VME64 to PCI Bridge System-on-Chip (SoC)
Technical Reference Manual

Silicore Corporation

•

Silicore Corporation
6310 Butterworth Lane; Corcoran, MN (USA) 55340
TEL: (763) 478-3567 FAX: (763) 478-3568
URL: www.silicore.net

Electronic Design
Sensors • IP Cores

VME64 to PCI Bridge System-on-Chip (SoC)

Table of Contents

Manual Revision Level

Manual Revisions		
Manual Revision Level	**Date**	**Description of Changes**
0	02 JAN 2002	Preliminary release for comment. PCI Rev ID Code: 0x0000
1	30 SEP 2002	Project complete. PCI Rev ID Code: 0x0001
2	09 OCT 2002	Incorporate the following changes: - Fix typographical errors - Add SPECIALREG register - Add three-state clock enable to EEPROM clock (CEEPROM entity)
3	7 DEC 2002	Incorporate the following changes: - Reset all sections (except PCI core) after VMEbus [SYSRESET*]. - Add 'PCIResetMonitor' bit (D04) to the DMC_HW_CONTROL register. - PCI Rev ID Code: 0x0002
4	17 JAN 2004	Release under the GNU Free Documentation License

1.0 Introduction

The VME64 to PCI Bridge is a System-on-Chip that allows data transfer between a VMEbus slave interface and a PCI target interface. It is delivered as a VHDL soft core module that is intended for use on a Xilinx Spartan 2 FPGA. However, with minor modifications this soft system can be implemented on other types or brands of FPGA and ASIC devices. Figure 1-1 shows a functional diagram of the bridge.

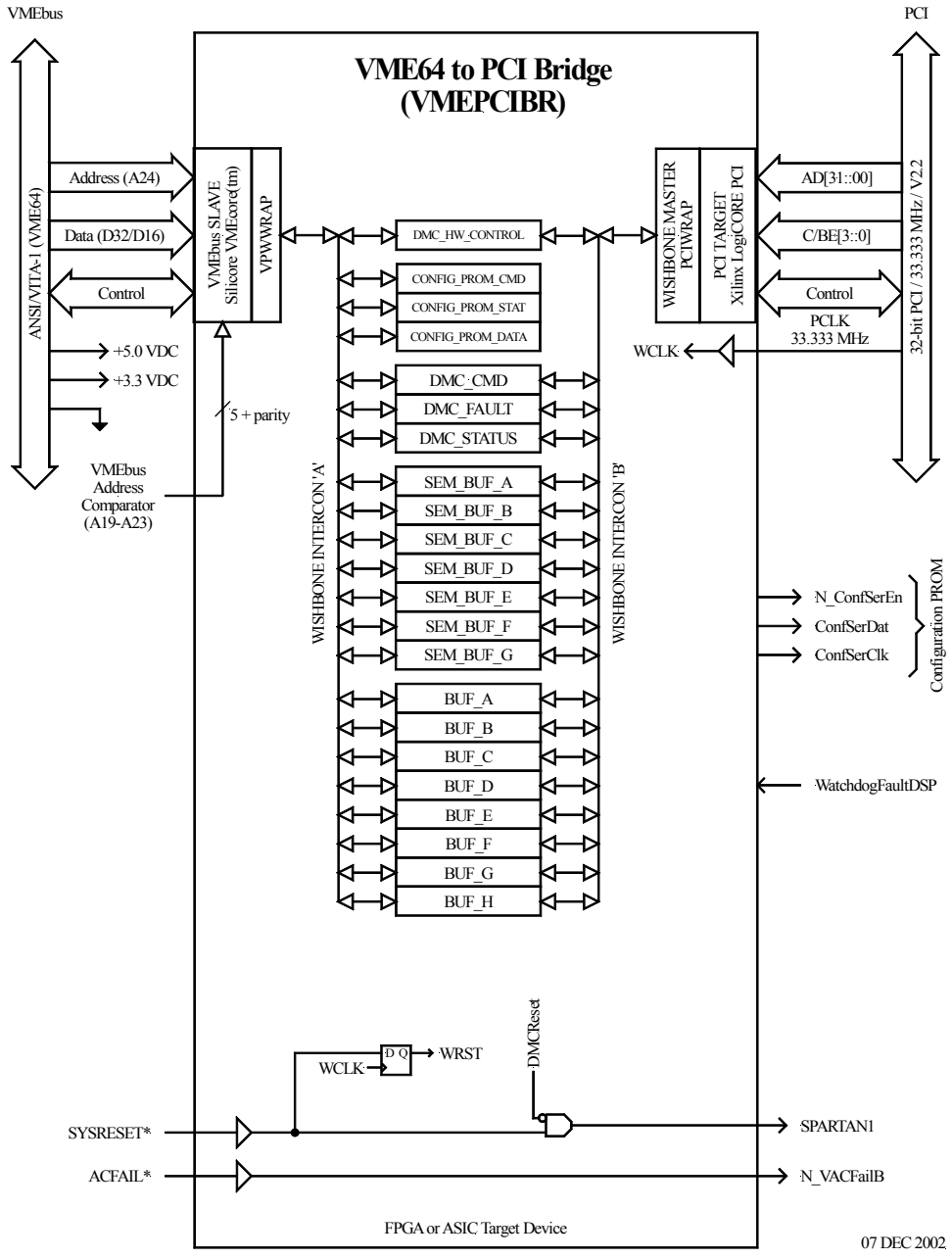


Figure 1-1. Functional diagram of the VMEbus to PCI bridge.

1.1 Features Of The Bridge

- VMEbus interface features:
  - Silicore VMEcore™ IP Core technology[1]
  - A24:D32:D16 slave interface
  - Posted read and write capabilities
  - Synchronous interface design
  - Conforms to ANSI/VITA 1 – 1994

- PCI interface features:
  - Xilinx LogiCORE™ PCI IP Core technology[2]
  - D32 target interface
  - 0 - 33.333 MHz operation
  - Conforms to PCI Revison 2.2

- Shared memory features:
  - Nine, 256 x 32-bit shared memory buffers
  - Semaphore control register for each buffer (except BUF H)
  - Buffers use independent memory arbitration

- Serial EEPROM interface for Atmel AT17 Series of FPGA Configuration ROM.

- Internal WISHBONE bus features[3]:
  - Conforms to WISHBONE Revision B.2
  - Simple, compact, logical hardware interfaces
  - Portable across FPGA and ASIC target devices
  - Third party support (including open source) is available at www.opencores.org

- Straightforward and highly reliable synchronous design.

- Written in flexible and portable VHDL hardware description language. All components (with the exception of the Xilinx LOGIcore PCI interface) are delivered as soft cores, meaning that all VHDL source code and test benches are supplied. This allows the end user to maintain and modify the design. Complete documentation is also provided.

---

[1] VMEcore™ is a VMEbus interface that is generated by the Silicore Bus Interface Writer™. The Bus Interface Writer is a parametric core generator that generates VHDL source code files.

[2] The VME64 to PCI Bridge SoC described in this manual interfaces to the back end of the Xilinx LOGIcore PCI, and is purchased separately from Xilinx. For more information, please refer to the Xilinx, Inc. web site at www.xilinx.com and the Xilinx LOGIcore PCI Design Guide.

[3] The internal WISHBONE SoC components were modified from source code in the public domain. Open source WISHBONE SoC components are available on the web from the WISHBONE Service Center at www.silicore.net/wishbone.htm or from www.opencores.org.

1.2 Glossary Of Terms

0x (numerical prefix)
The '0x' prefix indicates a hexadecimal number. It is the same nomenclature as commonly used in the 'C' programming language.

Active High Logic State
A logic state that is 'true' when the logic level is a binary '1' (high state). The high state is at a higher voltage than the low state.

Active Low Logic State
A logic state that is 'true' when the logic level is a binary '0' (low state). The low state is at a lower voltage than the high state.

ASIC
Acronym for: Application Specific Integrated Circuit. A general term which describes a generic array of logic gates or analog building blocks which are programmed by a metallization layer at a silicon foundry. High level circuit descriptions are impressed upon the logic gates or analog building blocks in the form of metal interconnects.

Asserted
(1) A verb indicating that a logic state has switched from the inactive to the active state. When active high logic is used it means that a signal has switched from a logic low level to a logic high level. (2) *Assert:* to cause a signal line to make a transition from its logically false (inactive) state to its logically true (active) state. Opposite of *negated*.

Bit
A single binary (base 2) digit.

Bridge
An interconnection system that allows data exchange between two or more buses. The buses may have similar or different electrical, mechanical and logical structures.

Bus Interface
An electronic circuit that drives or receives data or power from a bus.

Bus Cycle
The process whereby digital signals effect the transfer of data across a bus by means of an inter-locked sequence of control signals.

BYTE
A unit of data that is 8-bits wide. Also see: *WORD*, *DWORD* and *QWORD*.

Data Organization
The ordering of data during a transfer. Generally, 8-bit (byte) data can be stored with the most significant byte of a multi byte transfer at the higher or the lower address. These two methods

are generally called BIG ENDIAN and LITTLE ENDIAN, respectively. In general, BIG ENDIAN refers to byte lane ordering where the most significant byte is stored at the lower address. LITTLE ENDIAN refers to byte lane ordering where the most significant byte is stored at the higher address. The terms BIG ENDIAN and LITTLE ENDIAN for data organization was coined by Danny Cohen of the Information Sciences Institute, and was derived from the book Gulliver's Travels by Jonathan Swift.

DWORD
A unit of data that is 32-bits wide. Also see: *BYTE*, *WORD* and *QWORD*.

ENDIAN
See the definition under 'Data Organization'.

Firm Core
An IP Core that is delivered in a way that allows conversion into an integrated circuit design, but does not allow the design to be easily reverse engineered. It is analogous to a binary or object file in the field of computer software design.

FPGA
Acronym for: Field Programmable Gate Array. Describes a generic array of logical gates and interconnect paths which are programmed by the end user. High level logic descriptions are impressed upon the gates and interconnect paths, often in the form of IP Cores.

Granularity
The smallest unit of data transfer that a port is capable of transferring. For example, a 32-bit port can be broken up into four 8-bit BYTE segments. In this case, the granularity of the interface is 8-bits. Also see: *port size* and *operand size*.

Hard Core
An IP Core that is delivered in the form of a mask set (i.e. a graphical description of the features and connections in an integrated circuit).

Hardware Description Language (HDL)
(1) Acronym for: Hardware Description Language. Examples include VHDL and Verilog®. (2) A general-purpose language used for the design of digital electronic systems.

IP Core
Acronym for: Intellectual Property Core. Also see: *soft core*, *firm core* and *hard core*.

Negated
A verb indicating that a logic state has switched from the active to the inactive state. When active high logic is used it means that a signal has switched from a logic high level to a logic low level. Also see: *asserted*.

Operand Size
The operand size is the largest single unit of data that is moved through an interface. For example, a 32-bit DWORD operand can be moved through an 16-bit port with two data transfers. Also see: *granularity* and *port size*.

PCI
Acronym for: Peripheral Component Interconnect. Generally used as an interconnection scheme (bus) between integrated circuits. It also exists as a board level interconnection known as Compact PCI (or cPCI).

Port Size
The width of a data port in bits. Also see: *granularity* and *operand size*.

Posted Read and Write Cycles
A method for minimizing data transfer latency between a data source and its destination. During a posted read or write cycle a bus interface, lying between a data source and its destination, captures the data and completes any handshaking protocols with the data source. At the same time, the bus interface initiates handshaking with the data destination. This alleviates the need for the data source to wait until the destination is ready to accept the data.

QWORD
A unit of data that is 64-bits wide. Also see: *BYTE*, *WORD* and *DWORD*.

Register (REG)
A device capable of retaining information for control purposes that is contained in a single BYTE, WORD or DWORD of storage area. Said storage area is not general purpose memory. Also see: *shared register (SREG).*

Shared Memory (SMEM)
(1) The address space in a system which is accessible to all modules. (2) A type of memory that is shared between two or more ports. Shared memory uses a hardware arbitration scheme that allows simultaneous accesses from two or more processors. However, during simultaneous accesses one processor may be required to wait until the another has completed its accesses into the shared memory area. Also see: *shared register (SREG)*.

Shared Register (SREG)
(1) A register space in a system which is accessible to all modules. (2) A type of register that is shared between two or more ports. Shared registers uses a hardware arbitration scheme that allows simultaneous accesses from two or more processors. However, during simultaneous accesses one processor may be required to wait until the another has completed its accesses into the shared memory area. Also see: *register, shared memory (SMEM)*.

SoC
Acronym for System-on-Chip. Also see: *System-on-Chip*.

Soft Core
An IP Core that is delivered in the form of a hardware description language or schematic diagram.

System-on-Chip (SoC)
A method by which whole systems are created on a single integrated circuit chip. In many cases, this requires the use of IP cores which have been designed by multiple IP core providers. System-on-Chip is similar to traditional microcomputer bus systems whereby the individual components are designed, tested and built separately. The components are then integrated to form a finished system.

Target Device
The semiconductor type (or technology) onto which the IP core design is impressed. Typical examples include FPGA and ASIC target devices.

VHDL
Acronym for: VHSIC Hardware Description Language. [VHSIC: Very High Speed Integrated Circuit]. A textual based computer language intended for use in circuit design. The VHDL language is both a synthesis and a simulation tool. Early forms of the language emerged from US Dept. of Defense ARPA projects in the 1960's, and have since been greatly expanded and refined. Complete descriptions of the language can be found in the IEEE 1076, IEEE 1073.3, and IEEE 1164 specifications.

VMEbus
Acronym for: Versa Module Eurocard bus. A popular microcomputer (board) bus. Standardized under IEC 821, IEEE 1014 and ANSI/VITA 1-1994.

WISHBONE
A flexible System-on-Chip (SoC) design methodology. WISHBONE establishes common interface standards for data exchange between modules within an integrated circuit chip. Its purpose is to foster design reuse, portability and reliability of SoC designs. WISHBONE is a public domain standard.

Wrapper
A circuit element that converts a non-WISHBONE IP Core into a WISHBONE compatible IP Core. For example, consider a 16-byte synchronous memory primitive that is provided by an IC vendor. The memory primitive can be made into a WISHBONE compatible SLAVE by layering a circuit over the memory primitive, thereby creating a WISHBONE compatible SLAVE. A wrapper is analogous to a technique used to convert software written in 'C' to that written in 'C++'.

WORD
A unit of data that is 16-bits wide. Also see: *BYTE*, *DWORD* and *QWORD*.

1.3 Recommended Skill Level

It is recommended that the user have some experience with VHDL syntax and synthesis before attempting to integrate this core (or almost any other HDL core for that matter) into an FPGA or ASIC device. Most VHDL users report a fairly stiff learning curve on their first project, so it's better to have that experience before attempting to integrate the core. Prior experience with one or two medium size VHDL projects should be sufficient. On the other hand, some users may find the integration of the core a good way to learn many of the concepts in the VHDL language. Those users should find the integration experience rewarding.

1.4 References

- Ashenden, Peter J. The Designer's Guide to VHDL. Morgan Kaufmann Publishers, Inc. 1996. ISBN 1-55860-270-4. Excellent general purpose reference guide to VHDL. Weak on synthesis, stronger on test benches. Good general purpose guide, very complete.

- IEEE Standard VHDL Language Reference Manual. IEEE Std 1076-1993. IEEE, New York NY USA 1993. This is a standard, and not a tutorial by any means. Useful for defining portable VHDL code.

- IEEE Standard VHDL Synthesis Packages. IEEE Std 1076.3-1997. IEEE, New York NY USA 1997. This is a standard, and not a tutorial by any means. Useful for defining portable VHDL code.

- IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164). IEEE Std 1164-1993. IEEE, New York NY USA 1993. This is a standard, and not a tutorial by any means. Useful for defining portable VHDL code.

- Pellerin, David and Douglas Taylor. VHDL Made Easy. Prentice Hall PTR 1997. ISBN 0-13-650763-8. Good introduction to VHDL synthesis and test benches, and closely follows the IEEE standards.

- PCI Special Interest Group. PCI Local Bus Specification Revision 2.0.

- Peterson, Wade D. The VMEbus Handbook, 4th Ed. VITA 1997 ISBN 1-885731-08-6

- Peterson, Wade D. WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores. Revision B.2.

- Schmitz, Manfred. "PCI-to-VME Bridging: Keeping the Best of Both Worlds". RTC Magazine, Dec 2001. pp 59-64.

- Shanley, Tom and Don Anderson. PCI System Architecture 4th Ed. Addison-Wesley 2001. ISBN 0-201-30974-2.

- Skahill, Kevin. VHDL For Programmable Logic. Addison-Wesley 1996. ISBN 0-201-89573-0. Excellent reference for VHDL synthesis. Very good treatment of practical VHDL code for the synthesis of logic elements. Weak on test benches and execution of the IEEE standards.

- VITA. American National Standard for VME64: ANSI/VITA 1-1994.

- Xilinx, Inc. LogiCORE™ PCI Design Guide Version 3.0. Xilinx Inc 2001.

2.0 System Architecture

The VME64 to PCI Bridge connects a VMEbus slave interface to a PCI target interface. Communication between the two sides of the bridge is made through a set of four control registers, seven semaphore registers and nine shared memory buffers. Data is loaded into a buffer on one side of the bridge, and unloaded from the other side. The semaphore registers can also be used by system software to determine when and if a buffer is being used. The VMEbus side of the bridge also includes an EEPROM interface for programming Atmel AT17 series devices. Table 2-1 shows the address map from both sides of the bridge.

Table 2-1. Address Map.

VMEbus BYTE Address[4] (Base Offset)	PCI BYTE Address (BAR Offset)	Name	Type	Access Types
0x00000	0x0000	DMC_HW_CONTROL	SREG	R/W
0x00004	(*)	CONFIG_PROM_CMD	REG	R/W
0x00008	(*)	CONFIG_WRITE_DATA	REG	R/W
0x0000C	(*)	CONFIG_READ_DATA	REG	R
0x00010	0x0010	DMC_CMD	SREG	R/W
0x00014	0x0014	DMC_FAULT	SREG	R/W
0x00018	0x0018	DMC_STATUS	SREG	R/W
0x0001C	0x001C	SPECIALREG	SREG	R/W
0x00020	0x0020	SEM_BUF_A	SREG	R/W
0x00024	0x0024	SEM_BUF_B	SREG	R/W
0x00028	0x0028	SEM_BUF_C	SREG	R/W
0x0002C	0x002C	SEM_BUF_D	SREG	R/W
0x00030	0x0030	SEM_BUF_E	SREG	R/W
0x00034	0x0034	SEM_BUF_F	SREG	R/W
0x00038	0x0038	SEM_BUF_G	SREG	R/W
0x0003C – 0x007FF	0x003C – 0x07FF	Unused / Unreserved	-	-
0x00800 – 0x00BFF	0x0800 – 0x0BFF	BUF_A	SMEM	R/W
0x00C00 – 0x00FFF	0xC00 – 0x0FFF	BUF_B	SMEM	R/W
0x01000 – 0x013FF	0x1000 – 0x13FF	BUF_C	SMEM	R/W
0x01400 – 0x017FF	0x1400 – 0x17FF	BUF_D	SMEM	R/W
0x01800 – 0x01BFF	0x1800 – 0x1BFF	BUF_E	SMEM	R/W
0x01C00 – 0x01FFF	0x1C00 – 0x1FFF	BUF_F	SMEM	R/W
0x02000 – 0x023FF	0x2000 – 0x23FF	BUF_G	SMEM	R/W
0x02400 – 0x027FF	0x2400 – 0x27FF	BUF_H	SMEM (**)	R/W
0x02800 – 0x0FFFF	0x2800 – 0xFFFF	Unused / Unreserved	-	-
0x10000 – 0x7FFFF	-	Unused / Unreserved	-	-

Notes:
Access types: 'R': read cycle, 'W': write cycle.
Type: REG (register), SREG (shared register) and SMEM (shared memory) are defined in the glossary.
(*) Not implemented on the PCI side of the bridge.
(**) 'BUF_H' is formed from Xilinx distributed RAM, and is the only region that will accept PCI burst transfers. All other buffers will only respond to single read and write cycles.

[4] By definition, VMEbus and PCI addresses are specified at byte locations. The first address given in the table is the byte address that should be used when accessing the register or memory area using DWORD (32-bit) values.

2.1 VMEbus Interface

The VMEbus slave interface is compatible with ANSI/VITA 1-1994. It has a D32:D16 data interface, and responds to A24 SLAVE base addresses[5] on 512 Kbyte boundaries. Stated another way, the interface responds to A24 base address locations starting at byte locations of 0x000000, 0x080000, 0x100000 and so forth. For example, if the board resides at base address 0x0C0000, then the 'DMC_CMD' register can be accessed at 0x100000 + 0x000010 = 0x100010.

The bridge responds to VMEbus single read and write cycles. It does not respond to the VMEbus block transfer cycle, nor does it support the VMEbus read-modify-write (RMW) cycle.

All registers and buffer memory areas have port sizes of 32-bits (DWORD). All have 16-bit (WORD) granularity, meaning that they can be accessed as 16 or 32-bit quantities. Byte accesses are not supported.

The VMEbus interface is almost, but not totally compliant, with ANSI/VITA 1-1994. That specification requires (under RULE 2.77) that the interface support byte, or D08(EO), data transfer modes. However, this interface only responds to WORD (16-bit) accesses on 2-byte boundaries, and DWORD (32-bit) accesses on 4-byte boundaries[6].

All VMEbus accesses to 'unused' address areas are terminated with the VMEbus [BERR*] signal.

2.2 PCI Interface

The PCI target interface is implemented with a Xilinx LogiCORE PCI interface[7]. Unless otherwise noted, it is compatible with the PCI Version 2.2 bus specification.

The target interface responds to the following PCI command cycles:

- Configuration Read (CBE[3:0] = 1010)
- Configuration Write (CBE[3:0] = 1011)
- Memory Read (CBE[3:0] = 0110)
- Memory Write (CBE[3:0] = 0111)
- Memory Read Multiple (CBE[3:0] = 1100)
- Memory Read Line (CBE[3:0] = 1110)

The PCI interface responds to 64 Kbyte of memory I/O space starting at the address programmed in the BAR0 register.

[5] The VMEbus interface responds to A24 address modifier codes 0x3E, 0x3D, 0x3A and 0x39.
[6] This caveat is required because the target device for the bridge core is the Xilinx Spartan 2 FPGA. The limited number of block memories on that device forces an internal memory architecture (granularity) of 16-bits.
[7] For more details, please refer to the Xilinx LogiCORE PCI Design Guide.

All registers and buffer memory areas located on the PCI target interface have port sizes of 32-bits (DWORD). All have 16-bit (WORD) granularity, meaning that they can be accessed as 16 or 32-bit quantities. The bridge uses a 16/32-bit data interface, which means that it can read and write 16-bit WORD or 32-bit DWORD values. The interface does not support BYTE accesses.

All PCI accesses to 'unused' address areas are terminated with a PCI TARGET ABORT.

Special 'wrapper' circuitry connected to Xilinx LogiCore PCI interface controls how the target interface responds to the PCI commands. During the initial phase of a PCI burst cycle, a binary counter (located inside the wrapper circuitry) latches the starting PCI address. The counter then increments during subsequent phases within the burst cycle, thereby generating the next address. The counter is incremented after every cycle that accesses the high byte of a 32-bit DWORD transfer. The high byte is indicated when [S_CBE(3)] is low. That means that the address counter is incremented after every 32-bit transfer or after a high order 16-bit transfer.

PCI single and burst transactions are supported by the bridge. However, burst transactions[8] are not supported by all memory locations or registers. If a burst transaction is attempted in a memory region that does not support burst transfers, then the bridge responds with a TARGET ABORT termination[9].

Reading or writing to more than one 'SREG' or 'SMEM' area during a single PCI burst cycle may result in an unexpected behavior, and is not recommended.

Also note that the 8-bit address counter rolls over from 0xFF to 0x00 during burst cycles. This means that burst transfers cannot cross boundaries from one shared memory to another. For example, a 256 DWORD burst to the middle of 'BUF_H' will wrap around to the beginning of the same buffer.

The Xilinx LogiCORE PCI handles all transactions with the bridge. The core is implemented as a PCI target, meaning that it cannot initiate transactions. This manual does not attempt to define the operation of the Xilinx LogiCORE PCI interface.

The PCI core returns a PCI Device ID and a PCI Rev ID that are unique to the VMEbus to PCI Bridge core. These are returned from PCI configuration registers 0x00 and 0x08 respectively. The PCI Device ID always returns 0x030. The PCI Rev ID identifies the hardware revision level as shown in the 'Manual Revision' section at the front of this manual.

[8] Burst transactions are bus cycles with more than one data transfer phase.

[9] Burst transactions in excess of one data transfer are allowed as long as the memory structure supports it. This is because the core can be implemented on a number of target devices, memory types and speeds. The Xilinx LogiCore PCI requires that memories must support single clock data transfers. If this type of memory is implemented on the device, then the burst operation is supported. If this type of memory is not implemented with the core, then burst transactions are not supported. For more information please refer the Hardware Reference section of this manual.

2.3 Register Descriptions

Unless otherwise noted, the VMEbus and PCI shared registers (SREG) have identical bit descriptions.

2.3.1 DMC_HW_CONTROL Register

Tables 2-2 and 2-3 show the operation of the DMC_HW_CONTROL Register.

Table 2-2. DMC_HW_CONTROL Definition

Bit #	Name	Write	Read
D00	DMCReset	0 = Local run (*) 1 = Local reset	Readback
D01	SysfailLocalDriver	0 = Assert SYSFAIL (*) 1 = Negate SYSFAIL	Readback
D02	SysfailSystemMonitor	0	0 = SYSFAIL asserted 1 = SYSFAIL negated
D03	AcfailSystemMonitor	0	0 = ACFAIL asserted 1 = ACFAIL negated
D04	PCIResetMonitor	0	0 = No PCI reset 1 = PCI reset
D05-D31	Unused / Unreserved	0	0

Notes:
(1) Always set unused bits to ‘0’ to support future upgrades.
(2) Condition after VMEbus reset or device configuration denoted by: (*).

