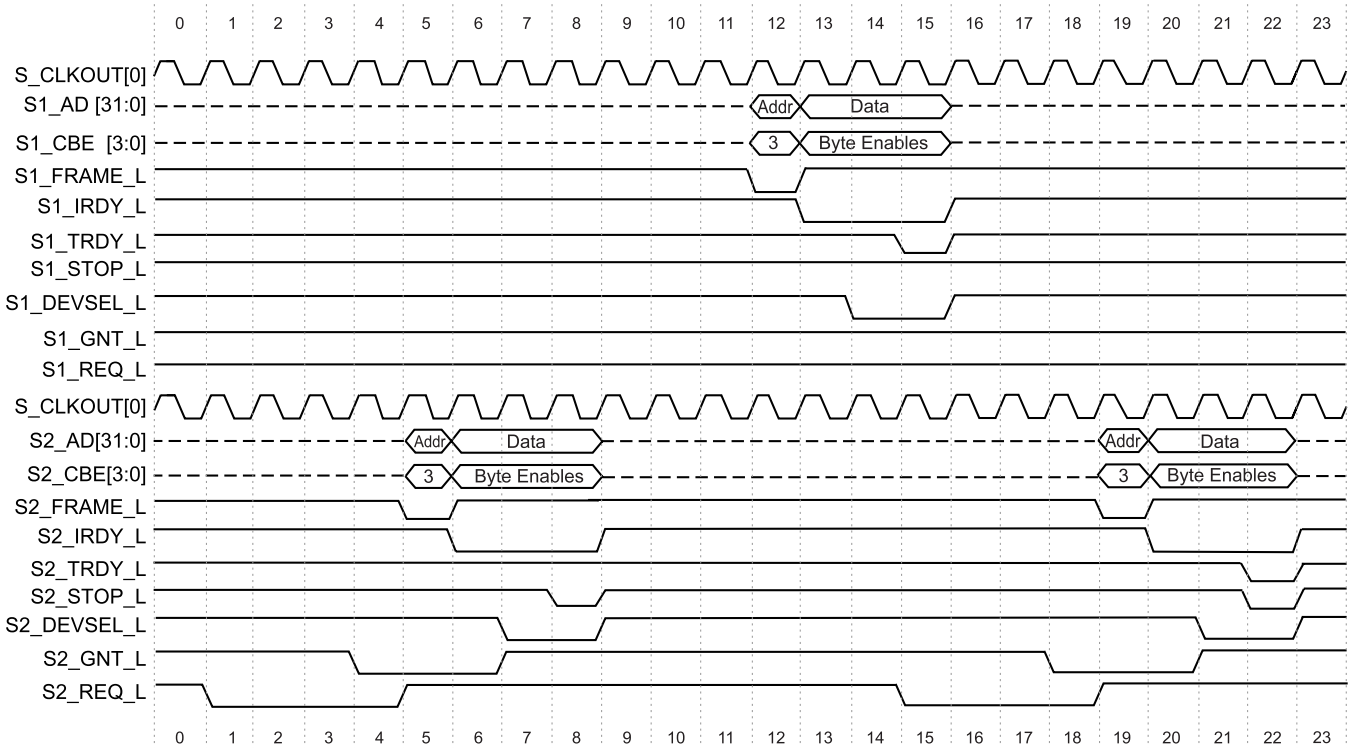


Figure 29. Downstream Delayed I/O Write Transaction (S1/33MHz-->S2/33MHz)