Table 2-3. DMC_HW_CONTROL Detailed Description	
Bit #	Detailed Description
D00 DMCReset	WRITE: Clearing this bit asserts the [SPARTAN1] output pin on the device. Setting this bit negates the [SPARTAN1] output pin. [SPARTAN1] is an active high signal. READ: Returns the current state of the WRITE bit.
D01 SysfailLocalDriver	WRITE: Clearing this bit causes the local SYSFAIL* driver to assert the VMEbus SYSFAIL* signal. Setting this bit negates the local SYSFAIL driver. For more information about the operation of this bit, please see the *SYSFAIL* Operation* section of this manual. READ: Returns the current state of the WRITE bit.
D02 SysfailSystemMonitor	WRITE: Always set to '0' to support future upgrades of this register. READ: Returns the current state of the VMEbus SYSFAIL* signal. When cleared, the VMEbus SYSFAIL* signal is asserted (i.e. failure mode). When negated, the VMEbus SYSFAIL signal is negated.
D03 AcfailSystemMonitor	WRITE: Always set to '0' to support future upgrades of this register. READ: Returns the current state of the VMEbus ACFAIL* signal. When cleared, the VMEbus ACFAIL* signal is asserted (i.e. AC power failure mode). When negated, the VMEbus ACFAIL* signal is negated.
D04 PCIResetMonitor	WRITE: Always set to '0' to support future upgrades of this register. READ: Returns the current state of the PCI [RST#] signal. When set, the PCI bus is in its reset condition. When cleared, PCI bus is in its 'run' condition.
D05-D31 Unused / Unreserved	WRITE: Always set to '0' to support future upgrades of this register. READ: Always returns '0'.

The VME64 to PCI Bridge supports a standard implementation of the SYSFAIL* line. This includes both its start-up diagnostic and run-time failure capabilities. While these capabilities are not defined by the VMEbus specification, they do conform to standard industry practices.

SYSFAIL* is an open-collector class VMEbus signal. That means it operates as a 'wire-nor' circuit that is asserted if one or more VMEbus modules drives it low. It is negated only after all VMEbus modules negate their on-board SYSFAIL* drivers.

The standard start-up diagnostic operation of SYSFAIL* is shown in Figure 2-1. Under that practice, SYSFAIL* is asserted by all modules in response to the VMEbus SYSRESET* signal. After SYSRESET* is negated, all modules are tested to see if they are operating correctly. As each module passes its test it negates its SYSFAIL* driver. SYSFAIL* is negated only after all modules pass these diagnostic tests. This procedure follows a standard industry practice where all modules boot up in a failed state (i.e. they are assumed to be 'bad' until proven 'good'). SYSFAIL* is negated only after all of the modules are proven to be good.

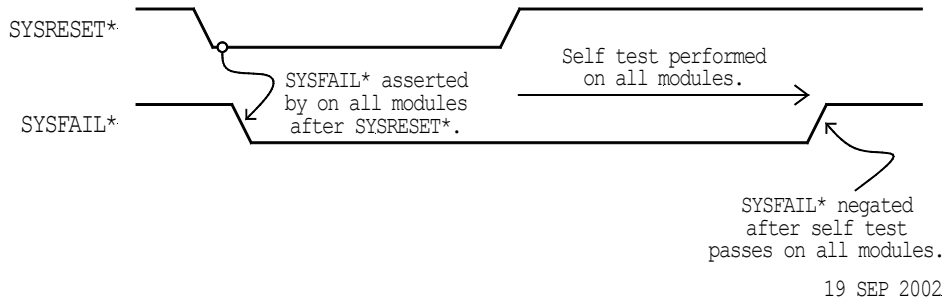


Figure 2-1. Standard VMEbus SYSFAIL* operation.

Very often a red/green LED is attached to a module's local SYSFAIL* driver. This allows quick diagnosis of the system at boot up time by an operator. Under this situation the system is booted and all LEDs turn red. As each module passes its diagnostic test, its LED turns green. In this way the system operator can quickly determine if one or more modules has failed.

A run-time failure can be indicated by a module at any time. To indicate the failure, the module simply asserts SYSFAIL*.

The VME64 to PCI Bridge has several capabilities that are specifically designed to support SYSFAIL* operation. Control and status bits are available from the DMC_HW_CONTROL VMEbus register.

The SYSFAIL* control and status bits are configured so that they look like an emergency stop (ESTOP) button. ESTOP buttons are a common practice in industrial systems. These are generally arranged into a 'wire-or' chain whereby pushing any ESTOP button in the chain causes the system to shut down in a known and controllable manner. This is a standard technique in all systems where a failure could cause damage to life or property. Commonly, ESTOP is a large red button that is pushed to immediately stop all motion or other hazardous conditions (such as de-energizing high voltage power supplies).

A common practice in industrial control systems is to provide an electronic ESTOP button for the control system. This allows the control system to safely power down the system in response to a failure. A common practice in VMEbus systems is to provide each critical module with it's own ESTOP button, and to wire-or them together with the SYSFAIL* line. SYSFAIL* on the backplane can then be wired to a relay (or other logic) that connects into the main system ESTOP chain.

The VME64 to PCI Bridge provides this capability on both sides of the bridge. Furthermore, each side of the bridge can monitor the other side of the bridge. For more information, please refer to the register descriptions.

2.3.2 Configuration EEPROM Registers

The Configuration EEPROM Interface assists in programming of the Atmel AT17 Series of FPGA Configuration EEPROM. The interface is configured by three registers:

- CONFIG_PROM_CMD: Command/status register
- CONFIG_WRITE_DATA: Write data register (8-bits).
- CONFIG_READ_DATA: Read data register (8-bits).

The user is encouraged to study the following data sheets (provided by Atmel) before programming a device:

1) *Programming Specification for Atmel's AT17 and AT17A Series FPGA Configuration EEPROMS*. Atmel Corporation 2002. See www.atmel.com.

2) *AT17LV040 FPGA Configuration EEPROM Memory* (data sheet).

The CONFIG_PROM_CMD register is shown in Tables 2-4 and 2-5. The bits in this register control the various operations of the interface. The CONFIG_READ_DATA and CONFIG_WRITE_DATA registers are for reading and writing data to the EEPROM. Load the lower eight bits of the CONFIG_WRITE_DATA register before issuing a write command, and read the lower eight bits of the CONFIG_READ_DATA register after completion of a read command.

The EEPROM interface can be reset (initialized) in one of two ways: (1) under hardware control in response to a system reset (via the WISHBONE reset line [RST_I]) or (2) under software control in response to the 'ResetPromInterface' bit in the CONFIG_PROM_CMD register.

After a system reset the 'ResetPromInterface' bit is set by hardware. This causes all sections of the EEPROM interface to be initialized. The reset operation may take some time to complete, as the interface operates with a clock speed of about 400 KHz. This means that the reset interval can last in excess of 3 – 6 microseconds. During this time all processor inquires to the 'ResetPromInterface' will return a 'BUSY' state. While busy, the host processor should not attempt to initiate EEPROM accesses.

To reset the EEPROM interface, the VMEbus host processor should perform the following:

1) Reset the interface by setting the 'ResetPromInterface' bit (D00) in the CONFIG_PROM_CMD register.
2) Monitor bit D00 ('BUSY') until the bit is cleared.
3) Perform other operations as needed.

Table 2-4. CONFIG_PROM_CMD			
Bit #	Name	Write	Read
D00	ResetPromInterface	0 = Operate interface (*) 1 = Reset interface	0 = Not busy 1 = Busy
D01	SendStartCondition	0 = No operation (*) 1 = Send start condition	0 = Not busy 1 = Busy
D02	SendStopCondition	0 = No operation (*) 1 = Send stop condition	0 = Not busy 1 = Busy
D03	WriteByteWithAck	0 = No operation (*) 1 = Write data byte	0 = Not busy 1 = Busy
D04	ReadByteWithAck	0 = No operation (*) 1 = Read data byte	0 = Not busy 1 = Busy
D05	ReadByteWithStop	0 = No operation (*) 1 = Read data byte	0 = Not busy 1 = Busy
D06	AckStat	0	0 = ACK received 1 = No ACK received
D07	ConfSerEn	0 = Disable configuration (*) 1 = Enable configuration	Readback
D08	ManualConfigOverride (Manual Override)	0 = Normal operation (*) 1 = Manual configuration	Readback
D09	ConfSerClkOverride (Manual Override)	0 = ConfSerClk cleared (*) 1 = ConfSerClk set	Readback
D10	ConfSerDatOverride (Manual Override)	0 = ConfSerDat cleared (*) 1 = ConfSerDat set	Readback
D11	ConfSerDat (Manual Override)	0	Input State
D12-D31	Unused / Unreserved	0	0

Notes:
(1) Always set unused bits to ‘0’ to support future upgrades.
(2) Condition after system reset or device configuration denoted by: (*).

Table 2-5. CONFIG_PROM_CMD Detailed Description	
Bit #	Detailed Description
D00 ResetPromInterface	WRITE: Setting this bit resets the EEPROM interface. Negating the bit has no effect. This bit need only be set once to initiate a reset. READ: Returns the current state of the reset that was initiated by asserting the WRITE bit. The bit returns '1' while the interface is busy implementing the reset operation, and '0' when it has completed. NOTE: a 'ResetPromInterface' command is initiated in response to a system reset or device configuration. That means that host processor could read this bit as a '1' shortly after a system reset. However, the bit will be eventually be negated when the EEPROM interface has completed its reset cycle.
D01 SendStartCondition	WRITE: Setting this bit initiates an EEPROM START CONDITION. Negating the bit has no effect. READ: Returns the current state of the START CONDITION which was initiated by asserting the WRITE bit. The bit returns '1' while the interface is busy implementing the START CONDITION, and '0' when it has completed.
D02 SendStopCondition	WRITE: Setting this bit initiates an EEPROM STOP CONDITION. Negating the bit has no effect. READ: Returns the current state of the STOP CONDITION which was initiated by asserting the WRITE bit. The bit returns '1' while the interface is busy implementing the STOP CONDITION, and '0' when it has completed.
D03 WriteByteWithAck	WRITE: Setting this bit writes a byte of data to the EEPROM. Negating the bit has no effect. Before setting this bit the data must be loaded into lower eight bits of the CONFIG_WRITE_DATA register. READ: Returns the current state of the write operation which was initiated by asserting the WRITE bit. The bit returns '1' while the interface is busy writing data, and '0' when it has completed. The state of the 'ACK' bit (returned by the EEPROM) is indicated by bit D06 (below).

Table 2-5. CONFIG_PROM_CMD Detailed Description (con't)	
Bit #	Detailed Description
D04 ReadByteWithAck	WRITE: Setting this bit reads a byte of data from the EEPROM and terminates it with an ACK CONDITION. Negating the bit has no effect. READ: Returns the current state of the read operation which was initiated by asserting the WRITE bit. The bit returns '1' while the interface is busy writing data, and '0' when it has completed. Note: After completion of the 'ReadByteWithAck' command, the EEPROM data byte is returned in the lower eight bits of the CONFIG_READ_DATA register.
D05 ReadByteWithStop	WRITE: Setting this bit reads a byte of data from the EEPROM and terminates it with a STOP CONDITION. Negating the bit has no effect. The data is returned in the CONFIG_READ_DATA register. READ: Returns the current state of the read operation which was initiated by asserting the WRITE bit. The bit returns '1' while the interface is busy writing data, and '0' when it has completed. Note: After completion of the 'ReadByteWithStop' command, the EEPROM data byte is returned in the lower eight bits of the CONFIG_READ_DATA register.
D06 AckStat	WRITE: Always write a '0' to this location. READ: Returns the current state of the ACK bit after implementing the WriteByteWithAck and ReadByteWithAck commands. Query this bit to determine if the EEPROM successfully acknowledged the commands.
D07 ConfSerEn	WRITE: Setting this bit enables configuration of the EEPROM. Negating the bit disables configuration. Always set this bit before attempting to read or write to the EEPROM. Always negate it when completed. READ: Returns the state of the WRITE bit.

Table 2-5. CONFIG_PROM_CMD Detailed Description (con't)	
Bit #	Detailed Description
D08 ManualConfigOverride	WRITE: Clearing this bit allows the interface to operate normally. Setting this bit overrides all other functions and allows the ConfSerClk and ConfSerDat to be operated manually. This function is used for test and debugging purposes. READ: Returns the state of the WRITE bit.
D09 ConfSerClkOverride	WRITE: asserts or negates the ConfSerClk signal when the ManualConfigOverride bit is set. READ: Returns the state of the WRITE bit.
D10 ConfSerDatOverride	WRITE: asserts or negates the ConfSerDat signal when the ManualConfigOverride bit is set. READ: Returns the state of the ConfSerDat signal.
D11 ConfSerDat Input	WRITE: Always write a '0' to this location. READ: Returns the state of the ConfSerDat input.
D12 – D31 Unused / Unreserved	WRITE: Always write a '0' to this location. READ: Always returns '0'.

When programming a configuration EEPROM the user is directed to the programming algorithm in the Atmel Programming specification. The interface provides much of the timing. For example, Figure 2-2 shows how the interface implements the first few bytes of data shown in the "Write to Whole Device" algorithm.

The Atmel AT17 series parts must be programmed at a speed that is sufficiently high to guarantee the 'Write Cycle Time' (T_{WR}) indicated in the data sheet. Stated another way, data written to the device must be delivered within a minimum length of time. For example, the Write Cycle Time of the Atmel AT17LV040 is specified as 25.0 ms (max). This means that the theoretical, maximum time required to deliver the data to the EEPROM is:

256 data bytes/page x 9 bits/byte[10] = 2,304 data bits/page

4 address bytes/page x 9 bits/byte = 36 address bits/page

2304 data bits/page + 36 address bits/page = 2,340 bits/page

[10] An address or data byte includes eight bits of information plus one acknowledge bit, or 9 bits total.

The EEPROM interface[11] operates at an clock speed (ECLK) of 393 KHz, and each address or data bit requires 4 clock cycles to complete. This means that the minimum, total Write Cycle Time is:

2,340 bits/page x 4/393 KHz = 23.8 ms

Under best case conditions a block of data takes at least 23.8 ms to write to the EEPROM. However, since the maximum Write Cycle Time for the AT17LV040 is 25.0 ms, this leaves only a very small guard band (1.2 ms) to meet the specification.

In order to achieve this data rate the host processor must deliver data to the interface in an expedient manner. During write cycles this usually means that back-to-back data write cycles (using the 'WriteByteWithAck' command) must occur seamlessly...one after the other. Stated another way, a new 'WriteByteWithAck' command must be issued immediately after the 'BUSY' bit has been negated from a previous command.

Once the EEPROM interface hardware has written a data byte to the device with the 'WriteByteWithAck' command, it negates the associated 'BUSY' bit. Once this happens, the host processor has [Tav] seconds to load the CONFIG_WRITE_DATA register and set the 'WriteByteWithAck' command bit. If the host processor does not meet [Tav] the interface will probably work just fine. However, if [Tav] is exceeded for too many cycles, then there is a risk that maximum Write Cycle Time of 25.0 ms could be exceeded.

The maximum time available [Tav] from 'BUSY' negated to 'WriteByteWithAck' set is found from the relation:

$$Tav = \frac{1}{ECLK} - \frac{6}{CLK\_I} - ECLKsetup$$

Given an EEPROM clock [ECLK] of 393 KHz, a WISHBONE clock speed [CLK_I] of 33.0 MHz and [ECLKsetup] of 1 WISHBONE clock yields a [Tav] of:

$$Tav = \frac{1}{393KHz} - \frac{6}{33MHz} - \frac{1}{33MHz} = 2.34\,uS$$

The system integrator / software programmer should also remember that non real-time operating systems (such as Unix or Windows®) may introduce significant time delays into software execution. These delays can be large when compared to the maximum Write Cycle Time of 25.0 ms. Use of such non deterministic software systems should be carefully evaluated.

Important Notice:

The first byte on the AT17 series EEPROM cannot be reliably read by the FPGA unless power is cycled after programming. For more information, please refer to the programming instructions.

[11] For more information about the EEPROM interface please refer to the 'CEEPROM.VHD' hardware reference located elsewhere in this manual.

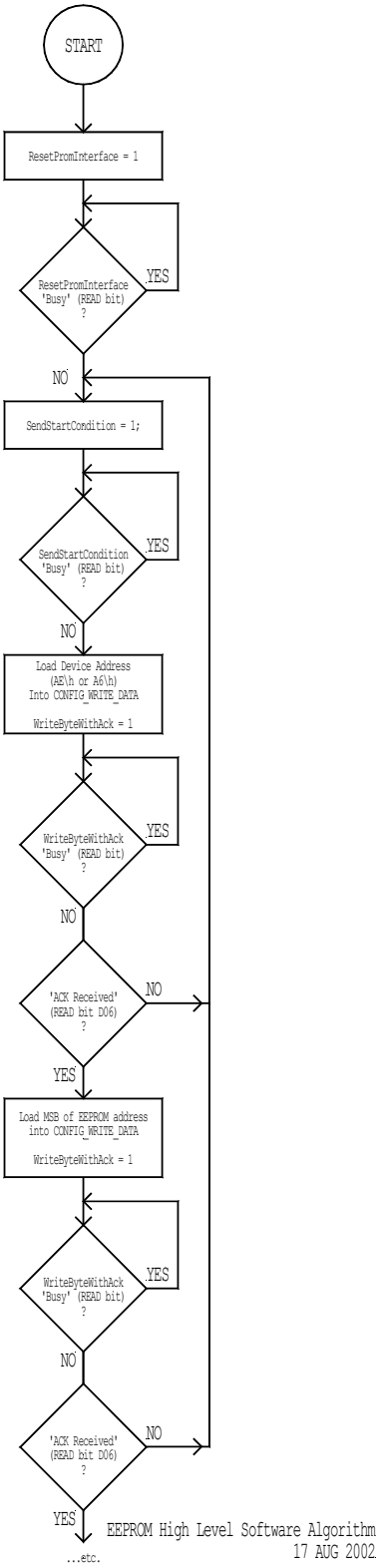


Figure 2-2. High level software control algorithm for the EEPROM Interface.

2.3.5 DMC_CMD Register

DMC_CMD is a 32-bit, user defined, general purpose read/write register.

2.3.6 DMC_FAULT Register

DMC_FAULT is a 32-bit read/write register, with the bit definitions shown in Table 2-6.

Table 2-6. DMC_FAULT Definition

Bit #	Name	Write	Read
D00	WatchdogFaultDSP	0	'0' = No timeout '1' = Timeout
D01-D31	User defined	Latched	Returns current state

2.3.7 DMC_STATUS Register

DMC_STATUS is a 32-bit, user defined, general purpose read/write register.

2.3.8 SPECIALREG Register

SPECIALREG is a 32-bit, user defined, general purpose read/write register.

2.3.9 PCI_SEM_BUF_(A-G)

The seven semaphore registers SEM_BUF_(A-G) are used by VMEbus or PCI system processors to obtain ownership rights for the seven buffer memories. Table 2-7 is representative of the seven semaphore registers. Each SEM_BUF(A-G) semaphore register corresponds to a related BUF(A-G) shared buffer memory.

Table 2-7. SEM_BUF_(A-G) Definition

Bit #	Name	Write	Read
D00	SemaphoreBufferN ($G \geq N \geq A$)	0 = Buffer released 1 = No effect	0 = Grant 1 = Busy
D01-D31	Unused / Unreserved	0	0

The 'SemaphoreBufferN' bit indicates whether a VMEbus or PCI system processor has obtained the memory space. The semaphore does not lock the memory buffer but just provides a mechanism for software to determine if the particular memory buffer is being used. If the semaphore is not used, or is disregarded by software, the associated buffer arbitration still operates normally.

The semaphore is accessed by reading the bit. If the bit is returned as '0', then the processor has obtained ownership of the buffer. If the bit is returned as '1', the buffer is busy. If a processor obtains the buffer by reading '0' (becomes the owner), and the bit is sampled for a second time, then the bit is returned as '1' on the second access.

If a VMEbus and PCI semaphore access occurs at the same time (i.e. during the same clock cycle), then the VMEbus access has priority.

The buffer is released by writing a '0' to the semaphore. Writing a '1' to the bit does not have any effect. The buffer may be released from either the VMEbus or PCI side of the bridge.

2.4 Operation of Shared Memory Buffers and Registers

The VME64 to PCI Bridge contains several different types of memories and buffers. These are classified as SMEM (shared memory), REG (register) and SREG (shared register). The classification for any particular memory or register can be found in the VMEbus and PCI address maps located elsewhere in this manual.

SMEM and SREG types are shared by both sides of the bridge[12]. Each has its own arbiter circuit that resolves any contention between the two sides of the shared resource. For the most part, this operation is transparent to the software programmer. However, in some cases it is important to understand how the arbitration works.

2.4.1 Shared Buffers

There are eight shared memory buffers named 'BUF_A' through 'BUF_H'. These can be used to pass data between the two sides of the bridge. They operate as 32-bit wide memories with 16-bit granularity. This means that information can be passed in WORD (16-bit) and DWORD (32-bit) data formats.

Each buffer operates as an independent shared memory. This means that both sides of the bridge supports full, simultaneous, read/write privileges into each buffer. This provides the software designer with a very flexible mechanism for moving data, as well as an excellent way to test memory from both sides of the bridge.

There may be some undesirable side effects as a result of contention within the shared areas. Specifically, the interface may slow down because a hardware arbiter must grant accesses to one

[12] A third type called 'REG' (register) is a non-shared register. These operate independently from the opposite side of the bridge.

side of the bridge or the other. To alleviate this problem, multiple shared memory buffers are provided. This allows one side of the bridge to access one buffer while the other side of the bridge accesses the other. Under this method there are never any simultaneous memory conflicts, so neither side of the bridge ever waits for an access.

As an option, the software programmer has the ability to use a set of semaphore registers. These can be used to determine which side of the bridge 'owns' a particular buffer. The semaphore does not lock the memory buffer, but rather provides a mechanism for software to determine if the particular memory buffer is being used. If the semaphore is not used, or is disregarded by software, the associated buffer arbitration still operates normally.

The first seven *memory buffers* have an associated semaphore *register* called SEM_BUF_(A-G). These are classified as type SREG, meaning that they also operate as a shared resource with its own hardware arbiter. The arbitration of SMEM and SREG types are identical. An eighth buffer, 'BUF_H', does not have an semaphore register associated with it.

2.4.2 Hardware Arbitration

Accesses from both the VMEbus and PCI side of the bridge proceed normally when they are uncontested. When simultaneous accesses take place in a shared memory or shared register, a hardware arbiter holds off the access from either the VMEbus or the PCI side of the bridge. This means that only one access can take place at any given time. Each SMEM or SREG contains an identical arbiter.

If a simultaneous access occurs from both the VMEbus and PCI sides of the bridge (over a single clock period), then the hardware arbiter will grant the access to the VMEbus side of the bridge, and hold off the PCI side.

2.4.3 PCI Accesses

Each register or buffer arbiter assigns a resource at the beginning of every bus cycle. If the PCI side of the bridge wins the arbitration, then it holds the resource until it is done with its bus cycle. This means that arbitration only takes place once, at the beginning of a bus cycle. For example, if the PCI side of the bridge does a burst transfer into a shared memory, then arbitration occurs immediately before the first data transfer within the burst. The shared memory remains granted to the PCI side during the transfer. At the end of the burst transfer the shared memory is automatically relinquished[13].

[13] Note that behavior during burst transfers depends upon the type of memory used in SMEM. For more information see the 'Memory Requirements' section of this manual.

2.4.4 VMEbus Accesses

The VMEbus side of the bridge works somewhat differently. That side of the bridge supports posted read and write operations. During a posted read or write operation the VMEbus interface captures the data and completes any handshaking protocols with the data source (e.g. a memory buffer). At the same time it initiates handshaking with the data destination. This alleviates the need for the data source to wait until the destination is ready to accept the data.

During a VMEbus posted write operation to an SREG or SMEM region, the bridge does three things: (1) it captures the VMEbus data, (2) it asserts the DTACK* signal to begin termination of the VMEbus cycle and (3) it begins writing the data to its destination. The data is held in the posted write latch[14] until the destination arbiter is ready to accept it. When ready, the interface completes the data transfer into the SREG or SMEM region.

Once the VMEbus posted write latch has captured the data, it continues to hold it until the destination is ready to accept it. In SREG regions the data is delivered within a few clock cycles because the PCI side of the bridge is limited to single transfer cycles. However, in SMEM regions it could take some time for the data to be delivered to its destination. For example, if a 33.333 MHz PCI interface were to transfer 512 words of data during a burst transfer, then the delay before data is accepted is at least 15 uS:

512 WORDS/BURST x 1/(33.333 MHz x WORD) = 15 uS / BURST

Once the VMEbus posted write latch has captured the data, it prevents the interface from responding to further VMEbus cycles until the data has been transferred to its destination. This prevents subsequent VMEbus cycles from overwriting the posted write data latch. The VMEbus cycle will be accepted only after the posted write data has been transferred from the latch.

Under these circumstances the waiting period could be long enough to trigger some VMEbus BERR* watchdog timers (depending upon how they are configured). However, if software uses the semaphore registers (SEM_BUF_A, etc.) to determine ownership of a buffer, then data will always be delivered within a few clock cycles.

The VMEbus posted read cycles operate in a similar, reverse manner. During read cycles from an SREG or SMEM region the VMEbus interface waits for the arbiter to grant the source to the interface. During this interval the VMEbus MASTER participating in the cycle waits until the data is ready. Once the arbiter has granted ownership of the data source to the VMEbus interface it does three things: (1) it captures the read data in the posted read latch, (2) it completes handshaking with the data source and (3) it asserts the VMEbus DTACK* signal to begin termination of the cycle. The data is held in the posted read latch until the VMEbus cycle is completed.