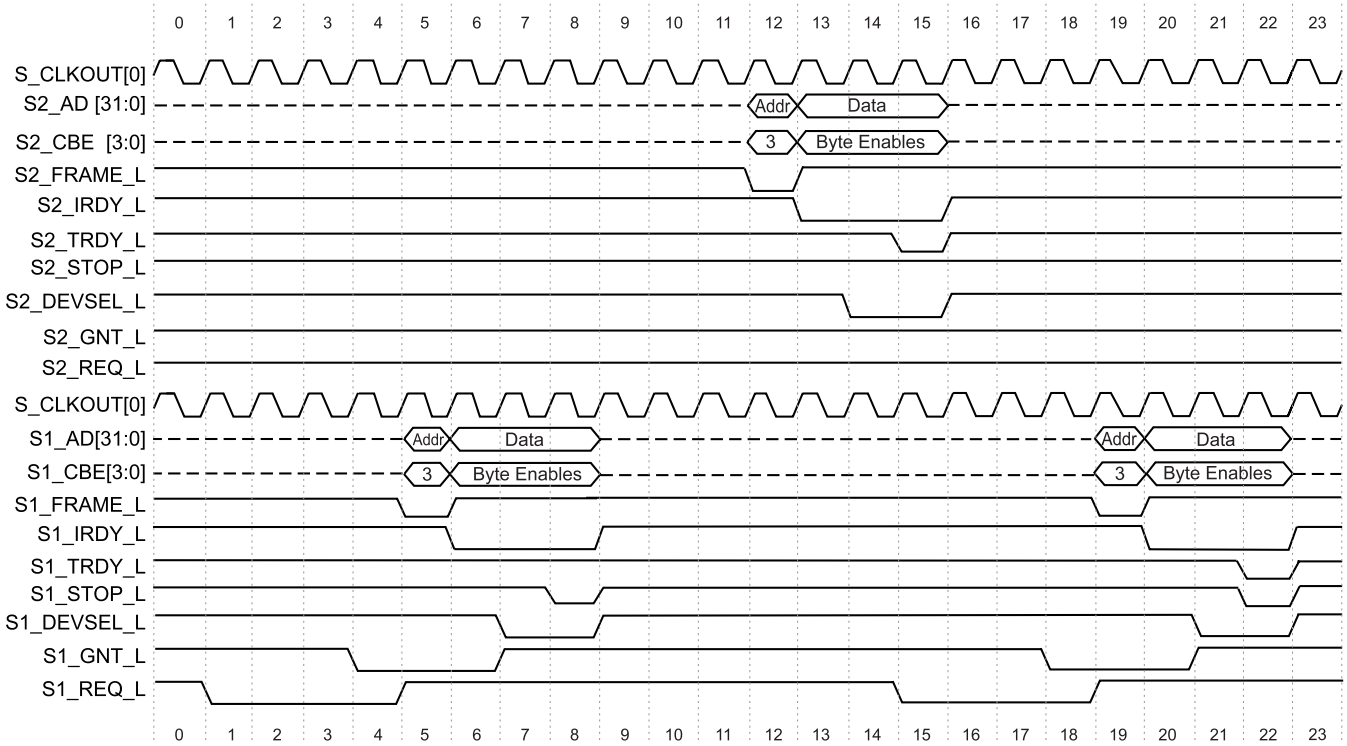
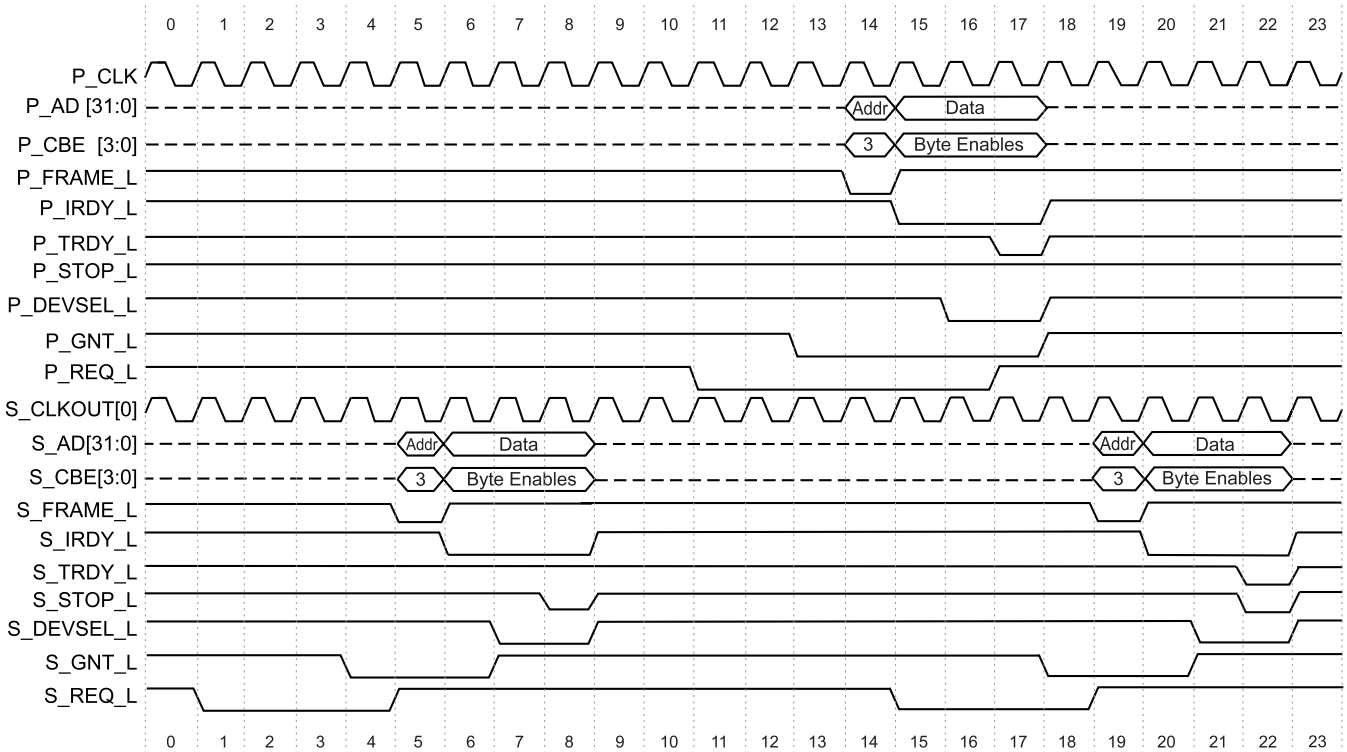


Figure 30. Upstream Delayed I/O Write Transaction (S --> P)



PI7C7100
3-Port
PCI Bridge

Appendix B

Evaluation
Board
User's Manual

PI7C7100 Evaluation Board User's Manual

General Information

1. Please make sure you have included with your PI7C7100 evaluation board, the five-page schematic and the preliminary specification for the PI7C7100.

2. Check all jumpers for proper settings:

Pin Name	Jumper	Function	Position
S_CFN#	JP4	Internal arbiter enable	1-2 (0)
S1_EN	JP5	S1 bus enable	2-3 (1)
S2_EN	JP6	S2 bus enable	2-3 (1)
SCAN_EN	JP7	SCAN control	1-2 (0)
SCAN_EN	JP7	SCLK_IN as clock input	2-3 (1)
PLL_TM	JP8	PLL test mode disable	1-2 (0)
BYPASS	JP9	PLL enable	1-2 (0)
P_FLUSH#	JP10	Primary FIFO flush disable	2-3 (1)

3. Check and make sure there are no shorts between power (3.3V, 5V, 12V, and –12V) and ground.
4. Plug evaluation board in any PCI slot on your system. Make sure your system is powered off before doing so.
5. Connect any PCI devices on the secondary slots of the evaluation board. Be careful that the orientation of the card is correct (see Diagram A).

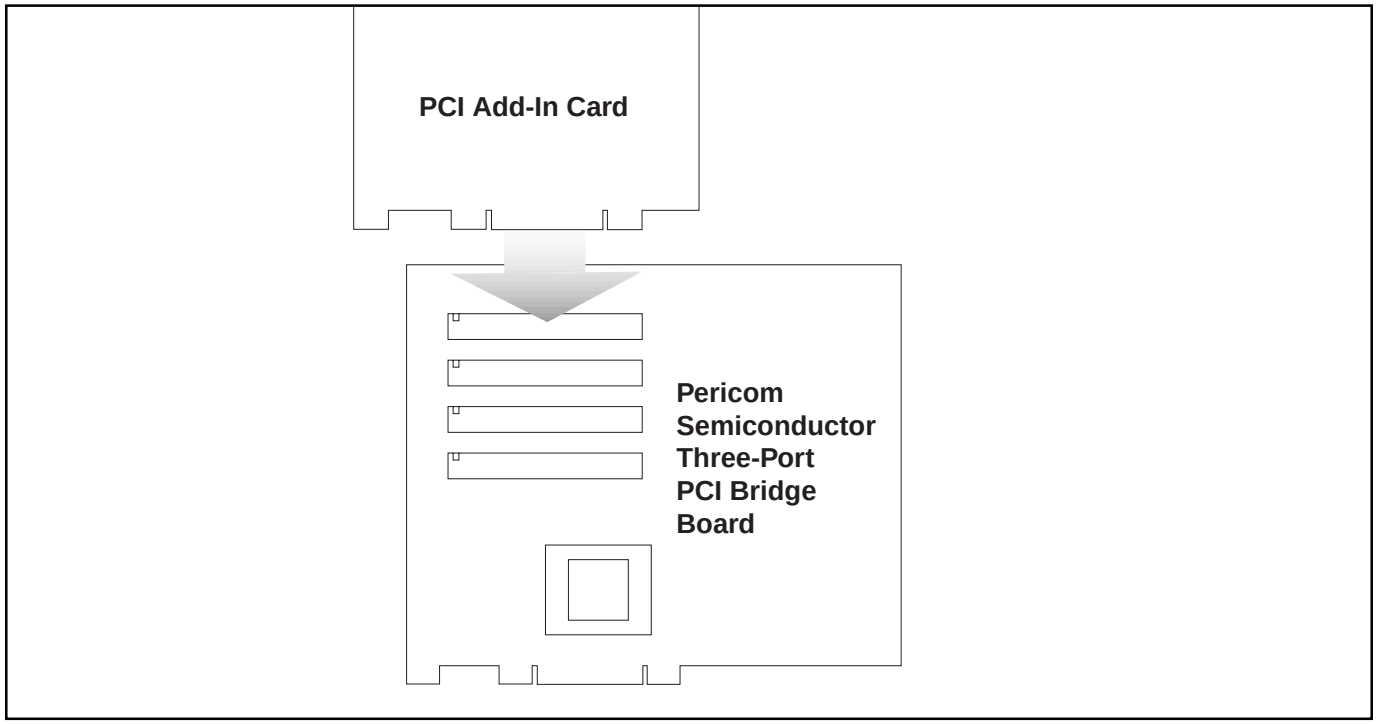


Diagram A

General Information (continued)

6. Turn on the power for the system. Your OS should already have drivers for the PI7C7100 evaluation board. In Win9X, Plug and Play should detect the device as a PCI-to-PCI bridge. The system may prompt you for the Win9X CD for the drivers. The OS will detect two PCI-to-PCI bridges as the PI7C7100 has two secondary PCI buses. In Win NT, you should not have to install drivers.
7. Install drivers for any PCI devices you have attached to the evaluation board.
8. If any of the steps are unclear or were unsuccessful, please contact your Pericom support person at 408-435-0800.
9. Thank you for evaluating Pericom Semiconductor Corporation's products.