The posted read capability prevents the VMEbus side of the bridge from ‘hogging’ the SREG or SMEM area. Once arbitrated, the source data is delivered to the posted read latch in a few clock cycles. Once the data is delivered, the arbiter can service any pending accesses on the PCI side of the bridge.

[14] The VMEbus posted write latch is physically located in the VPWWRAP hardware section.

2.5 Reset Operation

The VME64 to PCI Bridge SoC responds to both the VMEbus [SYSRESET*] signal and the PCI [RST#] signal. Most sections of the bridge are initialized whenever the VMEbus [SYSRESET*] signal is asserted. Only the Xilinx LogiCORE PCI core is reset in response to the PCI [RST#] signal.

3.0 Hardware Reference

Most of the VME64 to PCI Bridge SoC was created and delivered in the VHDL hardware description language. VHDL source code must be synthesized before operation on a particular target device (such as an FPGA or ASIC). A variety of simulation, synthesis and CASE[15] tools can be used with the core.

Most of the components used by the core are provided with the source code. However, there are a few exceptions. RAM and I/O drivers must be synthesized with entities provided by the FPGA or ASIC vendor. That's because portable, synthesizable RAM and ROM elements are not supported by the VHDL standards. Furthermore, the SoC uses a PCI target interface that was made from a Xilinx LogiCORE PCI element. Xilinx does not provide source code for this core, but rather a 'firm core' in the form of a proprietary Xilinx '.ngo' file. This file is combined with the rest of the SoC at route time.

With the exceptions noted above, the VME64 to PCI Bridge is provided as a 'soft core'. This means that all VHDL source code and test benches are provided with the design. This enables the user to see inside of the design, thereby allowing a better understanding of it. This is useful from both a design and test standpoint. From a design standpoint the user can tweak the source code to better fit the application. From a test standpoint, it allows the user to create custom test benches that incorporate both the core and other entities on the same IC.

The soft core approach allows the VME64 to PCI Bridge to be synthesized and tested with a variety of software tools. This reduces the cost of special VHDL development software. Users should verify that their software tools conform to the IEEE standards listed in the next section of this manual.

3.1 VHDL Simulation and Synthesis Tools

It is assumed by Silicore Corporation that all simulation and synthesis tools conform to the following standards[16]:

- IEEE Standard VHDL Language Reference Manual, IEEE STD 1076-1993.
- IEEE Standard VHDL Synthesis Packages, IEEE STD 1073.3-1997.
- IEEE Standard Multivalue Logic System for VHDL Model Interoperability, IEEE STD 1164-1993.

In most cases the VHDL source code should be fully portable as long as the simulation and synthesis tools conform to these standards. However, if incompatibilities between the source code and the user's tools are found, please contact Silicore Corporation so that the problem can be resolved.

[15] CASE: Computer Aided Software Environment

[16] Copies of the standards can be obtained from: IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ USA 08855 (800) 678-4333 or from: www.ieee.org

It is strongly recommended that the user have a set of VHDL simulation tools before integrating the VME64 to PCI Bridge. These help in two ways: (a) they build confidence that the core synthesizes correctly and (b) they help resolve any integration problems. The simulation tools do not need to be fancy...a simple logical, non-graphical simulator is adequate.

Explicit instructions for synthesis and routing of the core on a Xilinx Spartan 2 FPGA are provided in the source disk. These take the form of batch files that exactly define the various synthesis and routing operations needed to form the final target device. The following tools are used in the source disk examples:

- Xilinx Integrated Software Environment (ISE) 4.2 (FPGA Router)
- Xilinx XST Tools (VHDL Synthesis)
- ModelSim 5.5b (VHDL Simulator)

All original VHDL source files have been edited with a MS-DOS editor. Font style: COURIER (monotype), tab spacing: 4. Almost any editor can be used, but the user may find that the style and formatting of the source code is more readable using this (or a compatible) editor.

3.2 VHDL Portability

Portability of the VHDL source code is a very high priority in the VME64 To PCI Bridge design. It is assumed that the core will be used with a variety of target devices and tools.

Several proven techniques have been used in the source code to enhance its portability. These apply to the synthesizable code, and not to the test benches. These include:

- No *variable* types are used. Variables tend to synthesize unusual logic in some VHDL compilers, and have not been used in the *synthesizable* entities. For example, all counters are designed with logic functions, and not with incremental variables. Variable types are used in the test benches, however.

- No internal three-state buses are used[17]. Some FPGA architectures do not support three-state buses well, and have been eliminated from the core (except for the I/O port interfaces, which are user defined). However, some VHDL synthesis tools will automatically create three-state buses on large multiplexors. This is perfectly acceptable if the target device supports them.

- Synchronous resets and presets are used on flip-flops. No asynchronous resets or presets are used in the design. Most FPGA and ASIC flip-flops will handle synchronous

[17] The Xilinx LogiCORE PCI does contain a three-state bus. Because this core contains licensing provisions which restrict its use to Xilinx parts, and because Xilinx parts do have three-state buses, this is not a problem in this application.

resets and presets very well. The asynchronous resets and presets are less portable, but are still supported by most devices. Asynchronous presets are least portable, and have been eliminated from the design.

- Asynchronous, unintended latches have been eliminated from the design. These are usually the result of incompletely specified *if-then-elsif* VHDL statements.

- Each source file contains one entity/architecture pair. Some simulator and synthesis tools cannot handle more than one entity/architecture pair per file.

3.3 Required Resources on the Target Device

The logic resources required by the VME64 To PCI Bridge are fairly common, and are available on many FPGA and ASIC target devices. However, before synthesis the user should confirm that the following elements are available on the target device:

- At least two global, low skew clock interconnects for [PCLK] and [ECLK]. Most of the logic in the core is synchronous, and the global clocks coordinates all of the internal activity.

- Logic elements such as NAND gates, NOR gates, inverters and D-type flip-flops. Only elements defined by the IEEE STD 1164-1993 standard are used in the core.

- D-type flip-flops with known power-up conditions. The VME64 To PCI Bridge has some internal bits that must be set to pre-defined states after a power-up or configuration reset.

- Fourteen 256 x 16-bit block RAMs, and two 256 x 16-bit distributed RAMs. These are used for shared memory buffers. In order to support PCI burst transfers the RAM elements must be capable of operating within a single clock cycle. For more information see the section below regarding memory integration.

3.3.1 Clock Requirements

Three clocks are required by the VME64 to PCI Bridge. These are called [PCLK], [VCLK] and [ECLK]. [PCLK] stands for PCI CLocK, and is used by the Xilinx LogiCORE PCI IP core. [VCLK] stands for Vme CLocK, and is used by the Silicore VMEcore(tm) element. [ECLK] stands for EEPROM clock, and generates the clock for the EEPROM hardware interface. [ECLK] is formed from [PCLK]. Under certain conditions [PCLK] and [VCLK] may be tied together to form a single clock interconnection. This is done on the VMEbus to PCI Bridge Core. Table 3-1 shows the requirements for the two clocks.

Table 3-1. Allowable Frequency Range for [PCLK] and [VCLK].				
Clock	F_{min}	F_{max}	Duty Cycle	Controlled by
PCLK	0	33.333 MHz	40/60 – 60/40	PCI Rev 2.2
VCLK	33.333 MHz	50.000 MHz	40/60 – 60/40	VITA 1-1994 & VMEcore™

The [PCLK] clocking constraints follow those given in the PCI bus specification. However, the Xilinx LogiCORE PCI has several constraints as well. That core requires that the clock be operated at a fixed frequency[18], and not be driven by a device such as a phase lock loop (PLL). Furthermore, they do not guarantee that the core will operate correctly using industrial speed components[19].

The [VCLK] clocking constraints are governed by the Silicore VMEcore™ IP Core. That core samples an asynchronous bus, and must be provided with a clock using the constraints shown in Figure 3-1. The 33.333 MHz minimum frequency is dictated by the need to sample the asynchronous VMEbus signals fast enough to prevent aliasing problems. However, it can't go above 50.000 MHz because the period of [VCLK] must not exceed the data strobe skew on the backplane.

If [PCLK] and [VCLK] operate at (or near) 33.333 MHz, then they may be tied together. If the user believes that the manufacturing guard band on frequency is insufficient at this frequency, then it is recommended that the two cores be operated at separate frequencies.

[18] The PCI specification allows the PCI clock to vary as long as it remains monotonic and within the duty cycle specification.
[19] Xilinx offers countermeasures for these problems on their web site.

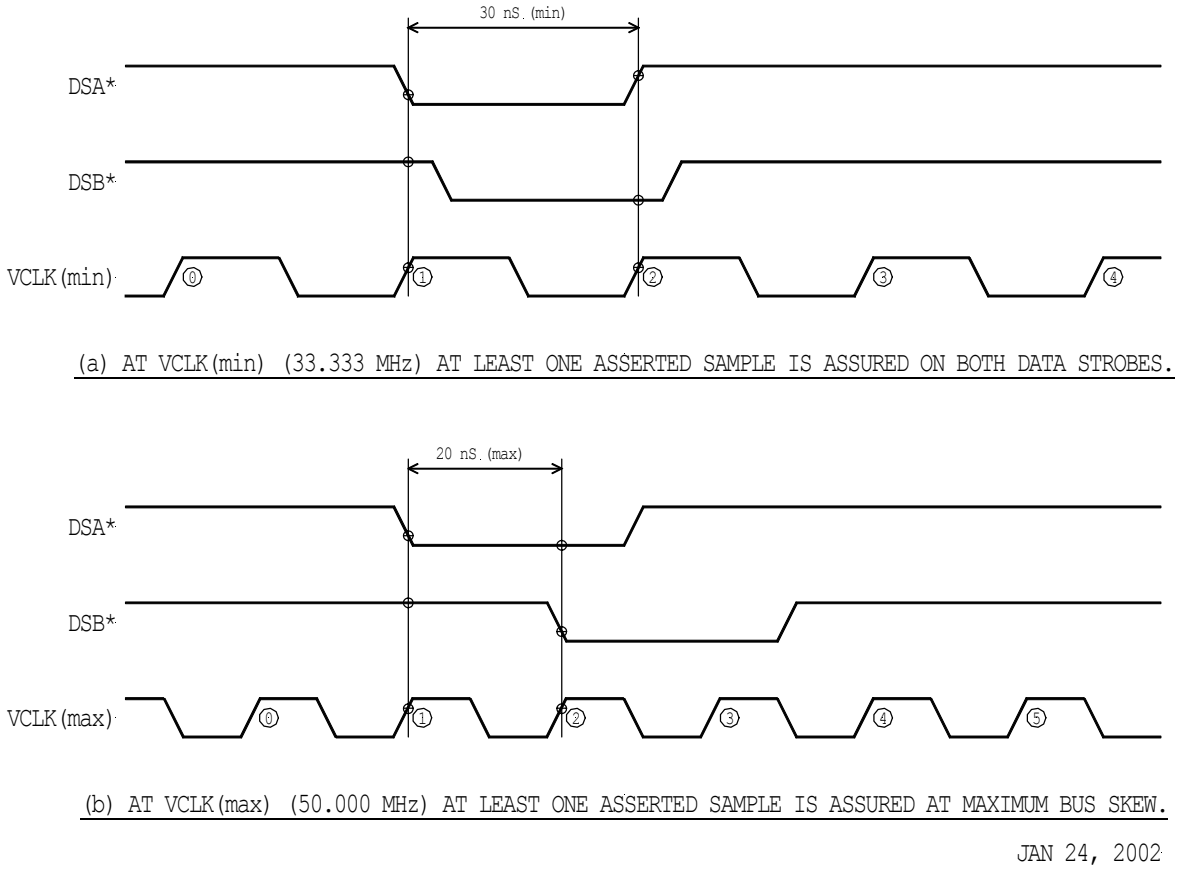


Figure 3-1. VMEbus requirements (constraints).

3.3.2 Memory Requirements

A wide variety of RAM memory elements can be used with the design. However, some types will operate faster and more efficiently than others. In general, if the memory interface closely resembles that needed by the IP cores in the design, then everything will run fast. If the memory is significantly different, then everything will slow down.

The internal architecture of the VME64 to PCI Bridge assumes that all memories conform to something called 'FASM', or the FPGA and ASIC Subset Model[20]. That's because the Xilinx LogiCORE PCI, the Silicore VMEcore and the WISHBONE interconnection all rely on the same type of FASM synchronous RAM elements.

The FASM synchronous RAM model conforms to the generic connection and timing diagram shown in Figure 3-2. During write cycles, FASM synchronous RAM stores input data at the indicated address whenever: (a) the write enable (WE) input is asserted, and (b) there is a rising clock edge.

[20] The original FASM model actually encompasses many type of devices, but here the focus will be on the FASM synchronous RAM models.

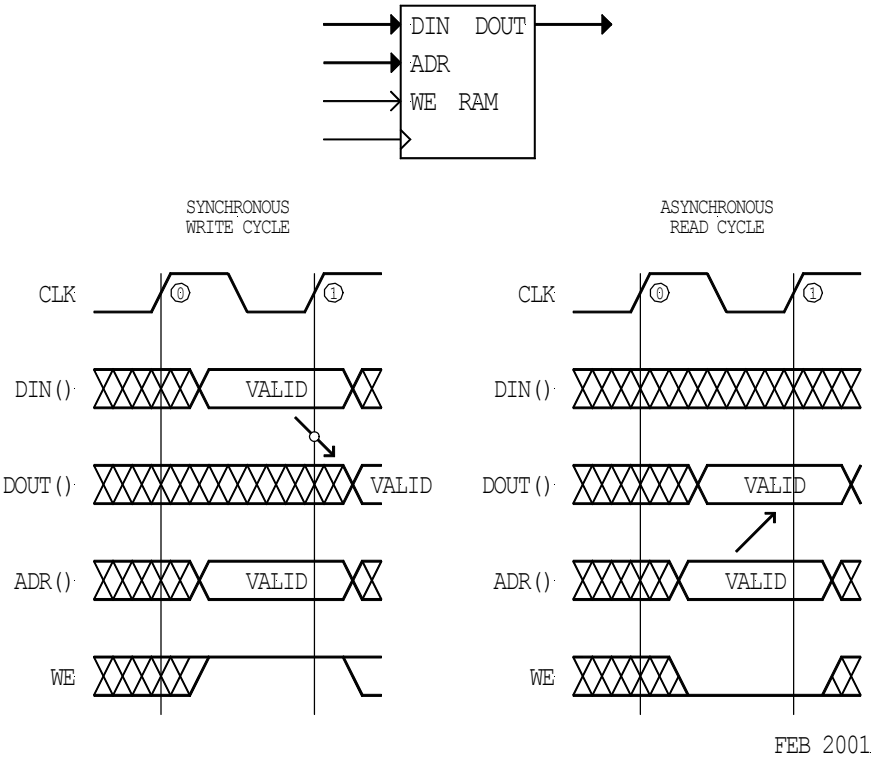


Figure 3-2. Generic FASM synchronous RAM connection and timing diagram.

During read cycles, FASM synchronous RAM works like an asynchronous ROM. Data is fetched from the address indicated by the ADR() inputs, and is presented at the data output (DOUT). The clock input is ignored. However, during write cycles, the output data is updated immediately after the rising clock edge.

The basic advantage behind the FASM synchronous RAM is its ability to work in systems that use single clock data transfer mechanisms. The limiting factor in this design is the Xilinx Logi-CORE PCI IP Core, which requires single clock data transfers during burst read and write cycles. However, this is complicated by the fact that the core does not include any provision for throttling the data transfers.

The Silicore VMEcore and WISHBONE interconnection systems both support single clock data transfers and data throttling.

While most FPGA and ASIC devices provide RAM that follow the FASM guidelines, many also support other types of memories too. Some of these interface smoothly to WISHBONE, while others introduce a wait-state. In all cases that we have found, all modern FPGA and most ASIC devices do support at least one style of FASM memory.

Since a Xilinx IP Core product is used in this design, and relatively large memories are needed, the SoC must support the Xilinx Block SelectRAM+ memories. Unfortunately, these do not

conform to the FASM guidelines, and will not work in single clock systems. They require an extra clock cycle either at the front of the cycle (to register the address) or at the back end of the cycle (to register the data). Although these can be adapted to operate in a single clock configuration (see Figure 3-3), this option was omitted in the design because the Xilinx LogiCORE PCI would not route to these memories at 33 MHz on a Xilinx Spartan 2.

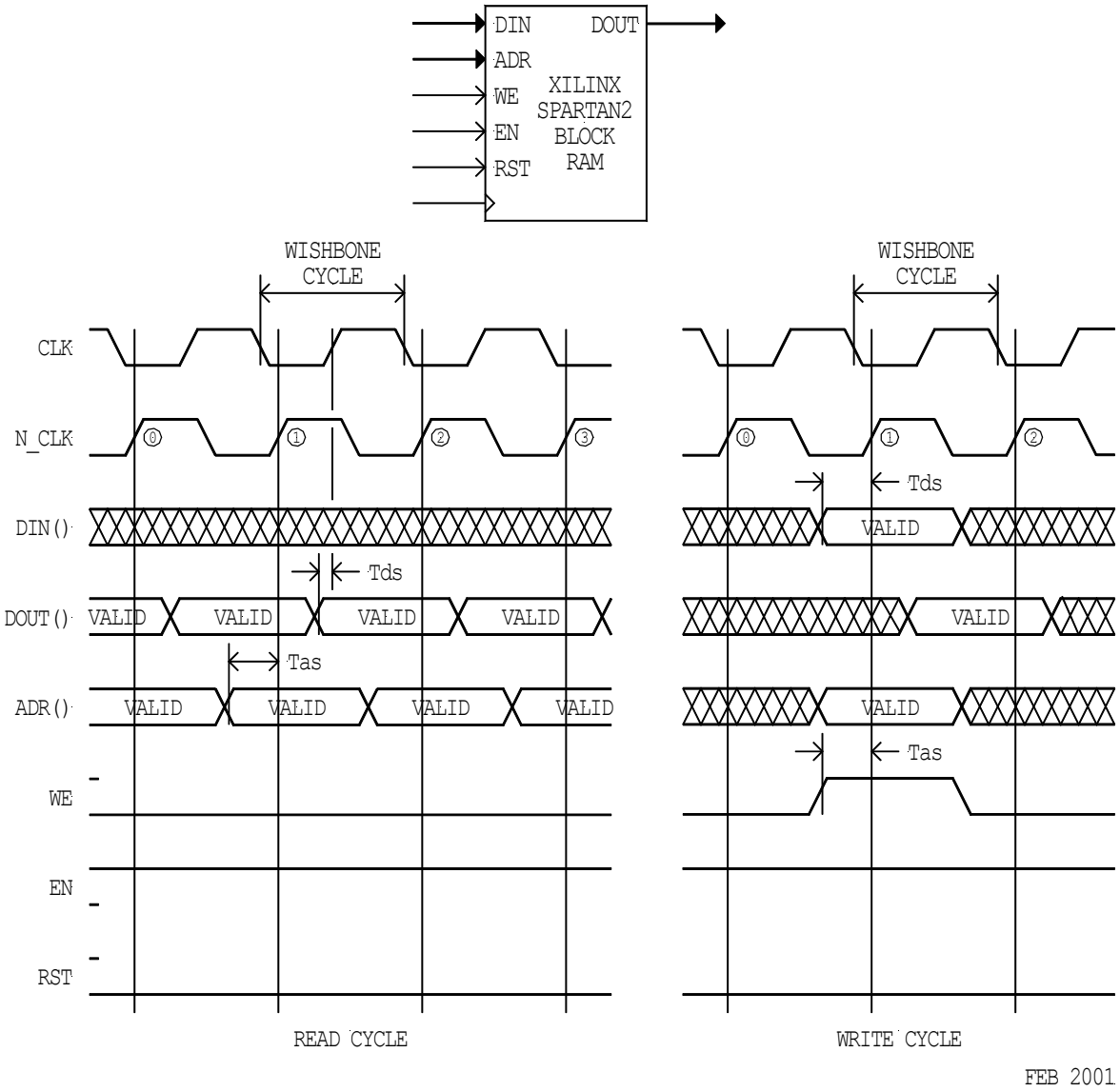


Figure 3-3. Xilinx LogiCORE PCI block RAMs when configured for single cycle operation.

The Xilinx Spartan 2 distributed RAM can support single clock bus cycles. Their main disadvantages are that they contain fewer memory bits than the Xilinx Block SelectRAM+ elements, and consume many logic LUTs (look up tables).

The internal memory buffers and circuits of the VME64 to PCI Bridge are designed to accept either single or double clock memory cycles. This is done by substituting the buffer memories for one or the other. If single clock memories are used then the burst transfers are supported on the PCI target interface. If double clock memories are used, then burst transfers are not supported. If Xilinx Block SelectRAM+ elements are used in a single clock configuration (as described above), the FPGA circuit will be more difficult to route because the clock rate of the internal circuit is effectively doubled from 33 to 66 Mhz.

Table 3-2 shows the Xilinx RAM types used for the buffer memories in the VMEbus to PCI bridge.

Table 3-2. RAMs Used for Buffer Memories.

Memory	RAM Used
BUF_A	2 ea. 256 byte x 16-bit Xilinx Block SelectRAM+
BUF_B	2 ea. 256 byte x 16-bit Xilinx Block SelectRAM+
BUF_C	2 ea. 256 byte x 16-bit Xilinx Block SelectRAM+
BUF_D	2 ea. 256 byte x 16-bit Xilinx Block SelectRAM+
BUF_E	2 ea. 256 byte x 16-bit Xilinx Block SelectRAM+
BUF_F	2 ea. 256 byte x 16-bit Xilinx Block SelectRAM+
BUF_G	2 ea. 256 byte x 16-bit Xilinx Block SelectRAM+
BUF_H	2 ea. 256 byte x 16-bit Xilinx Distributed RAM

4.0 VHDL Entity Reference

The VME64 To PCI Bridge is organized as a series of VHDL entities. These are tied together into an entity called VMEPCIBR_SOC. All of the higher level (first and second tier) entities are described in this chapter in alphabetical order. For more information about the SoC hierarchy please refer to the VMEPCIBR_SOC entity description below.

4.1 CEEPROM Entity

The CEEPROM entity forms a stand-alone interface for programming the Atmel AT17 Series of FPGA configuration EEPROM. Figure 4-1 shows a block diagram and Figure 4-2 shows a functional diagram for the interface. The entity has a WISHBONE compatible data bus interfaces with characteristics shown in Table 4-1.

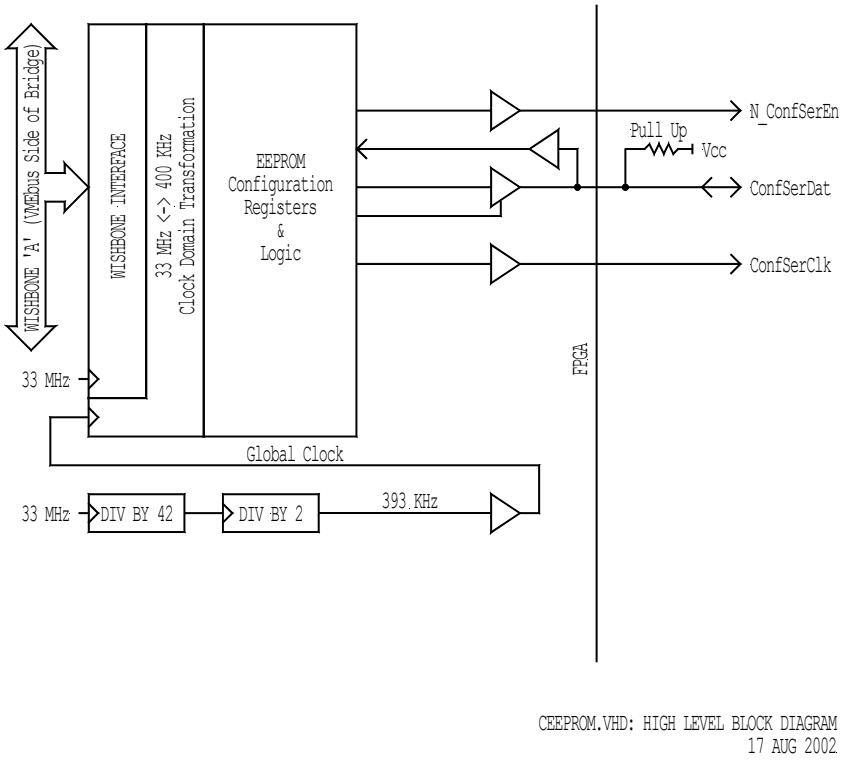


Figure 4-1. Block diagram of the configuration EEPROM Interface.

Data transactions over the data interface are synchronized to the WISHBONE clock [CLK_I]. The entity also includes a clock divider circuit to generate an EEPROM control clock called [ECLK__O]. The nominal frequency of [ECLK_O] is 393 KHz when the [CLK_I] is operated at 33 MHz. The clock divider value can be adjusted to any frequency by editing the VHDL entity. Any frequency of [CLK_I] and [ECLK_O] can be used as long as the frequency of [CLK_I] is at least four times the frequency of [ECLK_O].

EEPROM control clock [ECLK_O] is routed out of the module and then back in again via clock signal [ECLK_I]. This allows the [ECLK_O] signal to be routed to the highest level system entity so that it can drive a global clock net. This allows a global clock net buffer to reside at the highest system level for convenience.

Table 4-1. WISHBONE DATASHEET for the CEEPROM Entity	
General Description	SLAVE interface for the Atmel AT17 series of FPGA configuration EEPROM.
WISHBONE Revision Level	B.2
Supported WISHBONE Cycles	SLAVE: READ/WRITE
Data port, size	32-bit
Data port, granularity	16-bit
Data port, maximum operand size	32-bit
Data transfer ordering	BIG ENDIAN or LITTLE ENDIAN
Data transfer sequencing	None
Signal Description	All WISHBONE signal names are identical to those defined in the specification.
Terminating Signals	The WISHBONE interface on both ports support only the [ACK_O] terminating signal.