Frequently Asked Questions

1. What is the function of SCAN_EN?

SCAN_EN is for a full scan test or S_CLKIN select. During SCAN mode, SCAN_EN will be driven to logic "0" or "logic "1" depending on functionality. During normal mode, if SCAN_EN is connected to logic "0" (JP7 in the 1-2 position), S_CLKIN will be used for PLL test only when PL_TM is active. If SCAN_EN is connected to logic "1" (JP7 in the 2-3 position), S_CLKIN will be the clock input for the secondary buses. All secondary clock outputs, S_CLKOUT [15:0], are still derived from P_CLK with 0-10ns delay. The S_CLKOUT [15:0] should be disabled by programming the bits [15:0] in both configuration registers 1 and 2 at offset 68h.

2. What is the function of SCAN_TM#?

SCAN_TM# is for full scan test and power on reset for the PLL. SCAN_TM# should be connected to logic "1" or to an RC path (R1 and C13) during normal operation.

3. How do you use the external arbiter?

a) Disable the on chip arbiter by connecting S_CFN to logic "1" (JP4 in the 2-3 position).
b) Use S1_REQ0# as GRANT and S1_GNT0# as REQUEST on the S1 bus.
c) Use S2_REQ0# as GRANT and S2_GNT0# as REQUEST on the S2 bus.

4. What is the purpose of having JP1, JP2, and JP3?

JP1, JP2, and JP3 are designed for easy access to the primary bus signals. You may connect any of these pins to an oscilloscope or a logic analyzer for observation. No connection is required for normal operation. The following table indicates which bus signals correspond to which pins.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
JP2	REQ	AD29	AD26	CBE3	AD21	AD18	CBE2	IRDY	LOCK	PAR	AD14	AD11	CBE0	AD6	AD5	AD0
JP3	AD31	AD28	AD25	AD23	AD20	AD17	FRAME	DVSEL	PERR	CBE1	AD13	AD10	AD8	AD4	AD2	GND
JP1	GNT	AD30	AD27	AD24	AD22	AD19	AD16	IRDY	STOP	SERR	AD15	AD12	AD9	AD7	AD3	AD1

5. What is the purpose for having U17, U19, and U20?

U17, U19, and U20 are designed for easy access to the digital ground planes for observation.

6. How is the evaluation board constructed?

The evaluation board is a six-layer PCB. The top and bottom layers (1 and 6) are for signals, power, and ground routing. Layer 2 and layer 5 are ground planes. Layer 3 is a digital 3.3V power plane. Layer 4 is a digital 5V power plane with an island of analog 3.3V power.

7. What is the function of S_CLKIN?

The S_CLKIN pin is a test pin for the on chip PLL when PLL_TM is set to logic "1". During normal operation, if PLL_TM is set to logic "0", SCAN_TM# is set to logic "1", and SCAN_EN is set to logic "1", then S_CLKIN will be the clock input for both the secondary buses. However, the S_CLKOUT [15:0] are still derived by programming bits [15:0] in both configuration registers 1 and 2 at offset 68h.

8. What clock frequency combinations does the PI7C7100 support?

Primary Bus	Secondary (1 and 2) Buses
33MHz	33MHz

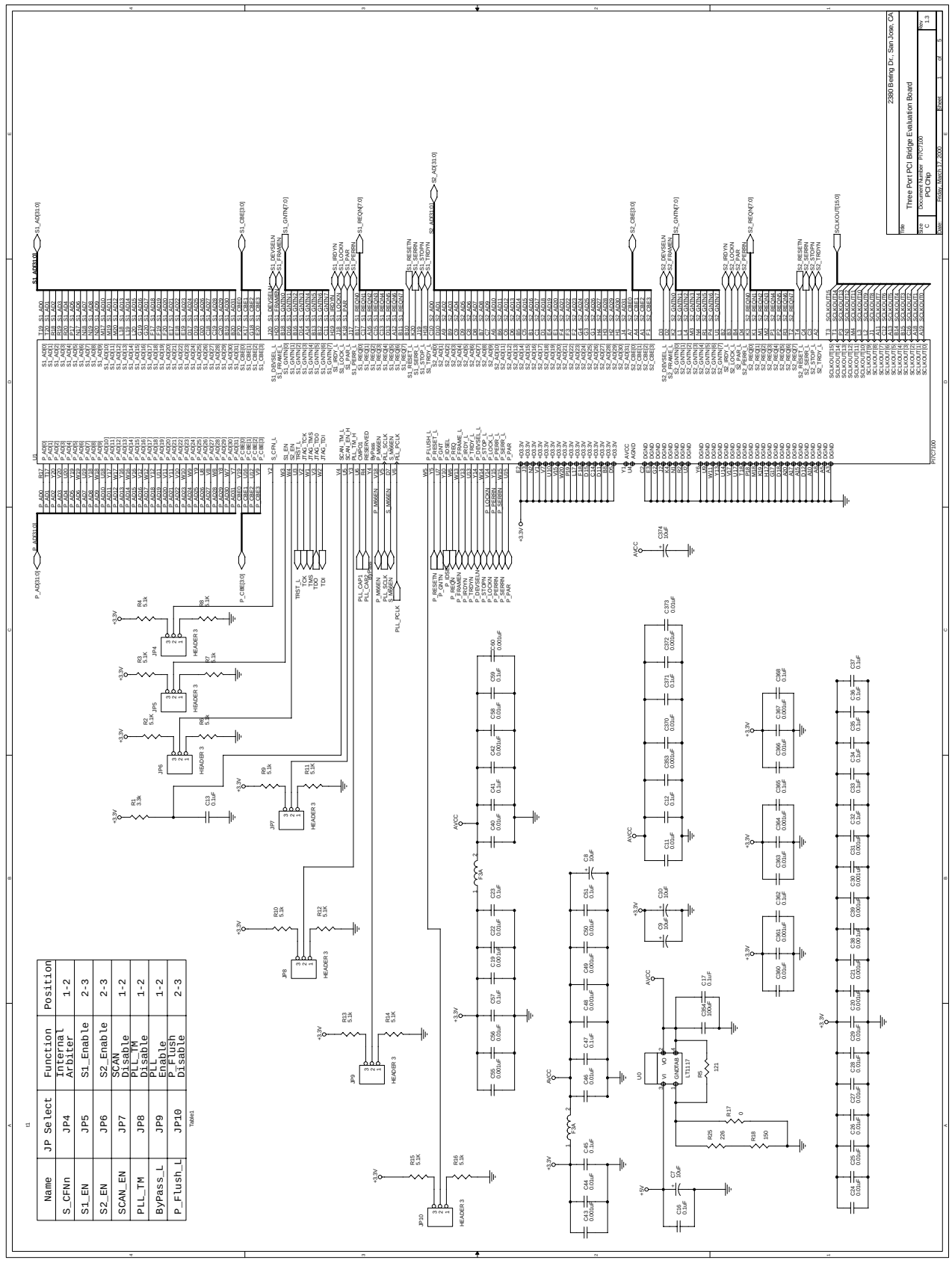
9. How are the JTAG signals being connected?

The JTAG signals consist of TRST#, TCK, TMS, TDI, and TDO. All the mentioned signals have weak internal pull-up connections. Therefore, no connection is needed if you want the JTAG circuit to be disabled. If you want to activate the JTAG circuit, you need to connect all five signals according to the JTAG specification (IEEE 1149).