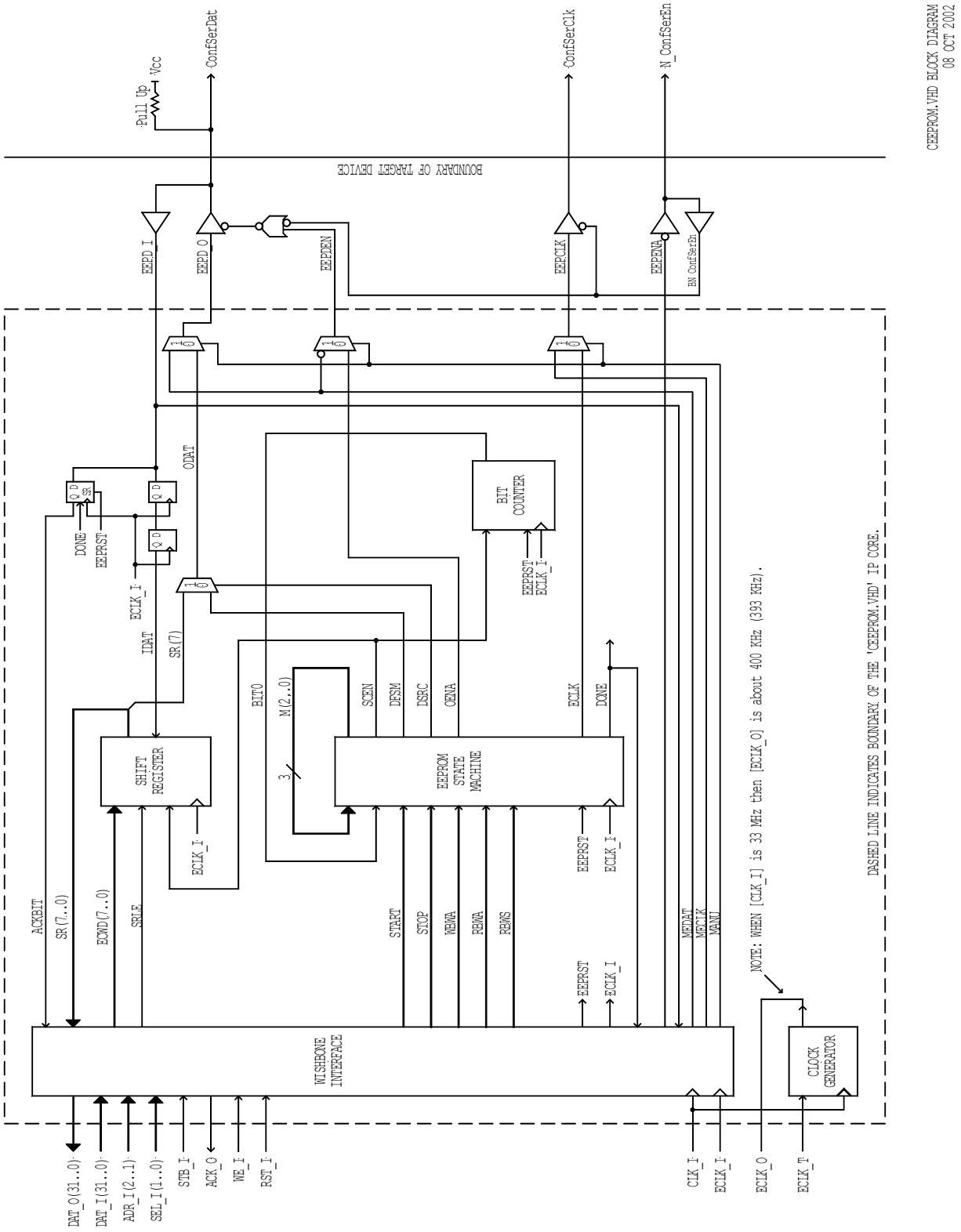


Figure 4-2. Functional diagram of CEEPROM.VHD.

The [EEPENA] signal is asserted whenever the 'ConfSerEn' bit (D07) is set in the CONFIG_PROM_CMD register. This indicates that the EEPROM is enabled for configuration. It also indicates that the three-state clock and data pin drivers on the target device are enabled. External hardware (other than the EEPROM itself) must refrain from driving these signals when [EEPENA] is asserted.

The WISHBONE interface includes all registers that are accessible through the interface. They operate at the nominal WISHBONE clock speed of 33 MHz. The WISHBONE interface also includes a clock domain transformation circuit as shown in Figure 4-3(a). Figure 4-3(b) describes the relationship between the two clock domains. The clock transformation circuit allows two clock domains to communicate seamlessly. Figure 4-4 shows a timing diagram for the circuit.

The EEPROM state machine controls all low frequency activity for the EEPROM interface, with its state diagram shown in Figure 4-5. The state machine forms an instruction set with six instructions. Five of these instructions correspond to the bits in the CONFIG_PROM_CMD register described elsewhere in this manual. A sixth instruction (WAIT) is a type of NOP (no operation). When an instruction is requested (e.g. SendStartCondition), the state machine generates four 'RISC type' instructions. These, in combination with some control bits, handshakes with the EEPROM. A loop instruction is formed with a the BIT_COUNTER process. Figures 4-6 and 4-7 shows the high-level timing for the EEPROM.

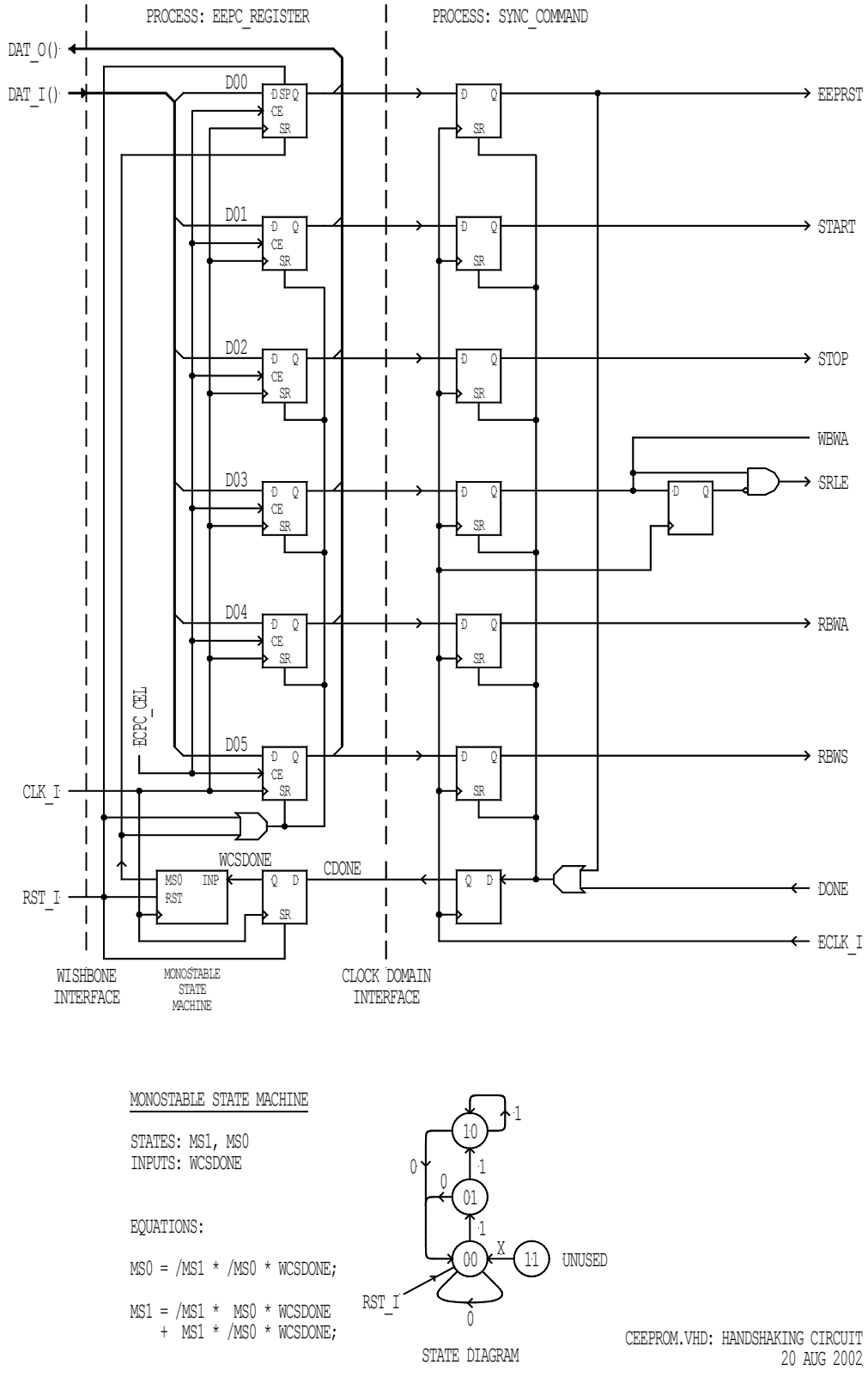


Figure 4-3(a). Clock domain transformation circuit.

```
NOTE: FOR PROPER OPERATION OF EACH BIT,THE FREQUENCY OF 'CLK_I'
MUST BE AT LEAST FOUR TIMES THE FREQUENCY OF 'ECLK'.

BECAUSE:

   1 CLK_I FLIP-FLOP SET-UP
   1 CLK_I SYNCHRONIZATION
 + 1 CLK_I MONOSTABLE STATE MACH.
 + 1 CLK_I DATA FLIP-FLOP
 + 2 CLK_I WISHBONE RMW CYCLE
 ---------------------------------------------
 = 6 CLK_I OVERHEAD         PER 1 ECLK
 + 1 ECLK  FLIP-FLOP SET-UP PER 1 ECLK



FURTHERMORE, IT IS RECOMMENDED THAT THE FREQUENCY OF [CLK_I] BE SUFFICIENTLY
HIGHER THAN THE FREQUENCY OF [ECLK] TO ALLOW FOR A
HOST PROCESSOR TO READ AND WRITE TO THE BITS AND
RESPOND ACCORDINGLY.

THE MAXIMUM RESPONSE TIME OF THE HOST PROCESSOR CAN BE FOUND THUSLY:

        1       6
Tav =  ----- - ----- - 1 ECLK_MAX_FF_SET-UP
       ECLK    CLK_I

FOR EXAMPLE:

  F, CLK_I = 33.000 MHz
  F, ECLK  =  0.400 MHz

         1        5        1
Tav =  ------- - ------ - ------ = 2.32 uS
      0.4 MHz   33 MHz   33 MHz

FOR MORE INFORMATION, SEE THE TIMING DIAGRAM.
                                              CEEPROM.VHD: HANDSHAKING CIRCUIT
                                                                  20 AUG 2002
```

Figure 4-3(a). Clock domain transformation circuit (con't).

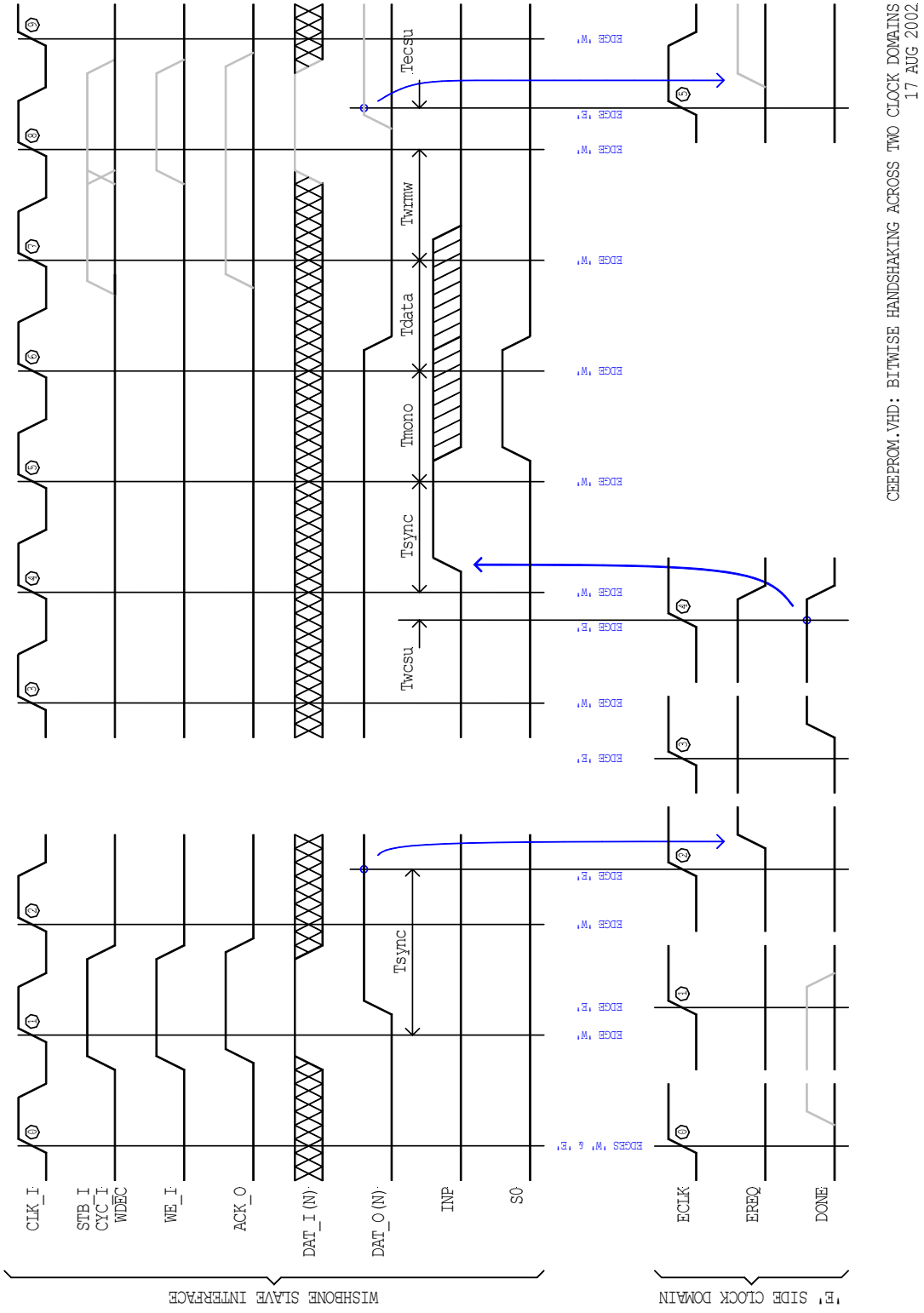


Figure 4-4. Timing for the clock domain transformation circuit.

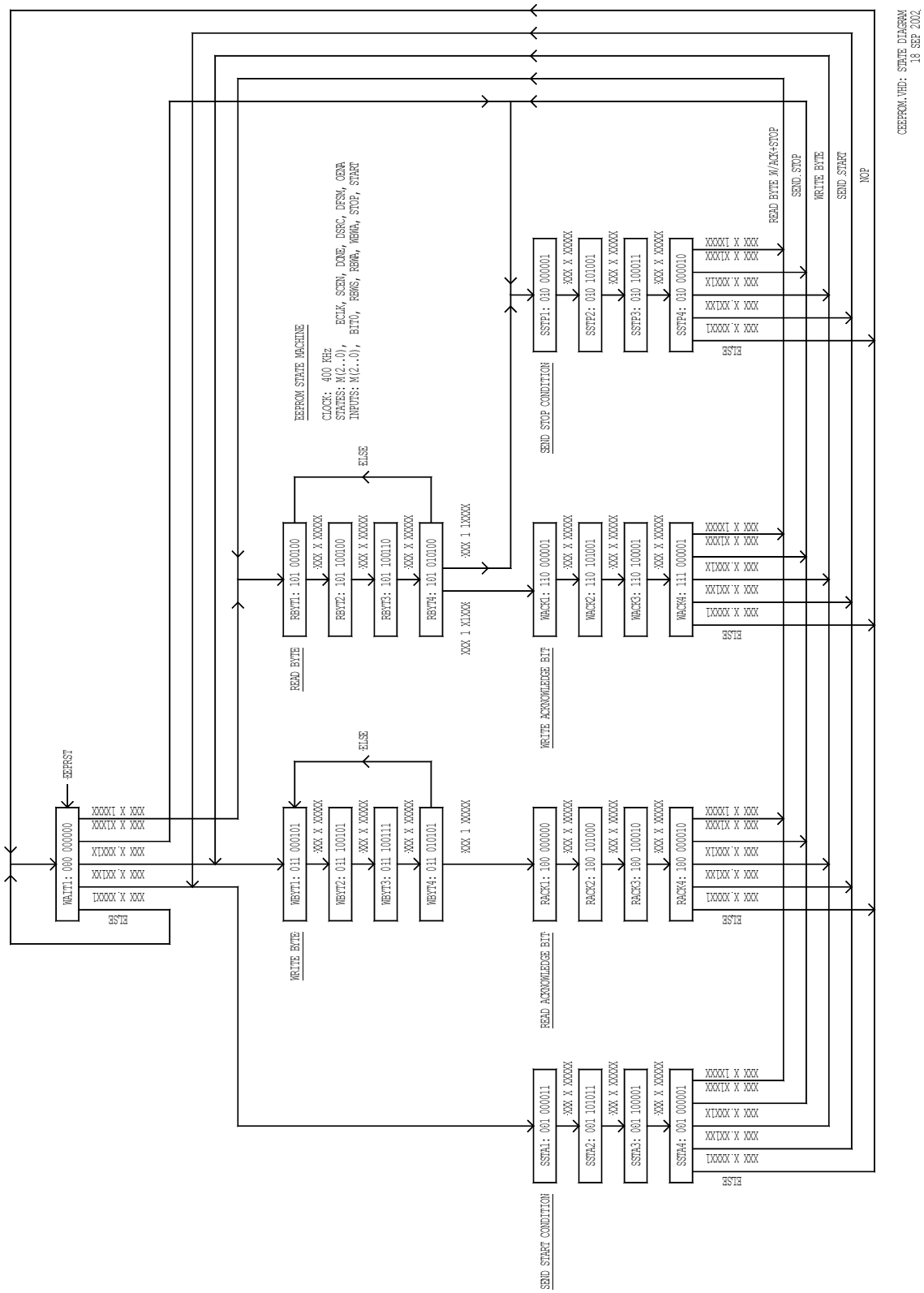


Figure 4-5. State diagram for EEPROM State Machine.

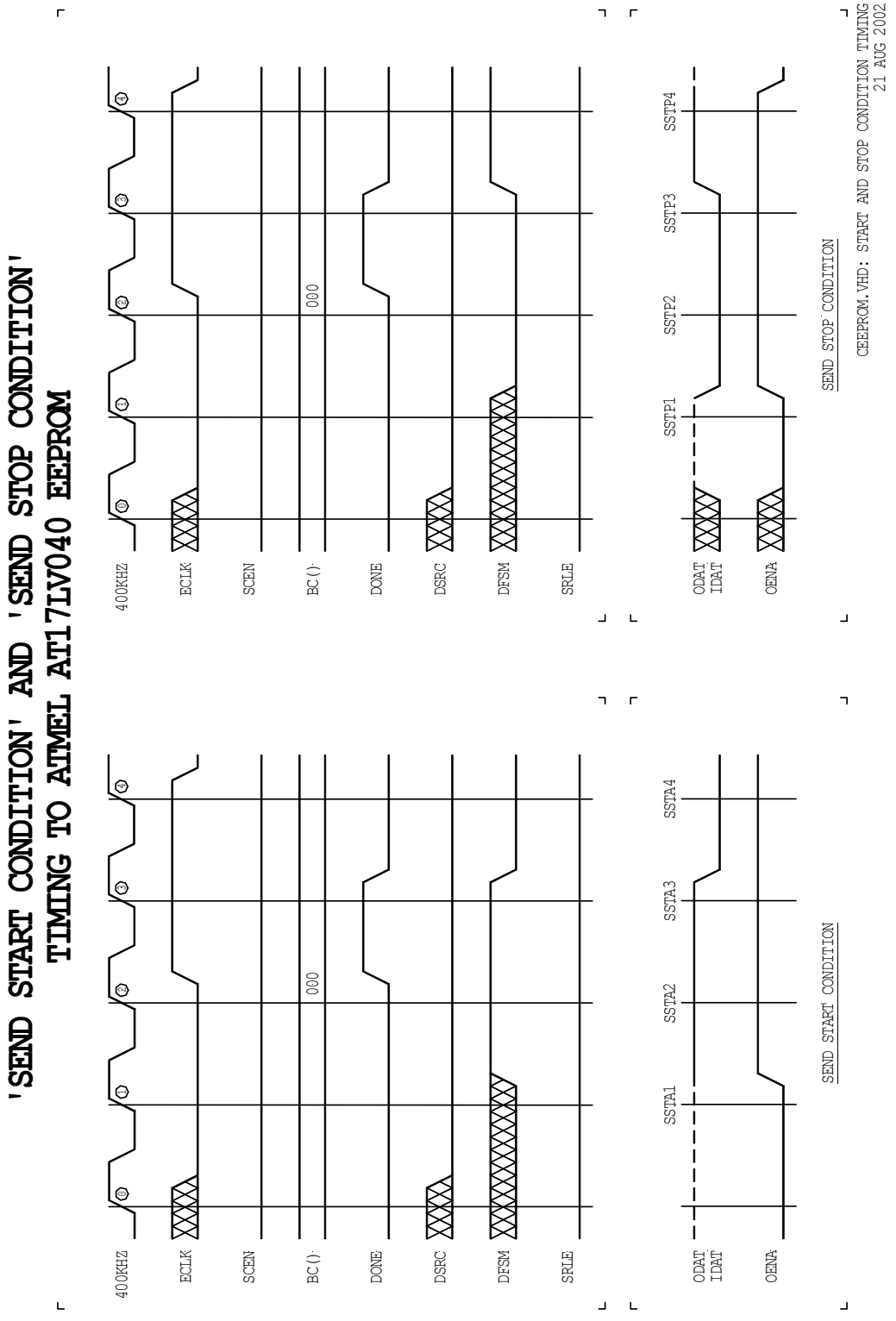


Figure 4-6. EEPROM START and STOP condition timing.

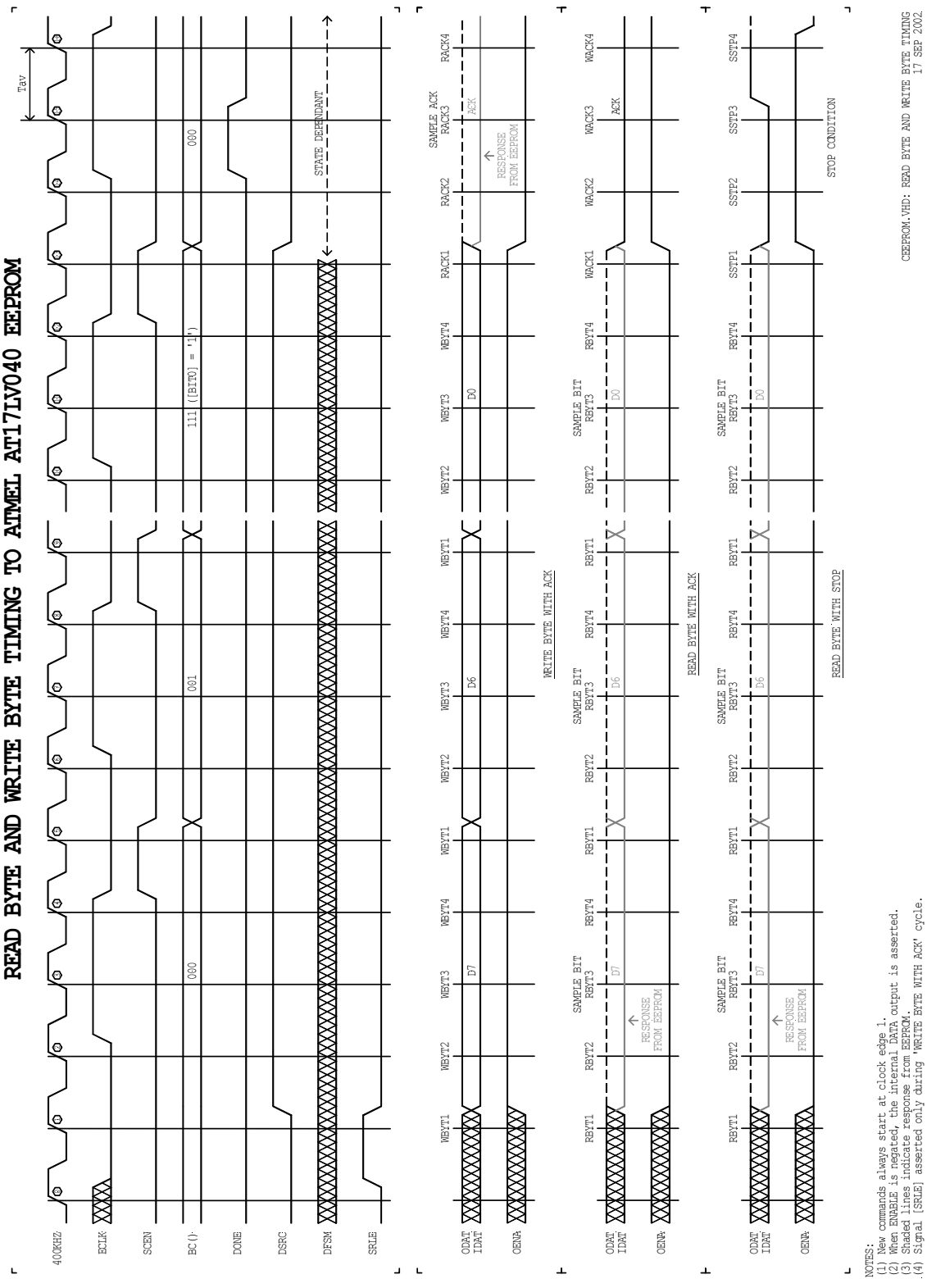


Figure 4-7. EEPROM read and write byte timing.

4.2 MISCREG Entity

The MISCREG entity contains all logic for controlling the lower seven registers in the VMEP-CIBR address map. It contains two WISHBONE interfaces named ‘A’ and ‘B’. In the VMEP-CIBR SoC the ‘A’ side is connected to the WISHBONE interconnection served by VMEbus, and the ‘B’ side is connected to the interconnection served by the PCI interface. The entity also contains all the arbitration logic necessary for handling accesses from the dual ‘A’ and ‘B’ ports.

Figure 4-8 shows a block diagram of the MISCREG entity, and Table 4-2 shows the WISHBONE DATASHEET for the interfaces. The seven registers handled by the entity include:

- DMC_HW_CONTROL
- CONFIG_PROM_CMD (see the CEEPROM entity)
- CONFIG_WRITE_DATA (see the CEEPROM entity)
- CONFIG_READ_DATA (see the CEEPROM entity)
- DMC_CMD
- DMC_FAULT
- DMC_STATUS

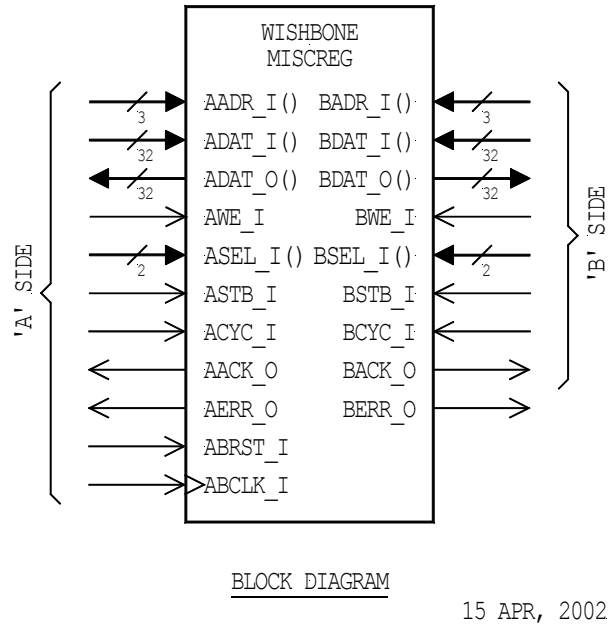


Figure 4-8. MISCREG block diagram.

The MISCREG arbiter and arbiter timing are shown in Figures 4-9 and 4-10 respectively.

Table 4-2. WISHBONE DATASHEET for the MISCREG Entity	
General Description	Logic for the lower seven registers in the system. Contains dual WISHBONE ports named 'A' and 'B'. This datasheet applies to both interfaces.
WISHBONE Revision Level	B.2
Supported WISHBONE Cycles	SLAVE: READ/WRITE
Data port, size	32-bit
Data port, granularity	16-bit
Data port, maximum operand size	32-bit
Data transfer ordering	BIG ENDIAN or LITTLE ENDIAN
Data transfer sequencing	None
Signal Description	All WISHBONE signal names are identical to those defined in the specification, except that they have an 'A' or 'B' at the front. The 'A' and 'B' refer to PORT A and PORTB respectively.
Terminating Signals	The WISHBONE interface on both ports support [ACK_O] and [ERR_O] terminating signals. [ERR_O] is generated when accesses to the unused/reserved registers are attempted.

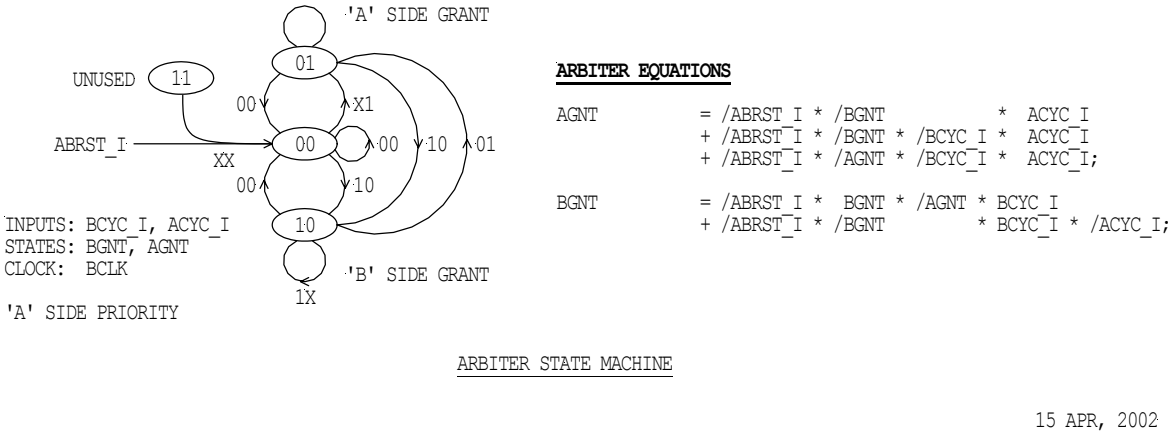


Figure 4-9. MISCREG arbiter.

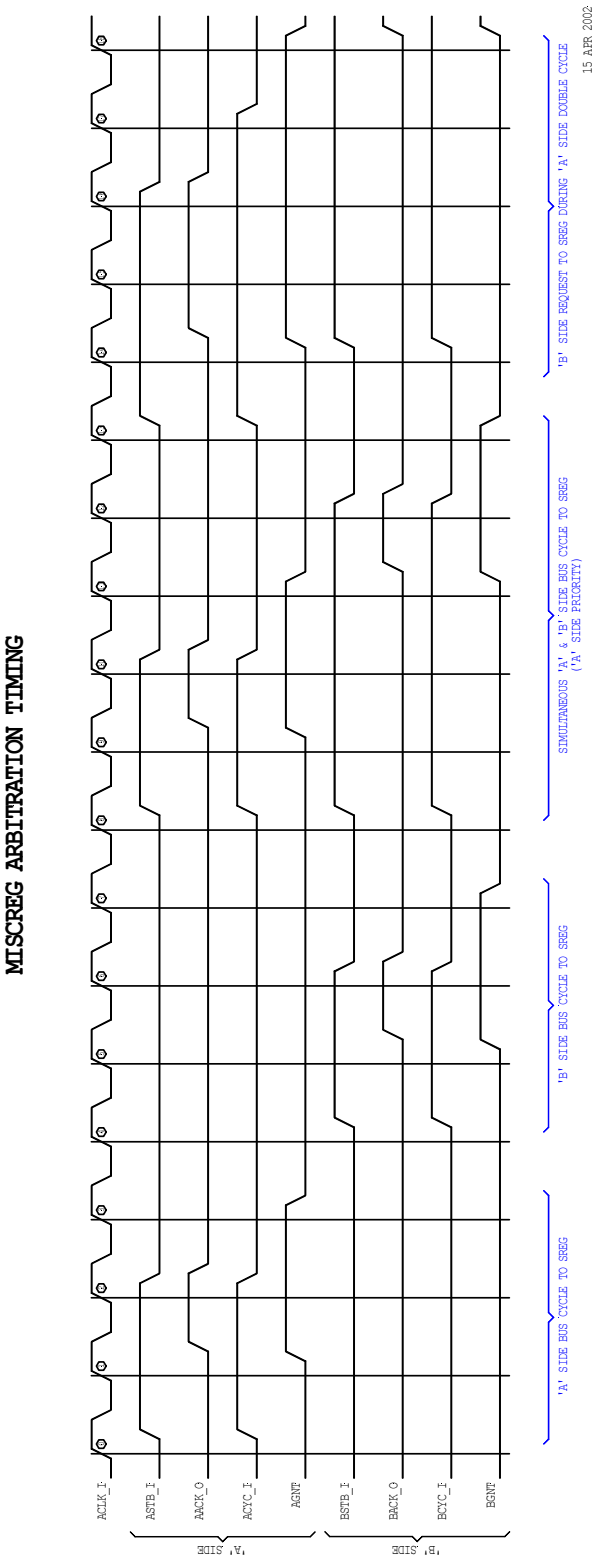


Figure 4-10. MISCREG arbiter timing.

4.3 PCIWRAP Entity

The PCIWRAP entity is a wrapper between the Xilinx LogiCORE(tm) PCI 32-bit 33 MHz IP Core and the WISHBONE SoC interconnection. As shown in the system block diagram of Figure 4-11, the wrapper establishes a PCI interface to the SoC. The wrapper is synthesized from the 'PCIWRAPc.VHD' file.

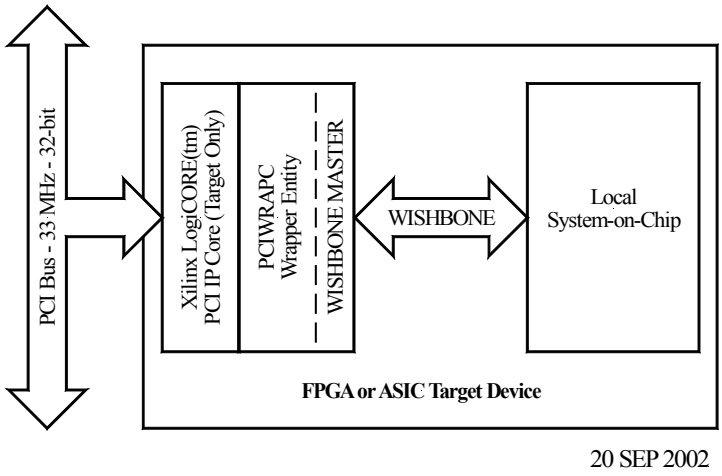


Figure 4-11. System block diagram showing the PCIWRAPc wrapper.

Figure 4-12 shows a functional diagram of the PCIWRAPc entity. Each box in the diagram corresponds to a VHDL process (as described below) of the same name.

The left side of the block diagram shows the signals connected to the PCI core. These are synthesized, routed and connected to the PCI core. The PCI core is delivered as a 'black box', so the signals must conform to the Xilinx specifications. The reader is directed to the Xilinx LogiCORE™ PCI Design Guide (Ver 3.0 – March 16, 2002 or later) for a complete description of the PCI core and signal names.

The PCI target interface responds to the following PCI commands (cycles):

- Configuration Read (CBE[3:0] = 1010)
- Configuration Write (CBE[3:0] = 1011)
- Memory Read (CBE[3:0] = 0110)
- Memory Write (CBE[3:0] = 0111)
- Memory Read Multiple (CBE[3:0] = 1100)
- Memory Read Line (CBE[3:0] = 1110)

The PCI wrapper supports BYTE granularity. However, the WISHBONE interface can be easily configured for WORD and DWORD granularity as well.

All PCI accesses to 'unused' address areas are terminated with a PCI TARGET ABORT.

The wrapper circuitry connected to Xilinx LogiCore PCI interface controls how the target interface responds to the PCI commands. During the initial phase of a PCI burst cycle, a binary counter (located inside the wrapper circuitry) latches the starting PCI address. The counter increments during subsequent phases within the burst cycle, thereby generating the next address. The counter is incremented after every cycle that accesses the high byte of a 32-bit DWORD transfer. The high byte is indicated when [S_CBE(3)] is low. That means that the address counter is incremented after every 32-bit transfer or after a high order 16-bit transfer.

PCI single and burst transactions are supported by the wrapper. However, burst transactions[21] may not necessarily be supported by all memory locations or registers. If a burst transaction is attempted in a memory region that does not support burst transfers, then the memory should respond with the WISHBONE [ERR_I] signal, which causes the wrapper to respond with a TARGET ABORT termination[22].

The wrapper is implemented as a PCI target, meaning that it cannot initiate PCI transactions.

The right side of the block diagram shows the signals connected to the WISHBONE MASTER interface. The interface supports 8, 16 and 32-bit data transfers and a 32-bit address bus. Table 4-3 is the WISHBONE DATASHEET that specifies the interface. The reader is directed to the WISHBONE specification, revision B.2 for a complete description of the WISHBONE MASTER and related signal names.

[21] Burst transactions are bus cycles with more than one data transfer phase.

[22] Burst transactions in excess of one data transfer are allowed as long as the memory structure supports it. This is because the core can be implemented on a number of target devices, memory types and speeds. The Xilinx LogiCore PCI requires that memories must support single clock data transfers. If this type of memory is implemented on the target device, then the burst operation is supported. If this type of memory is not implemented on the target device, then burst transactions are not supported. For more information please refer the Hardware Reference section of this manual.

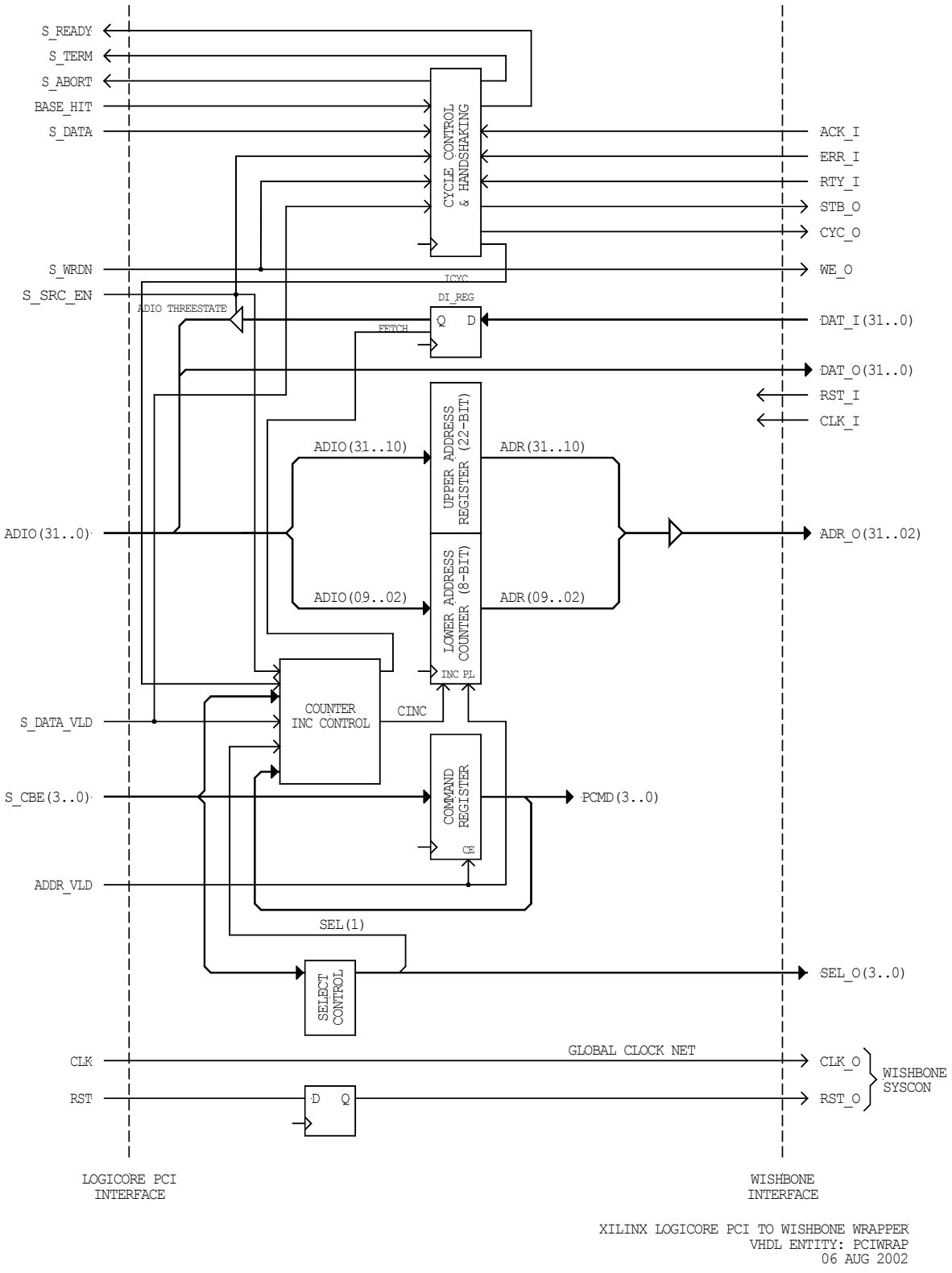


Figure 4-12. Functional diagram of the PCIWRAPC entity.

Table 4-3. WISHBONE DATASHEET for the PCIWRAP Interface.	
General Description	Wrapper between a 32-bit Xilinx Logi-CORE(tm) PCI interface and WISHBONE SoC interface.
WISHBONE Revision Level	B.2
Supported WISHBONE Cycles	MASTER: READ/WRITE MASTER: BLOCK READ/WRITE
Data port, size	32-bit
Data port, granularity	8-bit
Data port, maximum operand size	32-bit
Data transfer ordering	LITTLE ENDIAN
Data transfer sequencing	Sequential (burst) data transfers must be made from lower to higher addresses. The internal address counter rolls over whenever a most significant byte is transferred.
Signal Description	All WISHBONE signal names are identical to those defined in the specification. Refer to the signal descriptions for more details.
Terminating Signals	The WISHBONE interface supports all three termination signals: [ACK_I], [RTY_I] and [ERR_I]. See the text for more information about the operation of these signals.

4.3.1 PCIWRAP Timing Conversion

PCIWRAP is a VHDL ‘wrapper’, which means that it is a chunk of code that converts the PCI core signals to the WISHBONE MASTER signals. Figures 4-13 and 4-14 show the relationship between the PCI bus signals and the Xilinx PCI core back end (user application) signals. Diagrams for burst read and write cycles are shown. Figure 4-15 and 4-16 show the relationship between the PCI core back end (user application) signals and the WISHBONE MASTER interface signals. Diagrams for read and write cycles are shown.

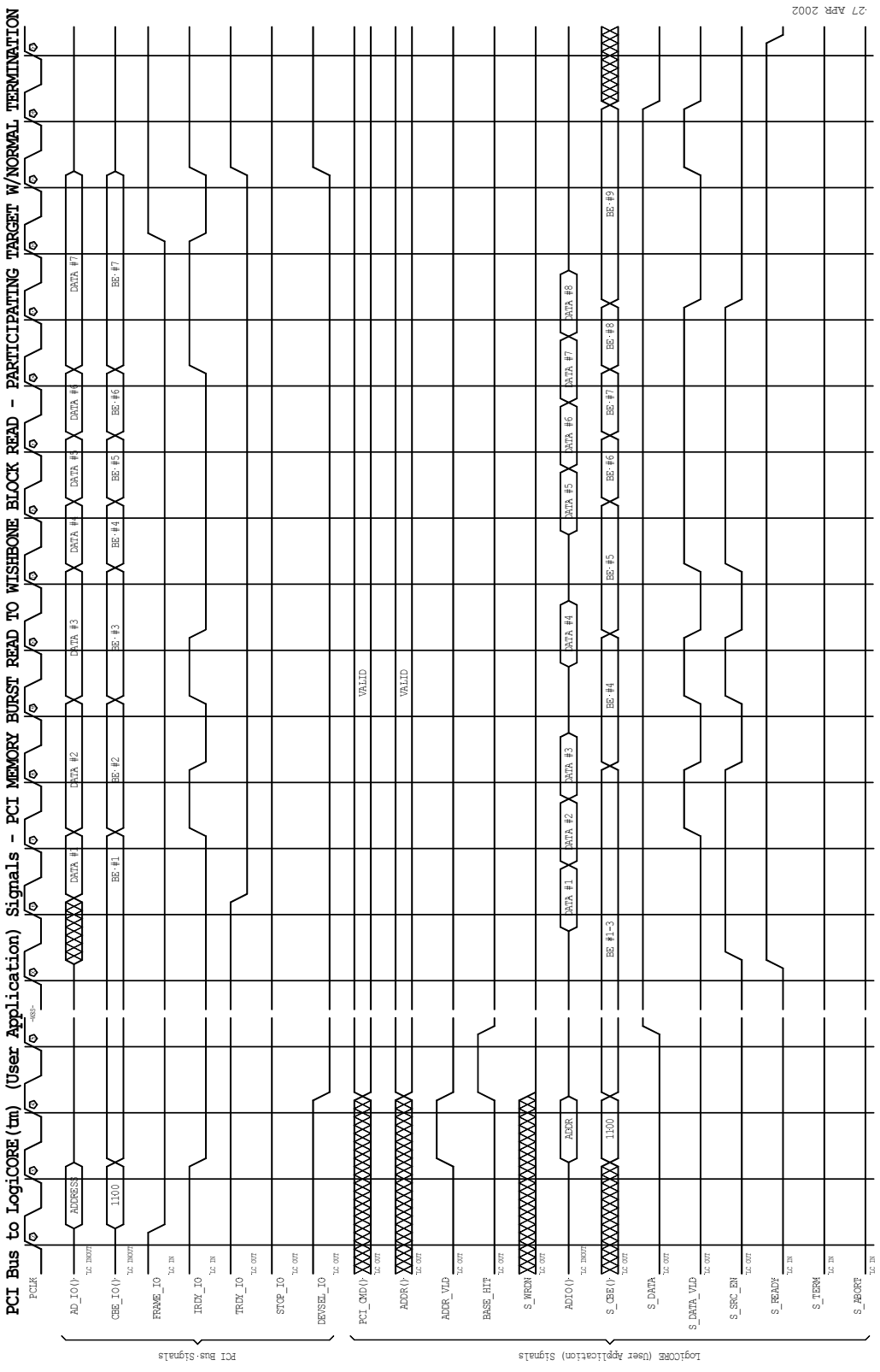


Figure 4-13. READ timing between PCI and Xilinx PCI core back end signals.

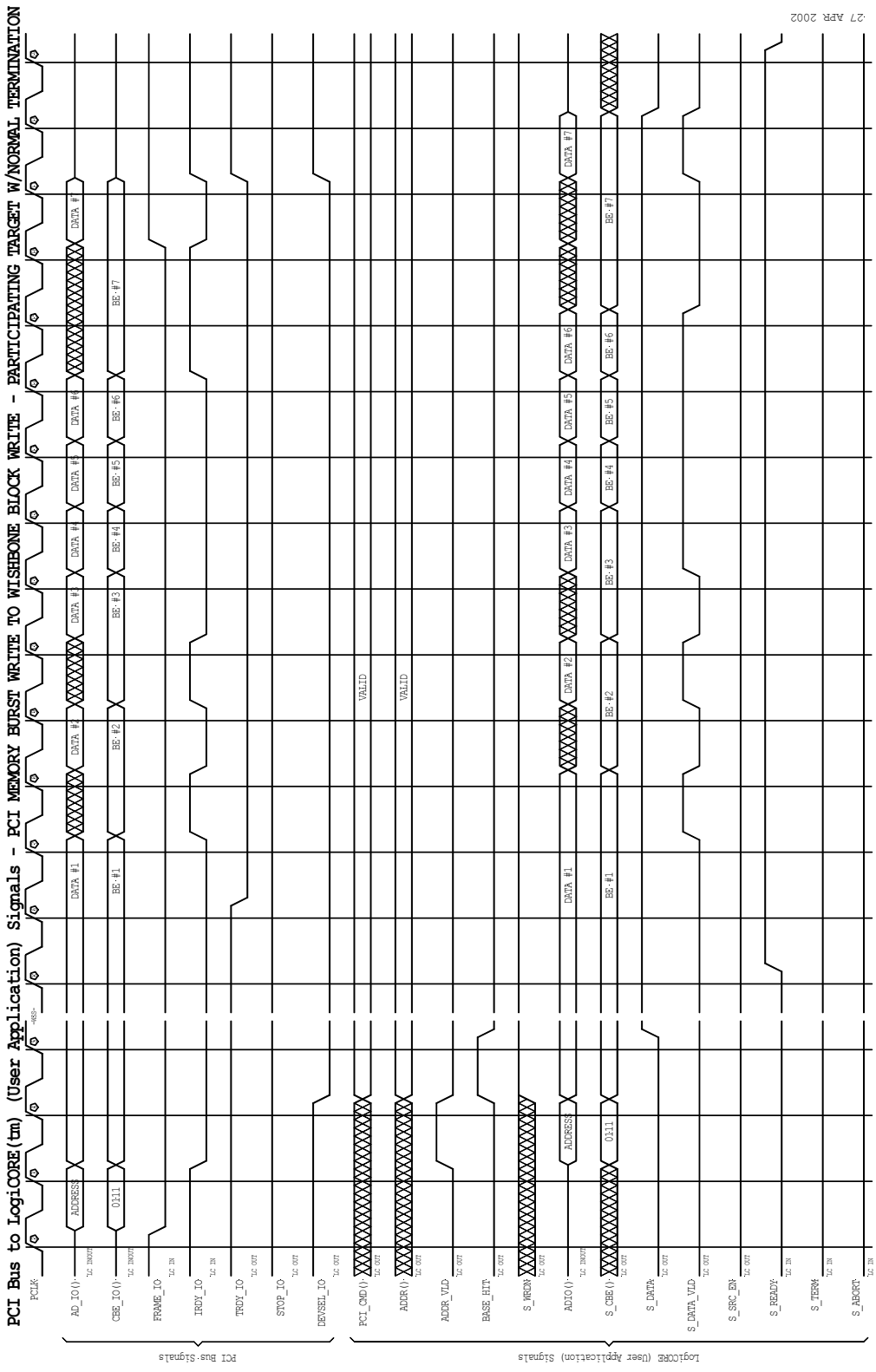


Figure 4-14. WRITE timing between PCI and Xilinx PCI core back end signals.