PI7C7100
3-Port
PCI Bridge

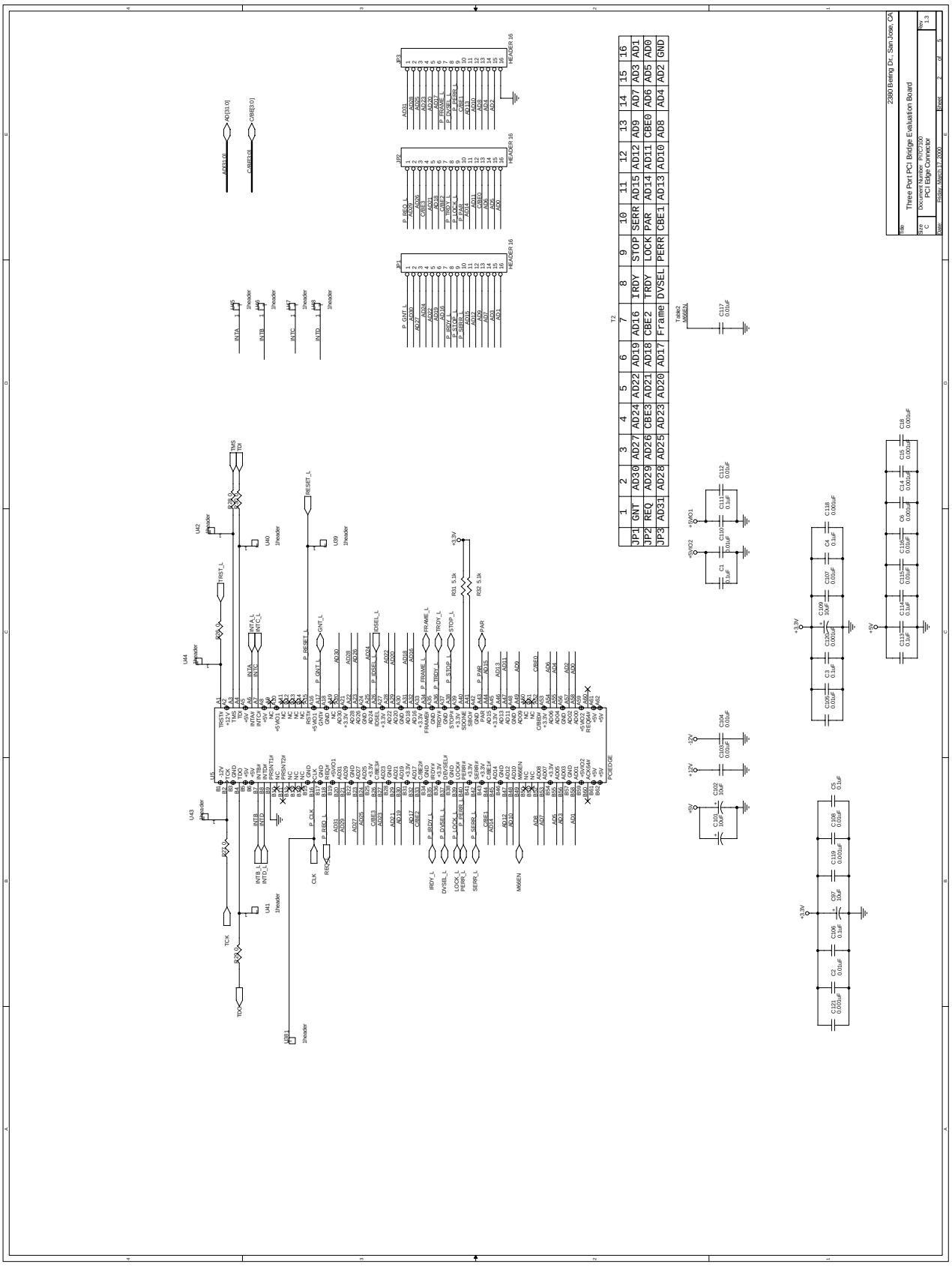
Appendix C

Schematics

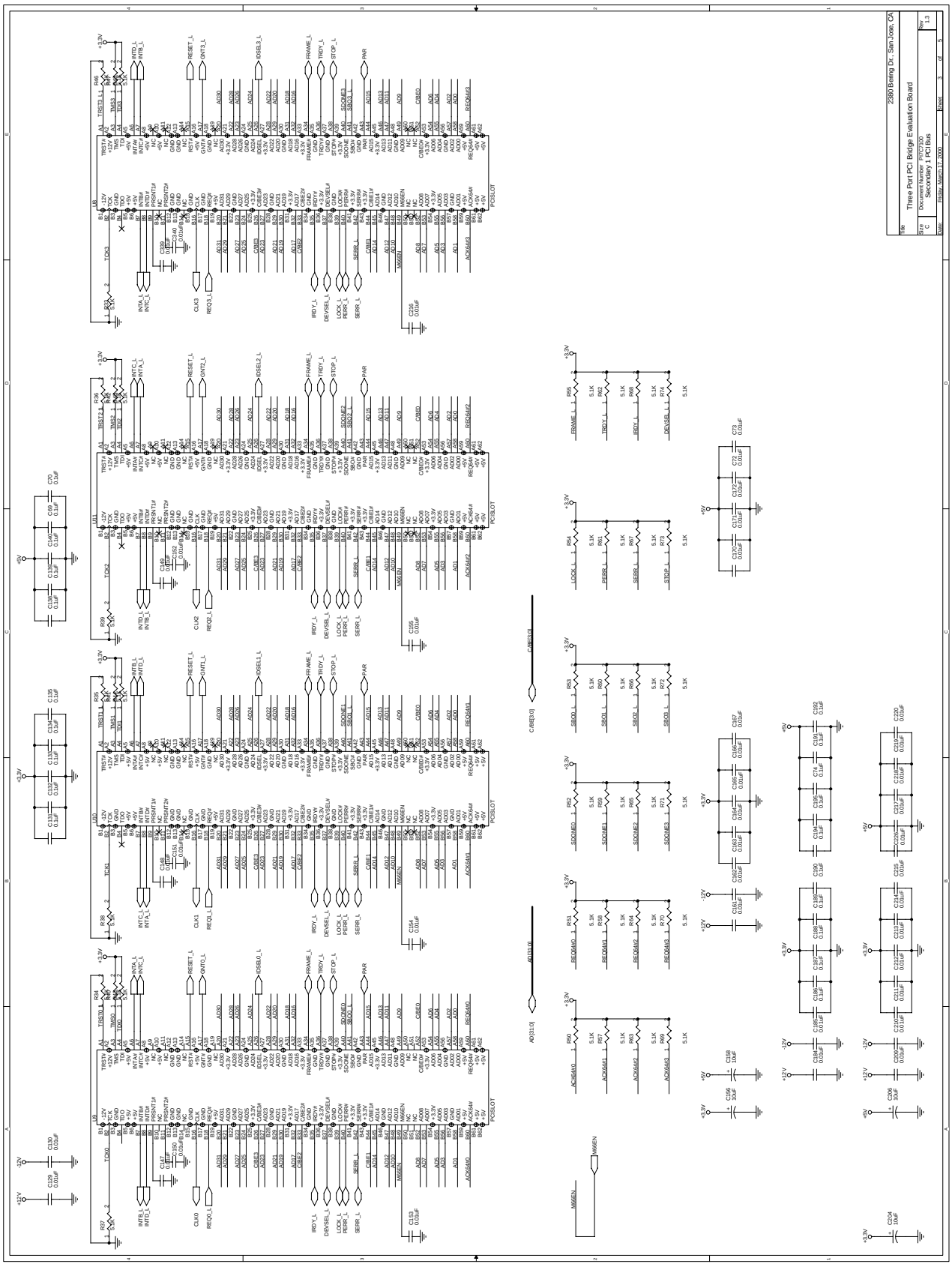


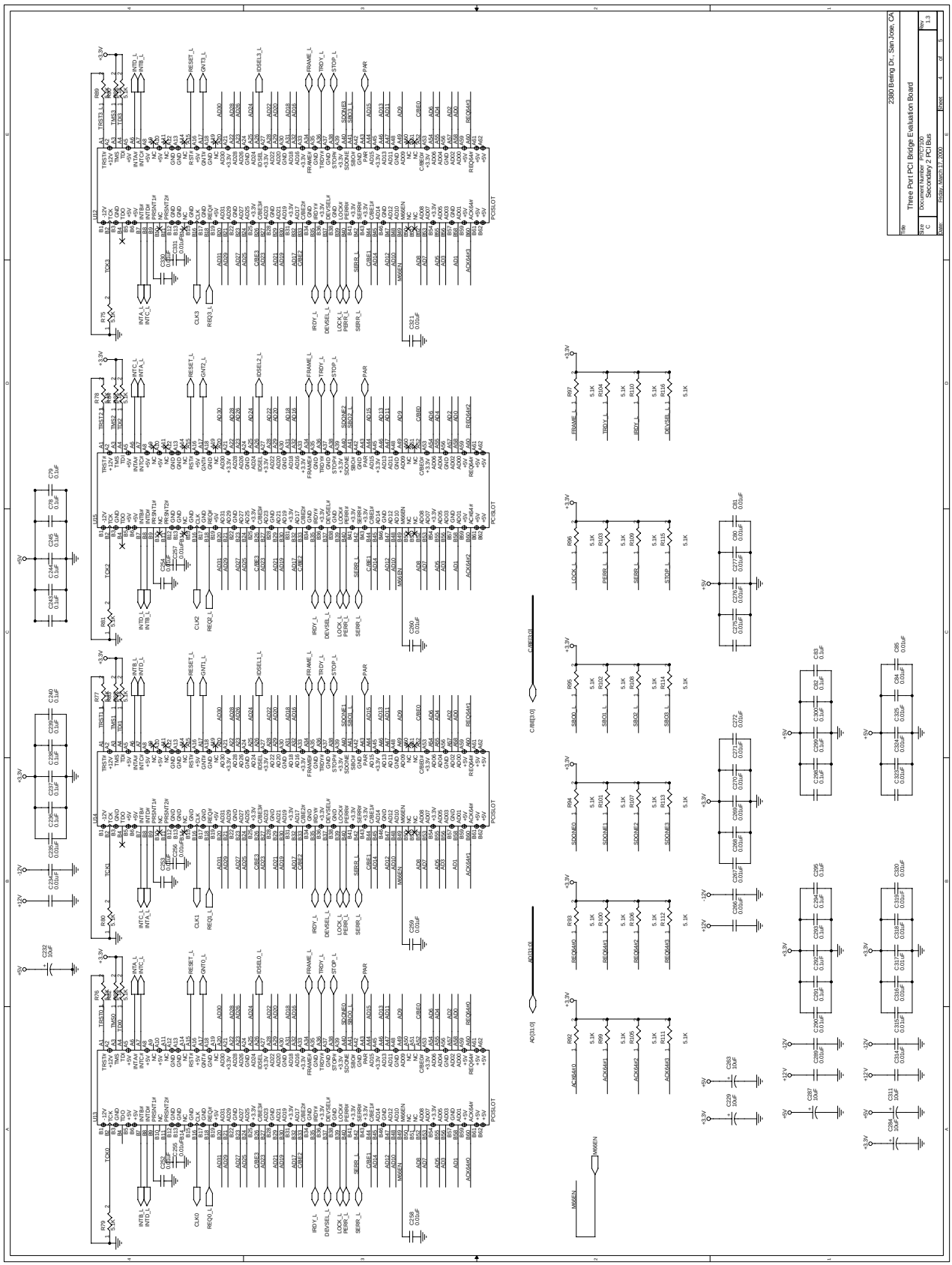
Name	JP Select	Function	Position
S_CFNn	JP4	Internal Arbiter	1-2
S1_EN	JP5	S1_Enable	2-3
S2_EN	JP6	S2_Enable	2-3
SCAN_EN	JP7	SCAN Disable	1-2
PLL_TM	JP8	PLL_TM Disable	1-2
ByPass_L	JP9	PLL Enable	1-2
P_Flush_L	JP10	P_Flush Disable	2-3