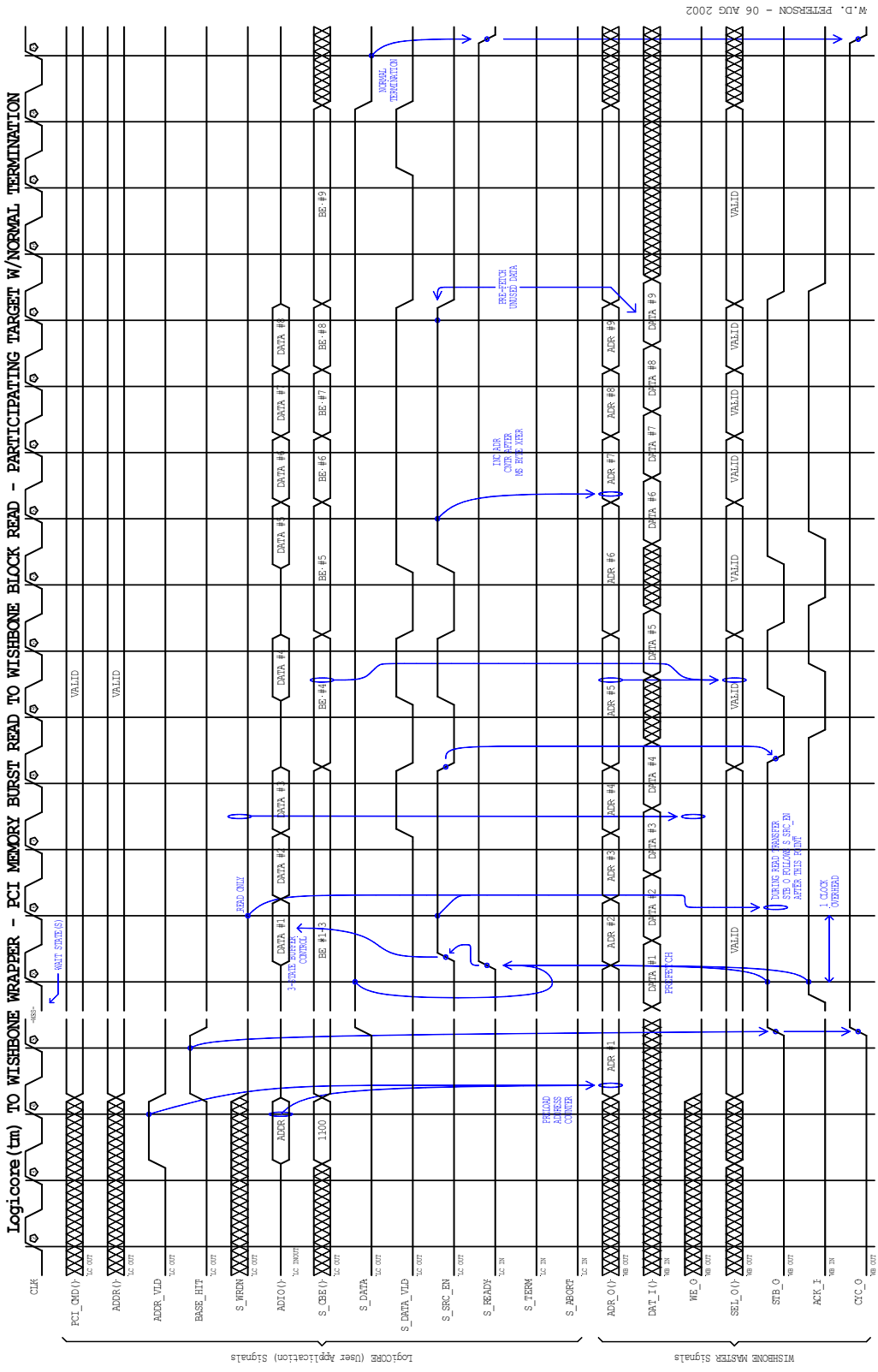


Figure 4-15. PCI memory burst read cycle to WISHBONE block read.

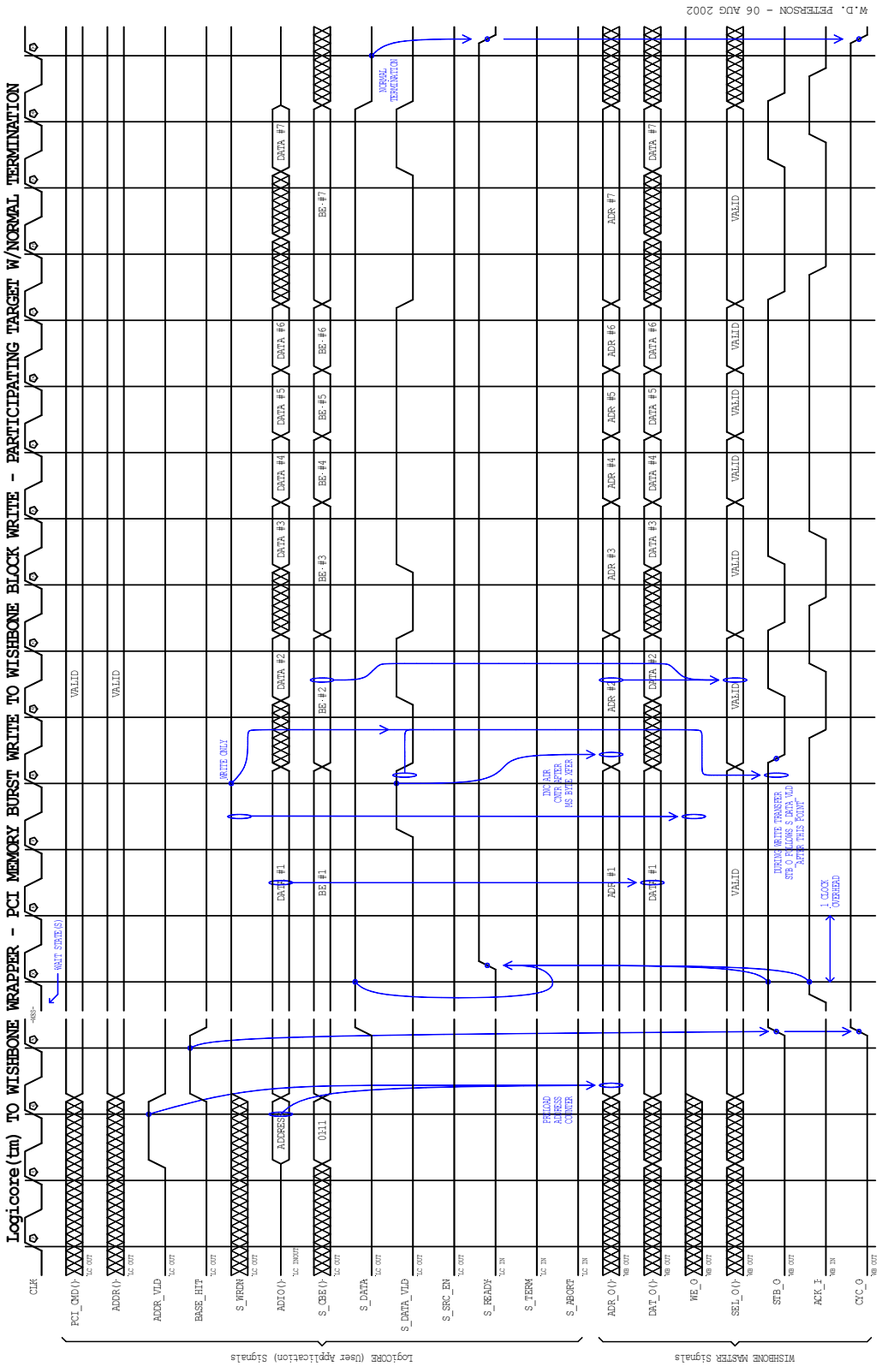


Figure 4-16. PCI memory burst write cycle to WISHBONE block write.

4.3.2 Operation of the WISHBONE Termination Signals

The PCIWRAP entity supports all three WISHBONE termination signals: [ACK_I], [RTY_I] and [ERR_I]. These are translated to control signals used by the Xilinx LogiCORE(tm) PCI core. The wrapper responds differently to these signals, depending upon when they are asserted during the PCI bus cycle.

Also note that the WISHBONE specification requires that only one termination signal be applied at any given time. Stated another way, [ACK_I] and [ERR_I] (or other combinations) cannot be asserted at the same time. Assertion of two or more acknowledge signals at any given time could result in an undefined operation.

4.3.3 PCI Normal Termination Using [ACK_I]

Figure 4-17 shows the timing diagram for a normal termination to a participating PCI TARGET cycle. There, the WISHBONE MASTER cycle starts in response to the assertion of BASE_HIT() by the Xilinx LogiCORE PCI (not shown). The PCIWRAPC wrapper then waits for the participating WISHBONE SLAVE to respond by asserting its [ACK_I] signal.

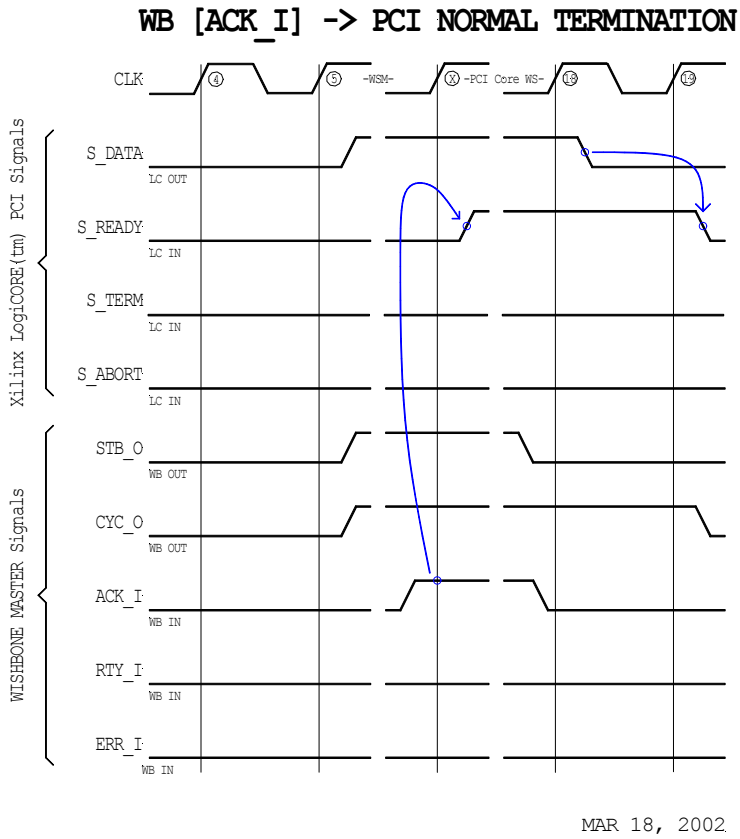


Figure 4-17. PCI normal termination.

In theory, the WISHBONE SLAVE can insert any number of wait states before responding to the cycle. However, the Xilinx documentation states that it must respond within 16 clock cycles[23]. This means that any arbitration on the WISHBONE SLAVE must be completed during this time.

It is also recommended that any WISHBONE SLAVEs be designed so that all arbitration is performed in response to the assertion of [CYC_O] by the MASTER (rather than [STB_O]). Once arbitration has been completed, the SLAVE should be made available to respond immediately to multiple assertions of [STB_O] by the MASTER. Stated another way, the SLAVE should only arbitrate at the beginning of a WISHBONE bus cycle. This allows immediate access to the SLAVE's resources throughout PCI burst transfers (if supported).

Once the WISHBONE MASTER interface has received the [ACK_I] signal, it asserts the [S_READY] signal to the PCI core. This signal remains asserted for the entire duration of the bus cycle, regardless of the state of the [ACK_I] signal. However, if the [RTY_I] or [ERR_I] signals are asserted, then the wrapper will generate either a PCI TARGET DISCONNECT W/O DATA or a TARGET ABORT.

During PCI burst transfers the [S_READY] signal remains asserted until the end of the PCI burst cycle. After the first cycle the wrapper does not respond to the [ACK_I] signal. This is because the PCI core requires that the WISHBONE SLAVE must supply or latch data during every clock cycle. Stated another way, the Xilinx PCI core does not support any handshaking mechanisms to throttle the speed of the transfer during this time.

4.3.4 PCI Target Retry Termination Using [RTY_I]

Figure 4-18 shows the timing diagram for a TARGET RETRY termination to a participating PCI TARGET cycle. There, the WISHBONE MASTER cycle starts in response to the assertion of BASE_HIT() (not shown) by the PCI core. The PCIWRAPC wrapper then waits for the participating WISHBONE SLAVE to respond. Normally, the WISHBONE SLAVE responds by asserting the [ACK_I] signal. However, if the SLAVE responds with [RTY_I], then the PCIWRAPC wrapper asserts [S_TERM] to the PCI core, which results in one of two behaviors:

- If the cycle is a burst cycle and the [ACK_I] had not been previously asserted, then the assertion of the [RTY_I] signal results in termination of the PCI bus cycle with a TARGET RETRY. The same reaction happens if the cycle is a non-burst cycle.

- If the [ACK_I] had been previously asserted during the current bus cycle, then the assertion of the [RTY_I] signal results in the termination of the PCI bus cycle with a TARGET DISCONNECT W/O DATA.

[23] The allowed delay is probably less than 16 clock cycles. The 16 clock cycle delay is evidently imposed by the PCI timer. However, there is probably some overhead added by the Xilinx LOGIcore(tm) PCI core.

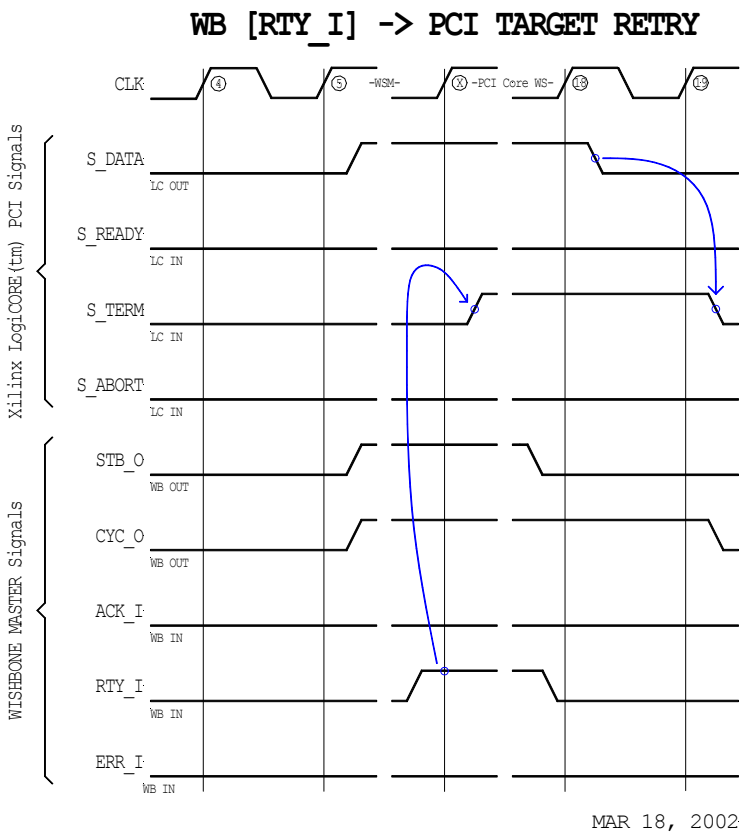


Figure 4-18. PCI target retry termination.

4.3.5 PCI Target Abort Termination Using [ERR_I]

Figure 4-19 shows the timing diagram for a TARGET ABORT termination. There, the WISHBONE MASTER cycle starts in response to the assertion of BASE_HIT() (not shown) by the PCI core. The PCIWRAPC wrapper then waits for the participating WISHBONE SLAVE to respond. Normally, the WISHBONE SLAVE responds by asserting the [ACK_I] signal. However, if the WISHBONE bus cycle is terminated by asserting the [ERR_I] signal, then the wrapper asserts [S_ABORT] to the PCI core. This results in a TARGET ABORT termination. The TARGET ABORT can be generated at any time during a single or burst transfer.

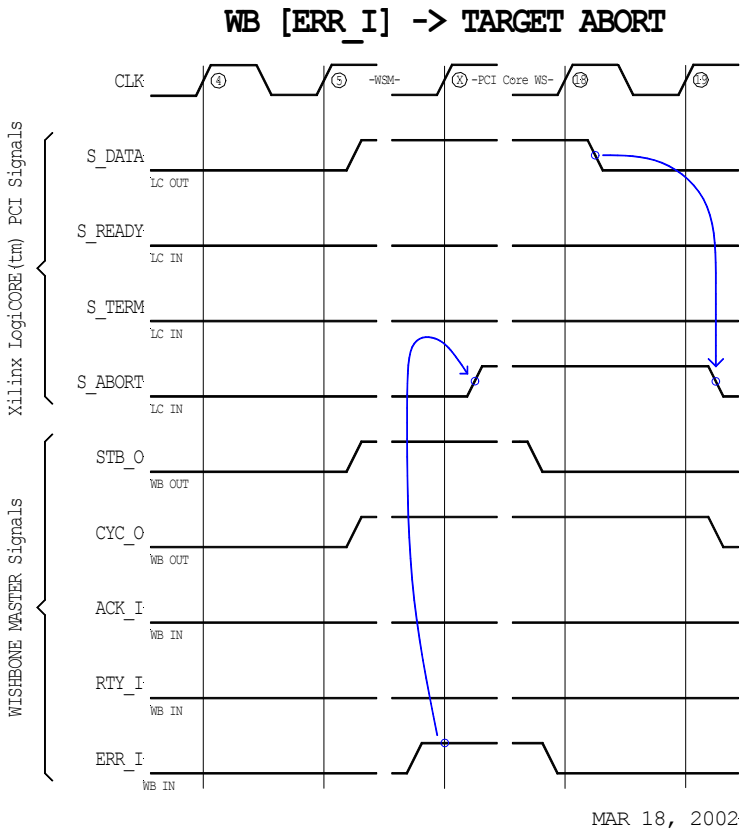


Figure 4-19. PCI target abort termination.

4.3.6 VHDL Entity Reference

The PCIWRAP entity is synthesized from a top level file called 'PCIWRAPC.VHD'. Each module of VHDL code is named with a seven character 'handle' that is related to its entity name. An additional (final) character is added to indicate it's use. A 'C' character indicates that it's a VHDL circuit file, a 'T' character indicates that it's a test bench file, and a 'V' character indicates that it's a test vector file (with a '.txt' file extension). The 'PCIWRAP' entity/architecture has the following names associated with it:

Entity (circuit) name:	PCIWRAPC
Architecture name:	PCIWRAPC1
Entity/architecture filename:	PCIWRAPC.VHD
Test bench filename:	PCIWRAPT.VHD
Text vector filename:	PCIWRAPV.TXT

4.3.7 Process: ADIO_THREESTATE

The ADIO_THREESTATE process generates a three-state output buffer on the 32-bit ADIO bus.

4.3.8 Process: COMMAND_REGISTER

The COMMAND_REGISTER process creates the command register. The command register stores the state of the PCI command that is indicated on 'S_CBE()' during the initial (address) phase of every bus cycle.

4.3.9 Process: COUNTER_INC_CONTROL

The COUNTER_INC_CONTROL generates the counter increment signal 'CINC'. The counter is incremented at different times, depending on if it's a read or a write cycle. The type of cycle is determined from the 'PCMD(3..0)' register, which stores the state of the PCI command (on [C/BE[3:0]#). The counter is incremented only during Memory Read Multiple ([C/BE[3:0]# = B"1100") and Memory Write ([C/BE[3:0]# = B"0111") cycles.

The counter is incremented after every cycle that accesses the high byte of a 32-bit DWORD transfer. The high byte is indicated when [S_CBE(3)] is low. That means that the address counter is incremented after every 32-bit transfer, after a high order 16-bit transfer and a high order 8-bit transfer.

4.3.10 Process: CYCLE_CONTROL

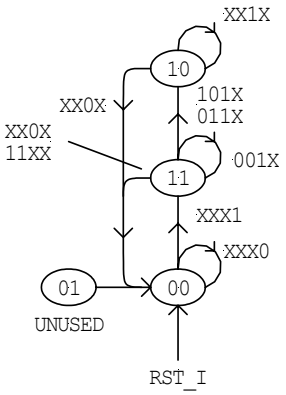
The CYCLE_CONTROL process generates the WISHBONE [STB_O] and [CYC_O] signals. It also generates the [S_READY], [S_TERM] and [S_ABORT] signals to the Xilinx Logi-CORE(tm) PCI interface. The process uses the state machines and equations shown in Figure 4-20.

CYCLE CONTROL STATE MACHINE:

```
STATES: CYC, FSTB
INPUTS: PWC, PRC, S_DATA, BASE_HIT
```

EQUATIONS:

```
PRC = /S_WRDN * S_SRC_EN;
PWC =  S_WRDN * S_DATA_VLD;

FSTB := /RST_I * /CYC * /FSTB                * BASE_HIT
      + /RST_I *  CYC *  FSTB * /PWC * /PRC * S_DATA;

CYC  := /RST_I * /CYC * /FSTB                * BASE_HIT
      + /RST_I *  CYC * /FSTB                * S_DATA
      + /RST_I *  CYC *  FSTB * /PWC         * S_DATA
      + /RST_I *  CYC                * /PRC * S_DATA;

CYC_O = CYC;

STB_O = /RST_I           *  FSTB
      + /RST_I * /S_WRDN * S_SRC_EN
      + /RST_I *  S_WRDN * S_DATA_VLD;
```

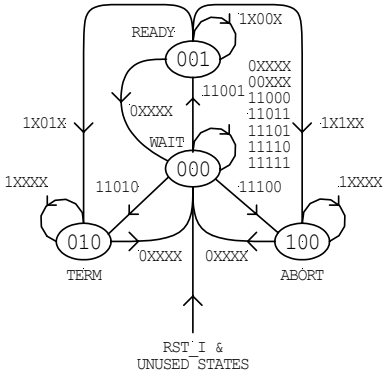


HANDSHAKING STATE MACHINE (PART OF 'CYCLE CONTROL'):

```
STATES: ABORT, TERM, READY
INPUTS: SDATA, FSTB, ERR_I, RTY_I, ACK_I
```

EQUATIONS:

```
S_READY <= READY;
S_TERM  <= TERM;
S_ABORT <= ABORT;
```

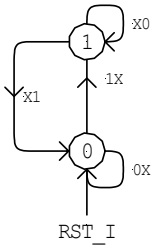


INITIAL CYCLE STATE MACHINE (PART OF 'CYCLE CONTROL'):

```
STATES: PCYC
INPUTS: BASE_HIT, AER
```

EQUATIONS:

```
AER = ACK_I + ERR_I + RTY_I;

PCYC := /RST_I * /PCYC * BASE_HIT
      + /RST_I *  PCYC * /AER;

ICYC  = PYC * AER;
```



XILINX LOGICORE PCI TO WISHBONE WRAPPER
VHDL ENTITY: PCIWRAP
06 AUG 2002

Figure 4-20. CYCLE CONTROL and HANDSHAKING state machines.

4.3.11 Process: LOWER_ADDRESS_COUNTER

The lower address counter generates the lower eight bits of the WISHBONE address bus. The counter is needed because the PCI core generates a base address during the first phase of a bus cycle. Subsequent PCI bus cycles do not generate any address information. The initial base address is preloaded into the counter. After that, each PCI cycle phase increments the counter.

The counter generates the lower eight bits of the local address bus ADR(9..2). The upper twenty-four bits are held by a latch. Local address bus ADR(33..2) generates the WISHBONE address bus ADR_O(33..2). The counter and latch implement the PCI to WISHBONE address translation. The PCI interface uses a 32-bit address bus with four byte enables. This effectively allows 34-bit byte addressability. The WISHBONE interface has an 8-bit granularity. For this reason the PCI address bits are shifted up by two bits, and the byte enables are translated into the WISHBONE SEL_O() bits.

The counter is designed in three sections, with the first two sections having a terminal count (TCNTX) bit. This reduces the number of 'and' terms in the equations of the higher counter bits.

The counter design has been used in other projects, and represents a reasonable compromise between speed and complexity.

Important signals and their uses are:

ADR(9..2): COUNTER OUTPUT (LOCAL ADDRESS BUS)
ADIO(31..0): COUNTER DATA INPUT (USED FOR PRELOAD)
CINC: COUNTER INCREMENT
ADDR_VLD: COUNTER PRELOAD
TCNT2: TERMINAL COUNT FROM BIT 2
TCNT5: TERMINAL COUNT FROM BIT 5

4.3.12 Process: SELECT_CONTROL

The SELECT_CONTROL process generates the local [SEL()] and WISHBONE [SEL_O()] signals.

4.3.13 Process: UPPER_ADDRESS_REGISTER

The UPPER_ADDRESS_REGISTER process creates a 24-bit register, and captures the state of the upper twenty-four address lines coming from ADIO(31..08). These remain static throughout the entire PCI bus cycle, regardless of the type of PCI bus cycle. The lower eight bits are captured by the address counter.

Important signals include:

ADR(33..10): REGISTER OUTPUT (LOCAL ADDRESS BUS)
ADIO(31..8): REGISTER DATA INPUT
ADDR_VLD: REGISTER LOAD

4.3.14 Process: WISHBONE_SYSCON

The WISHBONE_SYSCON process generates the WISHBONE SYSCON signals [CLK_O] and [RST_O]. The [CLK_I] and [RST_I] signals should be tied to the [CLK_O] and [RST_O] signals outside of this entity.

[RST_O] is a synchronized version of the asynchronous [RST] signal generated by the Xilinx LogiCORE.

4.3.15 Process: DI_REG

The DI_REG process is a 'data in' register. It's only purpose is to overcome some timing problems associated with the three-state buffers used with the Xilinx LogiCORE PCI. If this register is removed the router attempts to resolve the asynchronous data path from the data in port [DAT_I()], through the three-state buffers, back to the data out port [DAT_O()] and then on to other system components. This timing path is broken when the registers are added.

4.4 SEMABUD Entity

The SEMABUD entity is a shared memory. It is very similar to the SEMABUF entity, except that it implements the memory buffer with Xilinx Spartan 2 distributed RAM instead of Block-Select+ RAM. Furthermore, this entity responds in one clock cycle (the SEMABUF entity responds in two cycles). For more information please see the 'SEMABUF' entity described elsewhere in this manual.

4.5 SEMABUF Entity

The SEMABUF entity is a shared memory buffer. As shown in the block diagram of Figure 4-21, the entity has two WISHBONE SLAVE interfaces called 'PORT A' and 'PORT B'. These connect to two, 256 x 16-bit synchronous memories, thereby forming 32-bit data ports. The memories are created from two Xilinx Spartan 2 BlockSelect+ memories, with details shown in Figure 4-22.

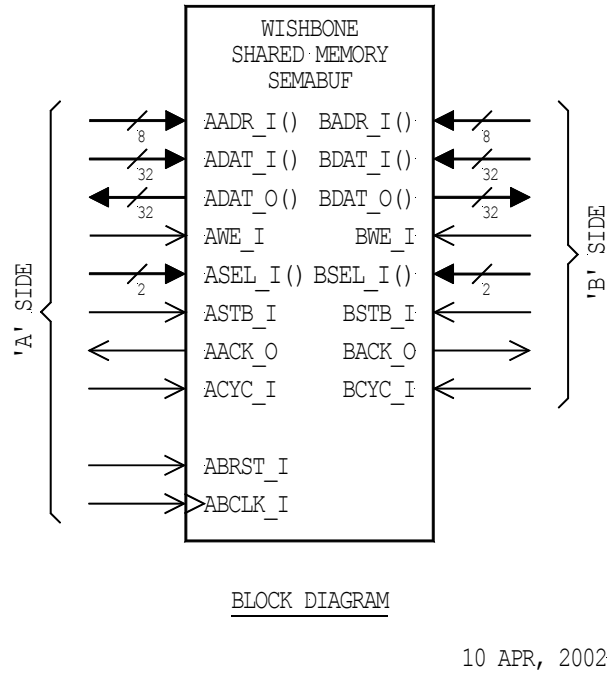


Figure 4-21. SEMABUF block diagram.

The buffer operates as an independent shared memory. This means that both sides of the memory supports full, simultaneous, read/write privileges into each buffer. During simultaneous transfers one side of the buffer or the other is held off until the other port has finished its operation. In this case the [AACK_O] or [BACK_O] signal holds off memory accesses. This activity is controlled by an internal arbiter circuit.

Table 4-4 gives the specifications for the WISHBONE ports.

An internal arbiter determines whether PORT A or PORT B gains access to the shared memory buffer. The arbiter is composed of a two bit state machine, with a state diagram shown in Figure 4-23, with the related timing diagram shown in Figure 4-24.

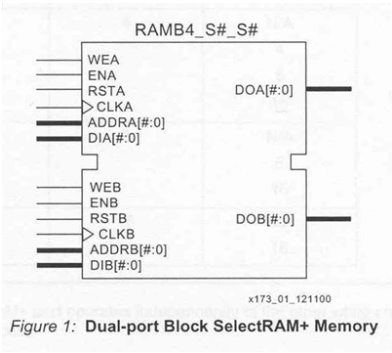


Figure 1: **Dual-port Block SelectRAM+ Memory**

XILINX SelectRAM+ CONNECTION DIAGRAM

WISHBONE ACKNOWLEDGE SIGNALS

```
AACK_O        = ARDY * ASTB_I;

BACK_O        = BGNT * BSTB_I;
```

XILINX SPARTAN 2 DUAL-PORT SelectRAM+ EQUATIONS

```
PORTA - BANK #0
WEA           = AWE_I;
ENA           = ARDY * ASTB_I * ASEL_I(0);
RSTA          = GND;
CLKA          = ABCLK_I;
ADDRA(7..0)   = ADDR_I(8..1);
DINA(15..0)   = ADAT_I(15..0);
ADAT_O(15..0) = DOA(15..0);

PORTB - BANK #0
WEB           = BWE_I;
ENB           = BGNT * BSTB_I * BSEL_I(0);
RSTB          = GND;
CLKB          = ABCLK_I;
ADDRB(7..0)   = BDDR_I(8..1);
DINB(15..0)   = BDAT_I(15..0);
BDAT_O(15..0) = DOB(15..0);

PORTA - BANK #1
WEA           = AWE_I;
ENA           = ARDY * ASTB_I * ASEL_I(1);
RSTA          = GND;
CLKA          = ABCLK_I;
ADDRA(7..0)   = ADDR_I(8..1);
DINA(15..0)   = ADAT_I(31..16);
ADAT_O(15..0) = DOA(31..16);

PORTB - BANK #1
WEB           = BWE_I;
ENB           = BGNT * BSTB_I * BSEL_I(1);
RSTB          = GND;
CLKB          = ABCLK_I;
ADDRB(7..0)   = BDDR_I(8..1);
DINB(15..0)   = BDAT_I(31..16);
BDAT_O(15..0) = DOB(31..16);

                                                    29 SEP 2002
```

Figure 4-22. Implementation details for the Xilinx BlockSelect+ RAM.

Table 4-4. WISHBONE DATASHEET for the SEMABUF Entity	
General Description	256 x 32-bit shared memory buffer with two WISHBONE SLAVE interfaces. This datasheet applies to both interfaces.
WISHBONE Revision Level	B.2
Supported WISHBONE Cycles	MASTER: READ/WRITE MASTER: BLOCK READ/WRITE
Data port, size	32-bit
Data port, granularity	16-bit
Data port, maximum operand size	32-bit
Data transfer ordering	BIG ENDIAN or LITTLE ENDIAN
Data transfer sequencing	None
Signal Description	All WISHBONE signal names are identical to those defined in the specification, except that they have an 'A' or 'B' at the front. The 'A' and 'B' refer to PORT A and PORTB respectively.
Terminating Signals	The WISHBONE interface on both ports support only the [ACK_O] terminating signal.

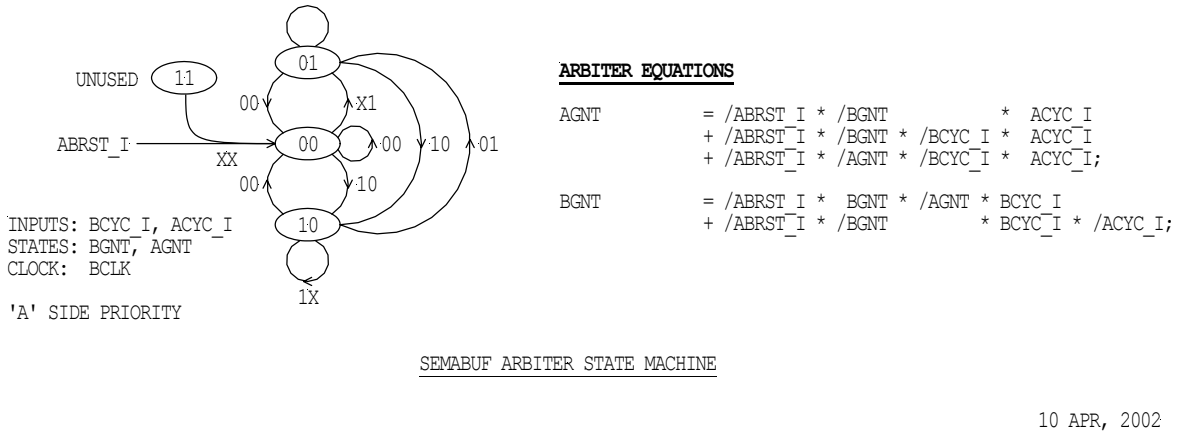


Figure 4-23. SEMABUF arbiter state machine.

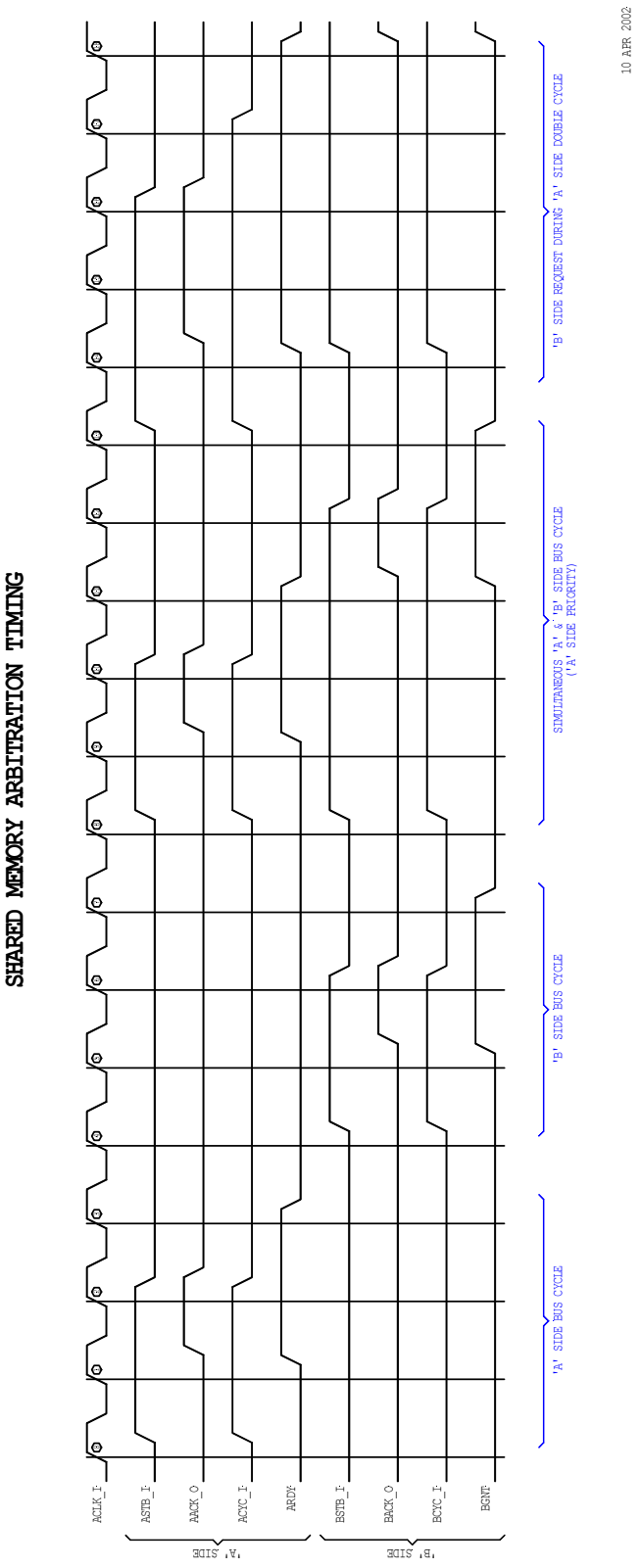


Figure 4-24. SEMABUF shared memory arbiter timing.

4.6 SEMAREG Entity

The SEMAREG entity provides a one-bit semaphore to two WISHBONE SLAVE interfaces. The interfaces are called ‘A’ and ‘B’. The semaphore is intended to be used for shared memory buffers where one interface or the other must gain access to the memory. The semaphore does not lock the memory buffer (which is located elsewhere) but just provides a mechanism for system software to determine if the particular memory buffer is being used. Figure 4-25 shows a block diagram of the entity.

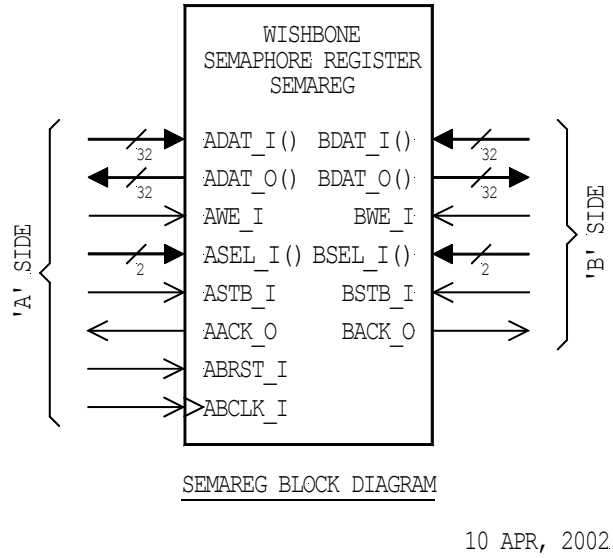


Figure 4-25. SEMAREG block diagram.

The semaphore is accessed by reading the bit. If the bit is returned as ‘0’, then the WISHBONE MASTER (usually a processor) accessing the device is granted the semaphore. If the bit is returned as ‘1’, the buffer is busy. If a processor obtains the buffer by reading ‘0’ (becomes the owner), and the bit is sampled for a second time, then the bit is returned as ‘1’ on the second access.

If both ports attempt to access the semaphore t the same time (i.e. during the same clock cycle), then port ‘A’ access has priority.

The buffer is released by writing a ‘0’ to the semaphore. Writing a ‘1’ to the bit does not have any effect. The semaphore may be released from either port.

Table 4-5 shows the WISHBONE DATASHEET for the SEMAREG entity.

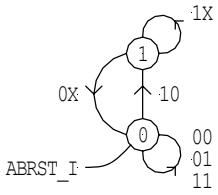
Table 4-5. WISHBONE DATASHEET for the SEMAREG Entity	
General Description	32-bit semaphore register with WISHBONE SLAVE ports (‘A’ and ‘B’). This datasheet applies to both interfaces.
WISHBONE Revision Level	B.2
Supported WISHBONE Cycles	MASTER: READ/WRITE
Data port, size	32-bit
Data port, granularity	16-bit
Data port, maximum operand size	32-bit
Data transfer ordering	BIG ENDIAN or LITTLE ENDIAN
Data transfer sequencing	None
Signal Description	All WISHBONE signal names are identical to those defined in the specification, except that they have an ‘A’ or ‘B’ at the front. The ‘A’ and ‘B’ refer to PORT A and PORTB respectively.
Terminating Signals	The WISHBONE interface on both ports support only the [ACK_O] terminating signal.

The circuit arbiter determines whether PORT A or PORT B gains access to the shared memory buffer. The arbiter is composed of a two bit state machine, with a state diagram shown in Figure 4-26. The timing diagram is shown in Figure 4-27.

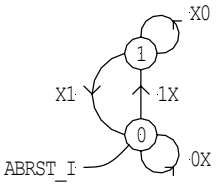
```
STATES: AGNT
INPUTS: AREQ, GNT
CLOCKS: BCLK

'A' SIDE PRIORITY

EQUATIONS:

AGNT := /ABRST_I * AREQ * /GNT
      + /ABRST_I * AREQ * AGNT;
```

```
STATES: GNT
INPUTS: REQ, REL
CLOCKS: BCLK

EQUATIONS:

GNT := /ABRST_I * /GNT *  REQ
     + /ABRST_I *  GNT * /REL;

REQ = AREQ + BREQ;
REL = AREL + BREL;
```

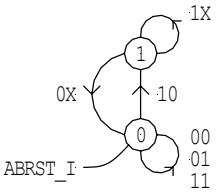
```
STATES: BGNT
INPUTS: MBREQ, GNT
CLOCKS: BCLK

'A' SIDE PRIORITY

EQUATIONS:

BGNT := /ABRST_I * MBREQ * /GNT
      + /ABRST_I * MBREQ * BGNT;

MBREQ = /AREQ *  BREQ;
```



AGNT STATE MACHINE



GNT STATE MACHINE



BGNT STATE MACHINE

```
OTHER EQUATIONS:

AREL    = ASTB_I *  AWE_I * ASEL_I(0) * /ADAT_I(0);
AREQ    = ASTB_I * /AWE_I * ASEL_I(0);
SAACK  := ASTB_I;
AACK_O  = SAACK * ASTB_I;

BREL    = BSTB_I *  BWE_I * BSEL_I(0) * /BDAT_I(0);
BREQ    = BSTB_I * /BWE_I * BSEL_I(0;
SBACK  := BSTB_I;
BACK_O := SBACK * BSTB_I;
```

30 MAY, 2002

Figure 4-26. SEMAREG state machines.

The semaphore is formed from a series of three, one-bit state machines called ‘GNT’, ‘AGNT’ and ‘BGNT’. The ‘GNT’ state machine indicates if the semaphore has been granted or not. Semaphore accesses from either side of the entity will cause the semaphore to be set. The other two state machines determine if one side of the entity or the other is granted the semaphore.

For example, if the semaphore is not granted, and a semaphore request occurs from the ‘A’ side, then the ‘GNT’ state machine bit will transition from ‘0’ to ‘1’. At the same time the ‘AGNT’ state machine will also transition from ‘0’ to ‘1’. This grants the semaphore to the ‘A’ side. Subsequent accesses from the ‘A’ side will not grant the semaphore, as it is designed so that the semaphore is only granted during the first read access.

Once the semaphore has been granted, neither side can obtain it until the semaphore is released. The release can come from either port of the interface.

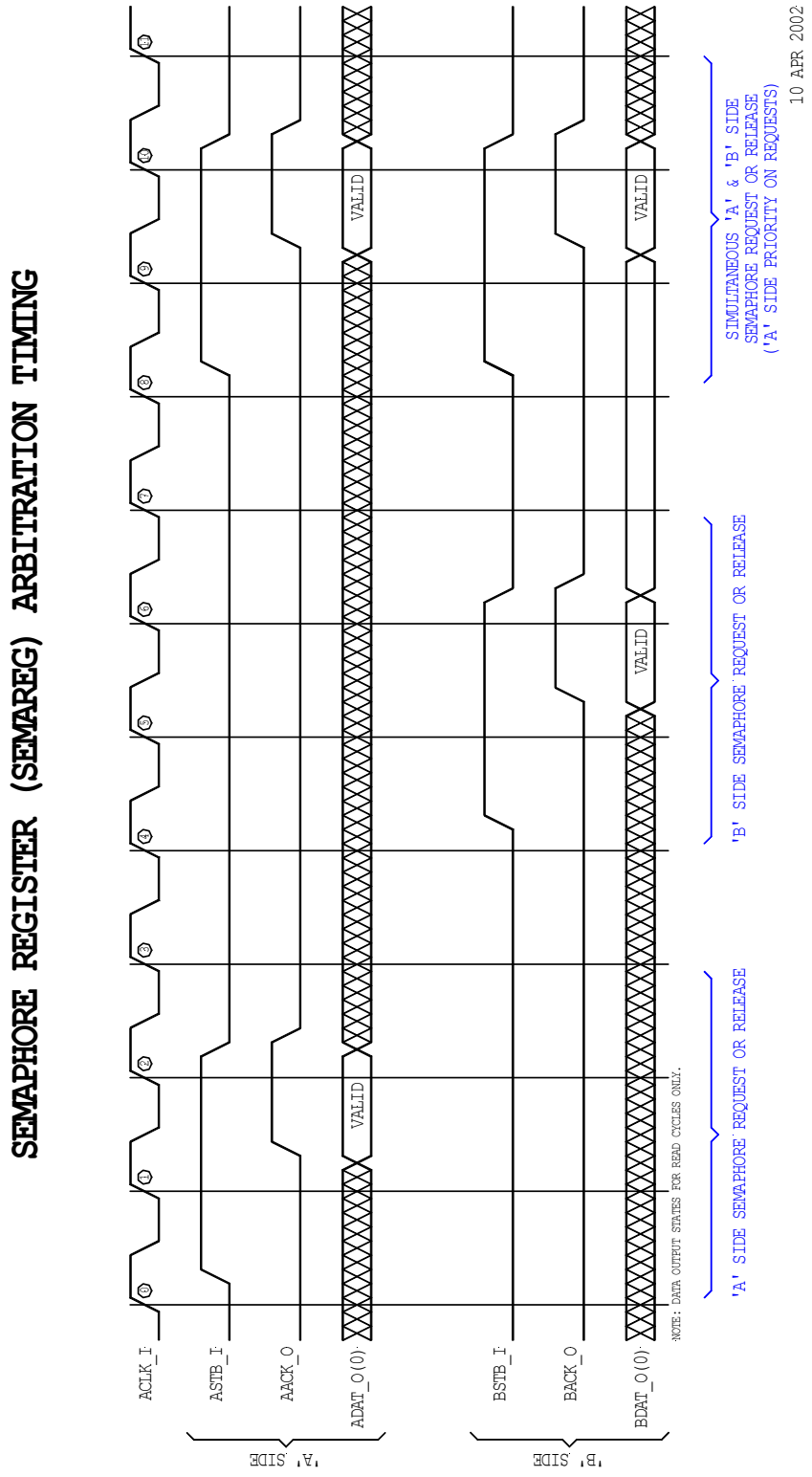


Figure 4-27. Timing diagram for the SEMAREG entity.

4.7 VMEcore™ Entity

The VMEcore™ entity is a A24:D32:D16:D08(O) VMEbus SLAVE to WISHBONE MASTER bridge. As shown in Figure 4-28, the bridge allows connection to a local SoC interconnection. The entire interface is synthesized as the 'VMECOREc' VHDL entity. Table 4-6 shows the characteristics of the WISHBONE interface.

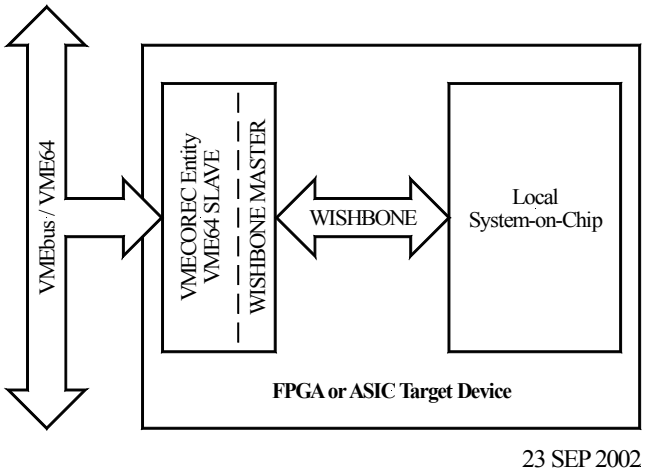


Figure 4-28. Block diagram of the VMEcore™ interface.

Features of the VMEcore(tm) interface include:

- A24:D32:D16:D08(EO) VMEbus slave interface.
- Compact design minimizes gate count and per-unit costs.
- Enforces VMEbus interface rules and timing.
- 32-bit WISHBONE MASTER (backend) interface.
- Allows fabrication of a complete VMEbus SLAVE on a single chip (with peripherals).
- Provided as VHDL source code.

Table 4-6. WISHBONE DATASHEET for the VMEcore(tm).	
General Description	Backend interface for a VMEbus SLAVE IP core. Operates as a WISHBONE SoC interface.
WISHBONE Revision Level	B.2
Supported WISHBONE Cycles	MASTER: READ/WRITE MASTER: RMW
Data port, size	32-bit
Data port, granularity	8-bit
Data port, maximum operand size	32-bit
Data transfer ordering	BIG ENDIAN
Data transfer sequencing	UNDEFINED
Signal Description	All WISHBONE signal names are identical to those defined in the specification. Signal [A24SGL_O] is a tag with TAG TYPE: [TGA_O]. Refer to the signal descriptions for more details.
Optional [ERR_I] support	WISHBONE cycles terminated with [ERR_I] terminate VMEbus cycles with BERR*.

4.7.1 VMEbus SLAVE Interface Signals

All VMEbus slave signal names have the '_I' or '_O' characters attached to them. These indicate if the signals are an input (to the core) or an output (from the core). For example, [ACK_I] is an input and [ACK_O] is an output. This convention is used to clearly identify the direction of each signal.

Signal arrays are identified by a name followed by a set of parenthesis. For example, [DAT_I()] is a signal array. Array limits may also be shown within the parenthesis. In this case the first number of the array limit indicates the most significant bit, and the second number indicates the least significant bit. For example, [DAT_I(31..0)] is a signal array with upper array boundary number thirty-one (the most significant bit), and lower array boundary number zero (the least significant bit). The array size on any particular core may vary. In many cases the array boundaries are omitted if they are irrelevant to the context of the description.

When used as part of a sentence, signal names are enclosed in brackets '[ ]'. This helps to discriminate signal names from the words in the sentence.

The VMEbus interface signals can be directly connected to the target device, or routed through buffer chips. Buffer chips are generally used on the VMEbus interface because FPGA and ASIC target devices usually do not have compatible inputs and outputs. VMEbus is based on TTL interface standards, and regulate[24]:

- Noise margins
- Load current (inputs)
- Output short circuit current
- Input clamp voltage
- Capacitive loading
- Output current
- Backplane impedance

VA_I()
VMEbus address input signal array [VA_I()]. Connect these signals to the corresponding VMEbus address lines A01 – A31 (either directly or through a buffer).

VAM_I()
VMEbus address modifier input signal array [VAM_I()]. Connect these signals to the corresponding VMEbus address modifier lines AM0 – AM5 (either directly or through a buffer).

[24] For more information, the reader is directed to the ANSI/VITA 1-1994 standard.

VD_I()
VMEbus data input signal array [VD_I()]. Connect these signals to the corresponding VMEbus data lines D00 – D31 (either directly or through a buffer).

VD_O()
VMEbus data signal output array [VD_O()]. Connect these signals (usually through a buffer) to the corresponding VMEbus data lines D00 – D31. They are generally connected through a buffer to provide sufficient drive current to the VMEbus backplane.

VNAS_I
VMEbus address strobe input signal [VNAS_I]. Connect this signal to the VMEbus [AS*] signal (either directly or through a buffer). Also note that both [VNAS_I] and [AS*] are active low signals.