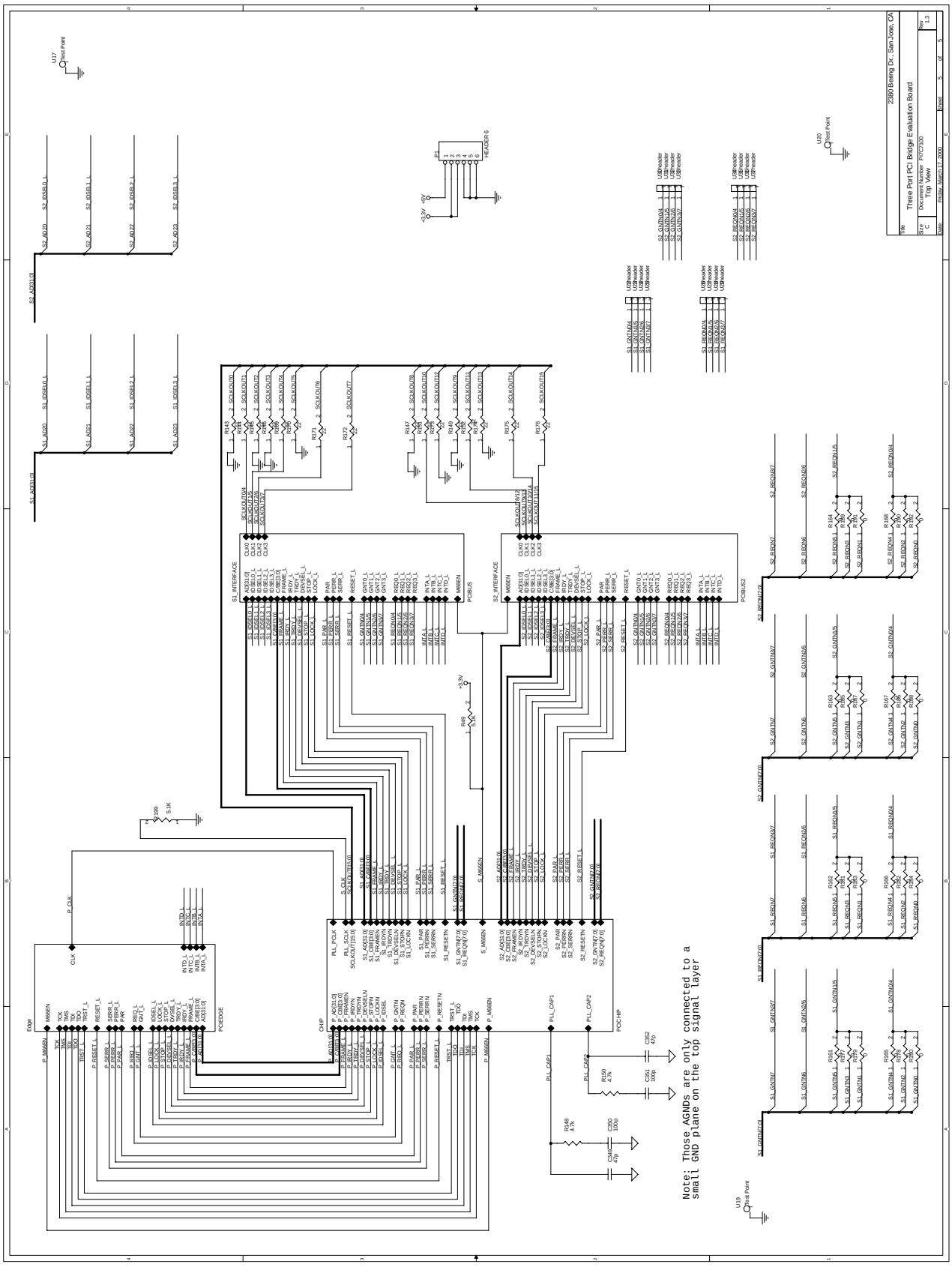
Table1



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
JP1	GNT	AD30	AD27	AD24	AD22	AD19	AD16	TRDY	STOP	SERR	AD15	AD12	AD9	AD7	AD3	AD1
JP2	REQ	AD29	AD26	CBE3	AD21	AD18	CBE2	TRDY	LOCK	PAR	AD14	AD11	CBE0	AD6	AD5	AD0
JP3	AD31	AD28	AD25	AD23	AD20	AD17	Frame	DVSEL	PERR	CBE1	AD13	AD10	AD8	AD4	AD2	GND


2380 Bering Dr., San Jose, CA
Three Port PCI Bridge Evaluation Board
PI7C7100
Secondary 1 PCI Bus
Friday, March 17, 2000




Note: Those AGNDs are only connected to a small GND plane on the top signal layer
2380 Bering Dr., San Jose, CA
Three Port PCI Bridge Evaluation Board
Document Number PI7C7100
Top View
Rev 1.3
Friday, March 17, 2000
Sheet 5 of 5