VNBERR_O
VMEbus bus error output signal [VNBERR_O]. Connect this signal to the VMEbus [BERR*] signal. It is generally connected through a buffer to provide sufficient current drive to the VMEbus backplane. Also note that both [BERR*] and [VNBERR_O] are active low signals.

VNDS0_I
VMEbus data strobe input signal [VNDS0_I]. Connect this signal to the VMEbus [DS0*] signal (either directly or through a buffer). Also note that both [DS0*] and [VNDS0_I] are active low signals.

VNDS1_I
VMEbus data strobe input signal [VNDS1_I]. Connect this signal to the VMEbus [DS1*] signal (either directly or through a buffer). Also note that both [DS1*] and [VNDS1_I] are active low signals.

VNDTACK_O
VMEbus data transfer acknowledge output signal [VNDTACK_O]. Connect this signal to the VMEbus [DTACK*] signal. It is generally connected through a buffer to provide sufficient current drive to the VMEbus backplane. Also note that both [DTACK*] and [VNDTACK_O] are active low signals.

VNIACK_I
VMEbus interrupt acknowledge input signal [VNIACK_I]. Connect this signal to the VMEbus [IACK*] signal (either directly or through a buffer). Also note that both [IACK*] and [VNIACK_I] are active low signals.

VNLWORD_I
VMEbus long word input signal [VNLWORD_I]. Connect this signal to the VMEbus [LWORD*] signal (either directly or through a buffer). Also note that both [LWORD*] and [VNLWORD_I] are active low signals.

VNSYSRESET_I
VMEbus system reset input signal [VNSYSRESET_I]. Connect this signal to the VMEbus [SYSRESET*] signal (either directly or through a buffer). Also note that both [SYSRESET*] and [VNSYSRESET_I] are active low signals.

VNWRITE_I
VMEbus write input signal [VNWRITE_I]. Connect this signal to the VMEbus [WRITE*] signal (either directly or through a buffer). Also note that both [WRITE*] and [VNWRITE_I] are active low signals.

VSAC24_I()
A24 VMEbus SLAVE address compare input signal array [VSAC24_I()]. This array determines when the SLAVE interface is selected by a VMEbus cycle. It is used by the local address comparator to decode the destination address of a bus cycle. They may be connected to a dip-switch or latch on the board which holds the base address of the SLAVE.

VSAE24_I
The A24 VMEbus SLAVE address enable input signal [VSAE24_I] enables the SLAVE interface. In most cases it should be permanently asserted (i.e. tied high). However, in some cases this signal is useful if the interface needs to be disabled.

VTST_I
The simulation test input signal [VTST_I] forces all self-starting counters and other devices in the VMEbus interface to an initial state. It is used for test simulation purposes, and should be negated (i.e. tied low) during normal core operation.

4.7.2 WISHBONE MASTER Interface Signals

A24SGL_O
The [A24SGL_O] signal indicates that a valid A24 VMEbus cycle is in progress, and that the core is participating in the cycle. It conforms to WISHBONE TAG TYPE: [TGA_O]. In many cases this signal is superfluous and can be ignored.

CLK_I
The clock input [CLK_I] coordinates all activities for the internal logic within the WISHBONE interconnect. All WISHBONE output signals are registered at the rising edge of [CLK_I]. All WISHBONE input signals are stable before the rising edge of [CLK_I].

DAT_I()
The data input array [DAT_I()] is used to pass binary data. The array boundaries are determined by the port size, with a maximum port size of 64-bits (e.g. [DAT_I(63..0)]). Also see the [DAT_O()] and [SEL_O()] signal descriptions.

DAT_O()
The data output array [DAT_O()] is used to pass binary data. The array boundaries are determined by the port size, with a maximum port size of 64-bits (e.g. [DAT_I(63..0)]). Also see the [DAT_I()] and [SEL_O()] signal descriptions.

RST_I
The reset input [RST_I] forces the WISHBONE interface to restart. Furthermore, all internal self-starting state machines will be forced into an initial state. This signal only resets the WISHBONE interface. It is not required to reset other parts of an IP core (although it may be used that way).

TGD_I()
Data tag type [TGD_I()] is used on MASTER and SLAVE interfaces. It contains information that is associated with a data lines [DAT_I()], and is qualified by signal [STB_I]. For example, parity protection, error correction and time stamp information can be attached to the data bus. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is pre-defined by this specification. The name and operation of a data tag must be defined in the WISHBONE DATASHEET.

TGD_O()
Data tag type [TGD_O()] is used on MASTER and SLAVE interfaces. It contains information that is associated with a data lines [DAT_O()], and is qualified by signal [STB_O]. For example, parity protection, error correction and time stamp information can be attached to the data bus. These tag bits simplify the task of defining new signals because their timing (in relation to every

bus cycle) is pre-defined by this specification. The name and operation of a data tag must be defined in the WISHBONE DATASHEET.

ACK_I
The acknowledge input [ACK_I], when asserted, indicates the termination of a normal bus cycle. Also see the [ERR_I] and [RTY_I] signal descriptions.

ADR_O()
The address output array [ADR_O()] is used to pass a binary address. The maximum size of the array is specified as [ADR_O(63..0)]. However, the higher array boundary is specific to the address width of the core, and the lower array boundary is determined by the data port size (e.g. the maximum array size on a 32-bit data port is [ADR_O(63..2)]. In some cases (such as FIFO interfaces) the array may not be present on the interface.

CYC_O
The cycle output [CYC_O], when asserted, indicates that a valid bus cycle is in progress. The signal is asserted for the duration of all bus cycles. For example, during a BLOCK transfer cycle there can be multiple data transfers. The [CYC_O] signal is asserted during the first data transfer, and remains asserted until the last data transfer. The [CYC_O] signal is useful for interfaces with multi-port interfaces (such as dual port memories). In these cases, the [CYC_O] signal requests use of a common bus from an arbiter. Once the arbiter grants the bus to the MASTER, it is held until [CYC_O] is negated.

ERR_I
The error input [ERR_I] indicates an abnormal cycle termination. The source of the error, and the response generated by the MASTER is defined by the IP core supplier. Also see the [ACK_I] and [RTY_I] signal descriptions.

RTY_I
The retry input [RTY_I] indicates that the interface is not ready to accept or send data, and that the cycle should be retried. When and how the cycle is retried is defined by the IP core supplier. Also see the [ERR_I] and [RTY_I] signal descriptions.

SEL_O()
The select output array [SEL_O()] indicates where valid data is expected on the [DAT_I()] signal array during READ cycles, and where it is placed on the [DAT_O()] signal array during WRITE cycles. The array boundaries are determined by the granularity of a port. For example, if 8-bit granularity is used on a 64-bit port, then there would be an array of eight select signals with boundaries of [SEL_O(7..0)]. Each individual select signal correlates to one of eight active bytes on the 64-bit data port. For more information about [SEL_O()], please refer to the data organiza-

tion section in Chapter 3 of this specification. Also see the [DAT_I()], [DAT_O()] and [STB_O] signal descriptions.

STB_O
The strobe output [STB_O] indicates a valid data transfer cycle. It is used to qualify various other signals on the interface such as [SEL_O()]. The SLAVE asserts either the [ACK_I], [ERR_I] or [RTY_I] signals in response to every assertion of the [STB_O] signal.

TGA_O()
Address tag type [TGA_O()] contains information associated with address lines [ADR_O()], and is qualified by signal [STB_O]. For example, address size (24-bit, 32-bit etc.) and memory management (protected vs. unprotected) information can be attached to an address. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is defined by this specification. The name and operation of an address tag must be defined in the WISHBONE DATASHEET.

TGC_O()
Cycle tag type [TGC_O()] contains information associated with bus cycles, and is qualified by signal [CYC_O]. For example, data transfer, interrupt acknowledge and cache control cycles can be uniquely identified with the cycle tag. They can also be used to discriminate between WISHBONE SINGLE, BLOCK and RMW cycles. These tag bits simplify the task of defining new signals because their timing (in relation to every bus cycle) is defined by this specification. The name and operation of a cycle tag must be defined in the WISHBONE DATASHEET.

WE_O
The write enable output [WE_O] indicates whether the current local bus cycle is a READ or WRITE cycle. The signal is negated during READ cycles, and is asserted during WRITE cycles.

4.7.3 VHDL Synthesis and Test

The VMEcore is organized as a series of VHDL entities. These are synthesized together from a top level entity known as 'VMECOREC.VHD'. Figure 4-29 shows the hierarchy and Figure 4-30 shows a functional diagram for the core.

Each module of VHDL code is named with a seven character 'handle' that is related to its entity name. An additional (final) character is added to indicate it's use. The 'C' character indicates that it's a VHDL circuit file (i.e. entity/architecture pair, usually contained in a file), the 'T' character indicates that it's a test bench file, and a 'V' character indicates that it's a test vector file. For example, the top level entity has the following names associated with it:

Entity (circuit) name:	VMECOREC
Architecture name:	VMECOREC1
Entity/architecture filename:	VMECOREC.VHD
Test bench filename:	VMECORET.VHD
Text vector filename:	VMECOREV.TXT

Only the top level module in the hierarchy includes a VHDL test bench. It is assumed that all of the files are simulated and synthesized as a group.

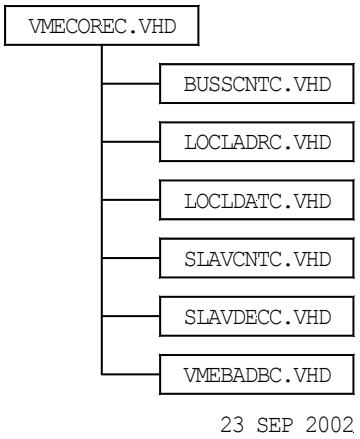


Figure 4-29. VMEcore entities.

The entire set of VMEcore™ entities is created with a parametric core generator called the VMEbus Interface Writer™. The tool itself is a trade secret of Silicore Corporation, and is not available under any license. However, the output files provided as VMEcore entities, test benches and test vector files are fully readable with standard editors. They may be easily and fully modified and maintained without the parametric core generator.

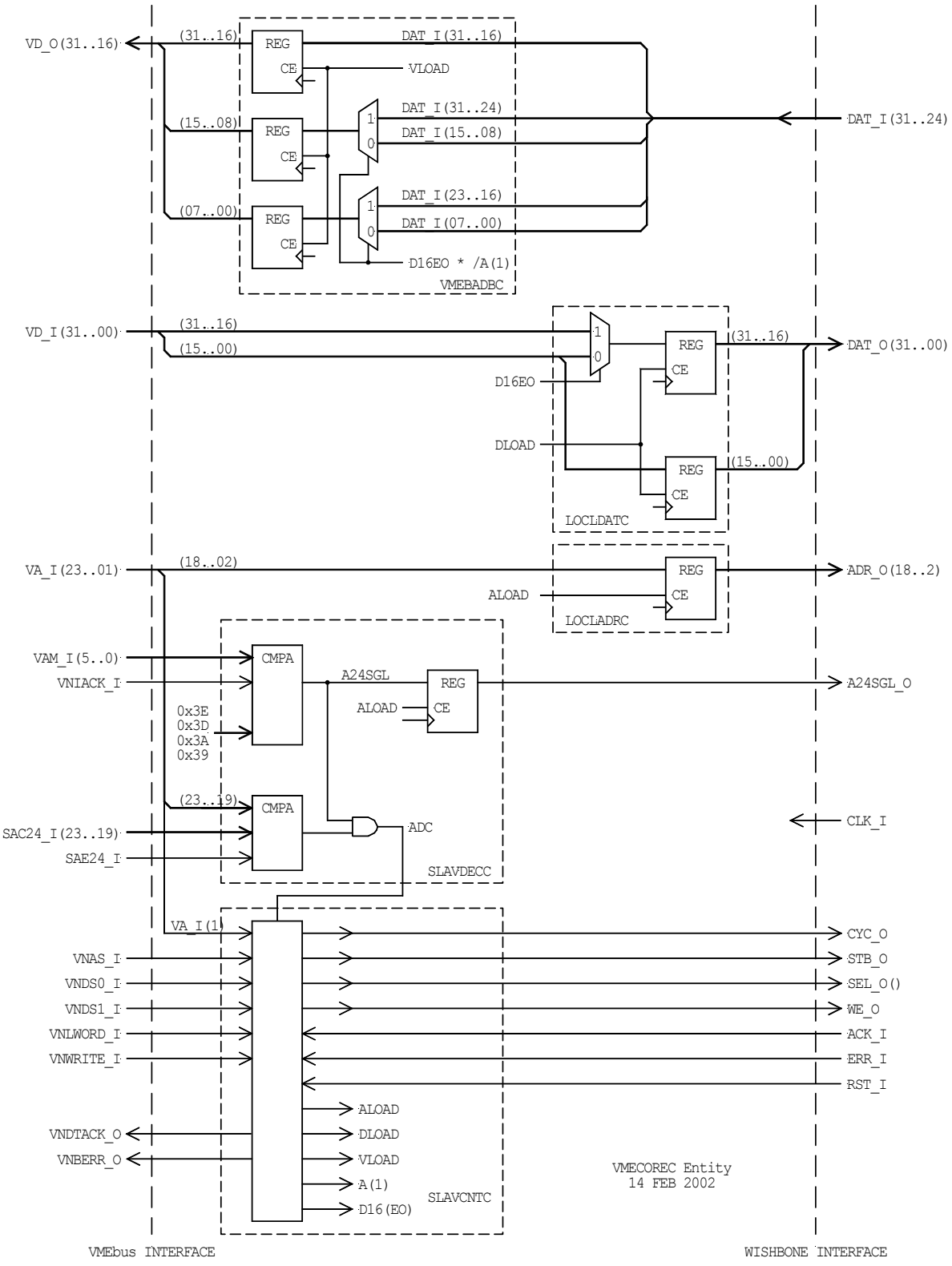


Figure 4-30. Functional diagram of the VMECOREC entity.

4.7.4 Bus Interface Timing

Before routing the VMEcore(tm) interface, correct timing specifications must be entered into the design. Furthermore, these timing specifications must take into account not only the requirements of the VMEcore design, but also of any external driver or receiver chips. These external chips are needed because FPGA and ASIC target devices aren't compatible with the electrical characteristics of the VMEbus backplane.

The VMEcore logical design has only three simple timing constraints:

- Input-to-clock setup time: 1 [VCLK]
- Flop-to-flop transition: 1 [VCLK]
- Clock-to-output delay time: 1 [VCLK]

The *input-to-clock setup time* can be analyzed with Figures 4-31(a) and 4-32(a). This timing parameter includes the time it takes for a VMEbus backplane signal to propagate through an input buffer, through the input pin of the target device and then to set up at the input of a flip-flop.

For example, consider a VMEbus input signal that traverses through an input buffer that has a timing delay [Tplh] of 7.0 ns. Furthermore, assume that [VCLK] operates at a clock speed of 50.000 MHz (or a period of 20.0 ns). This means that the target device must be routed with a worse case input-to-clock delay of:

$$Tpdlhin(max) = Tvclk(min) - Tphl$$

$$Tpdlhin(max) = 20.0\ ns - 7.0\ ns = 13.0\ ns$$

Although these numbers are given for the 'low-to-high' transition delay, a similar case exists for the 'high-to-low' transition delay. The method by which this time specification is entered into the software development tools depends upon the target technology and router used. For example, the timespec for the VMEbus data strobe [N_VDS0] input would be entered into a Xilinx FPGA router thusly:

```
NET "N_VDS0" OFFSET = IN 13 ns BEFORE "VCLK";
```

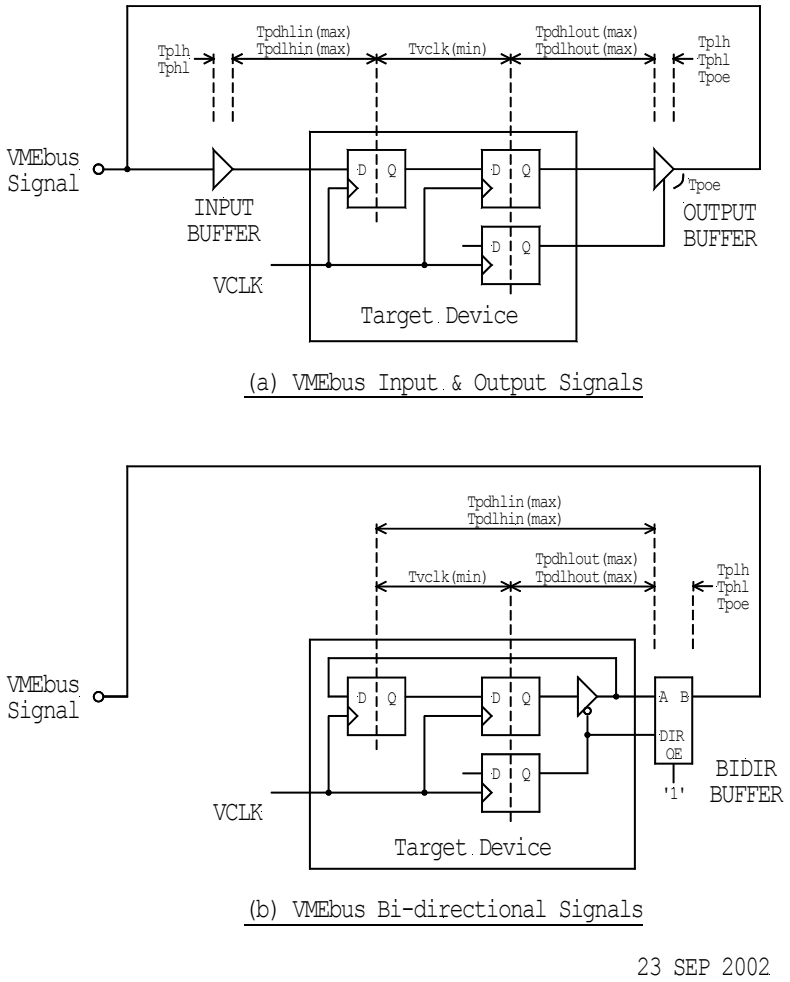
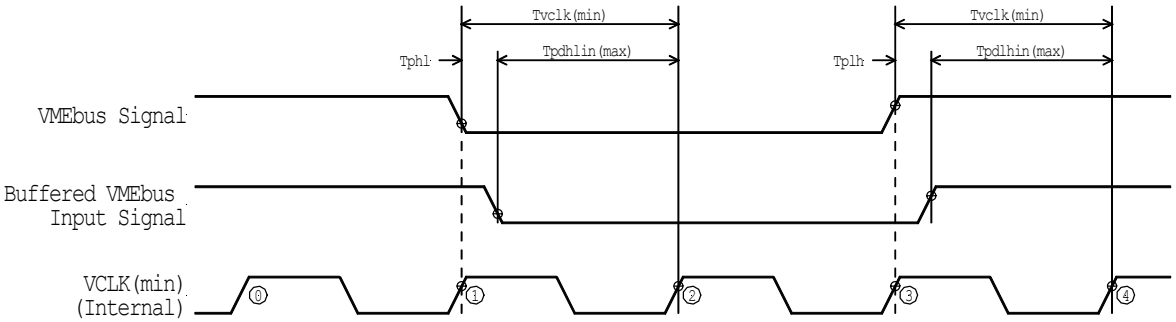
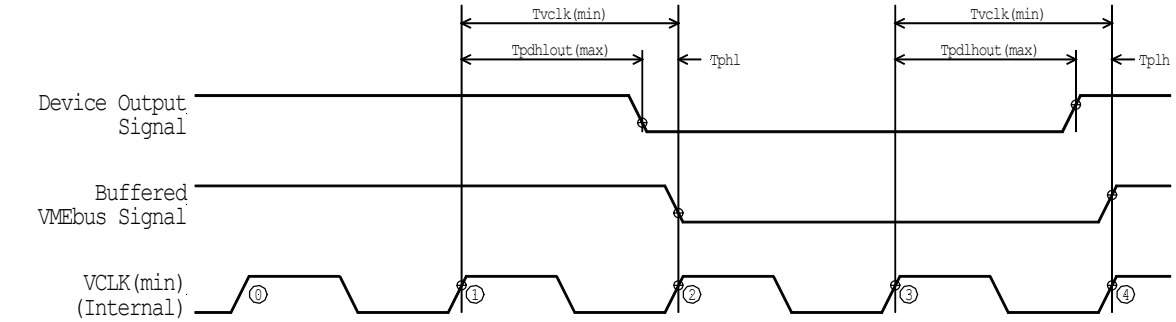


Figure 4-31. Determining bus interface timing characteristics.



(a) VMEcore Input Timing Constraints

Device input to VCLK setup, high-to-low transition: Tpdhlin(max) = Tvclk(min) - Tphl
Device input to VCLK setup, low-to-high transition: Tpdlhin(max) = Tvclk(min) - Tplh



(b) VMEcore Output Timing Constraints

VCLK to device output, high-to-low transition: Tpdhlout(max) = Tvclk(min) - Tphl
VCLK to device output, low-to-high transition: Tpdlhout(max) = Tvclk(min) - Tplh

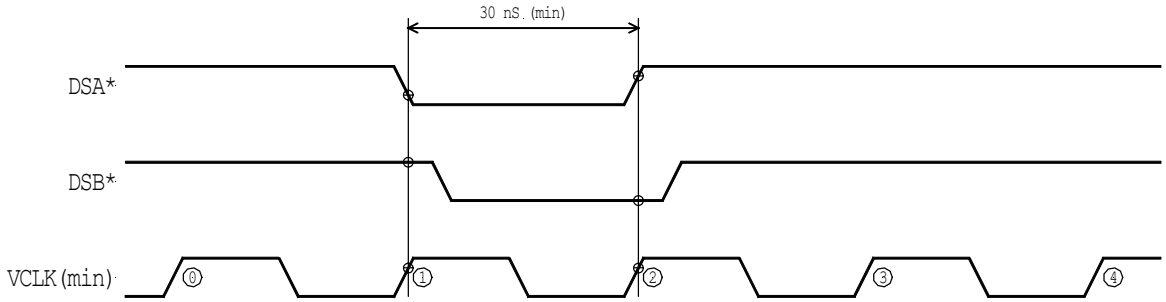
FEB 08, 2002

Figure 4-32. VMEbus interface timing.

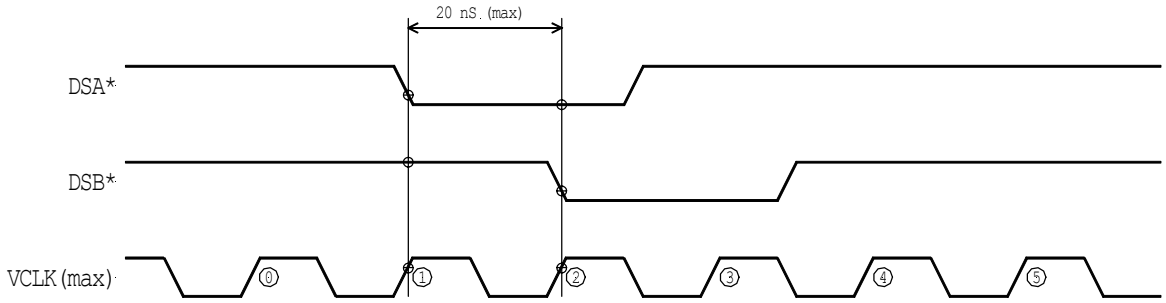
The *flop-to-flop transition time* is the time it takes for a synchronous signal (inside the core) to move from the output of one flip-flop and be set-up at the input of another. Stated another way, it is the internal timing of the core's RTL[25] logic. This is shown as [Tvclk(min)] in Figure 4-32(a). For example, if [VCLK] operates at 50.000 MHz, then [Tvclk(min)] is 1/VCLK, or 20.0 ns. It would be entered into a Xilinx FPGA router thusly:

```
NET "VCLK" PERIOD = 30.000;
```

There are additional constraints place on [VCLK] by the VMEbus timing specification. Since VMEbus is asynchronous it must be sampled sufficiently fast to prevent aliasing problems. As shown in Figure 4-33 the core frequency must be at least 33.333 MHz (or a period of 30.0 ns). However, it can't go above 50.000 MHz because the period of [VCLK] must not exceed the data strobe skew on the backplane.



(a) AT VCLK(min) (33.333 MHz) AT LEAST ONE ASSERTED SAMPLE IS ASSURED ON BOTH DATA STROBES.



(b) AT VCLK(max) (50.000 MHz) AT LEAST ONE ASSERTED SAMPLE IS ASSURED AT MAXIMUM BUS SKEW.

JAN 24, 2002

Figure 4-33. VMEbus constraints place on [VCLK] frequency.

[25] Register-transfer-logic.

The *clock-to-output delay time* is the time it takes for a synchronous signal to exit a flip-flop, propagate through the FPGA or ASIC target device, an output buffer, and then arrive at the VMEbus signal interconnection.

For example, consider a VMEbus input signal that traverses through an output buffer that has a timing delay [Tplh] of 10.0 ns. Furthermore, assume that [VCLK] operates at a clock speed of 50.000 MHz (or a period of 20.0 ns). This means that the target device must be routed with a worse case clock-to-output delay of:

$$Tpdlhout(max) = Tvclk(min) - Tphl$$

$$Tpdlhout(max) = 20.0\ ns - 10.0\ ns = 10.0\ ns$$

Although these numbers are given for the 'low-to-high' transition delay, a similar case exists for the 'high-to-low' transition delay. The method by which this time specification is entered into the software development tools depends upon the target technology and router used. For example, the timespec for the VMEbus data line [VD<0>] output would be entered into a Xilinx FPGA router thusly:

```
NET "VD<0>" OFFSET = OUT 10 ns AFTER "VCLK";
```

The timing diagram of Figures 4-34 and 4-35 show how VMEbus read and write cycles are translated to WISHBONE read and write cycles.

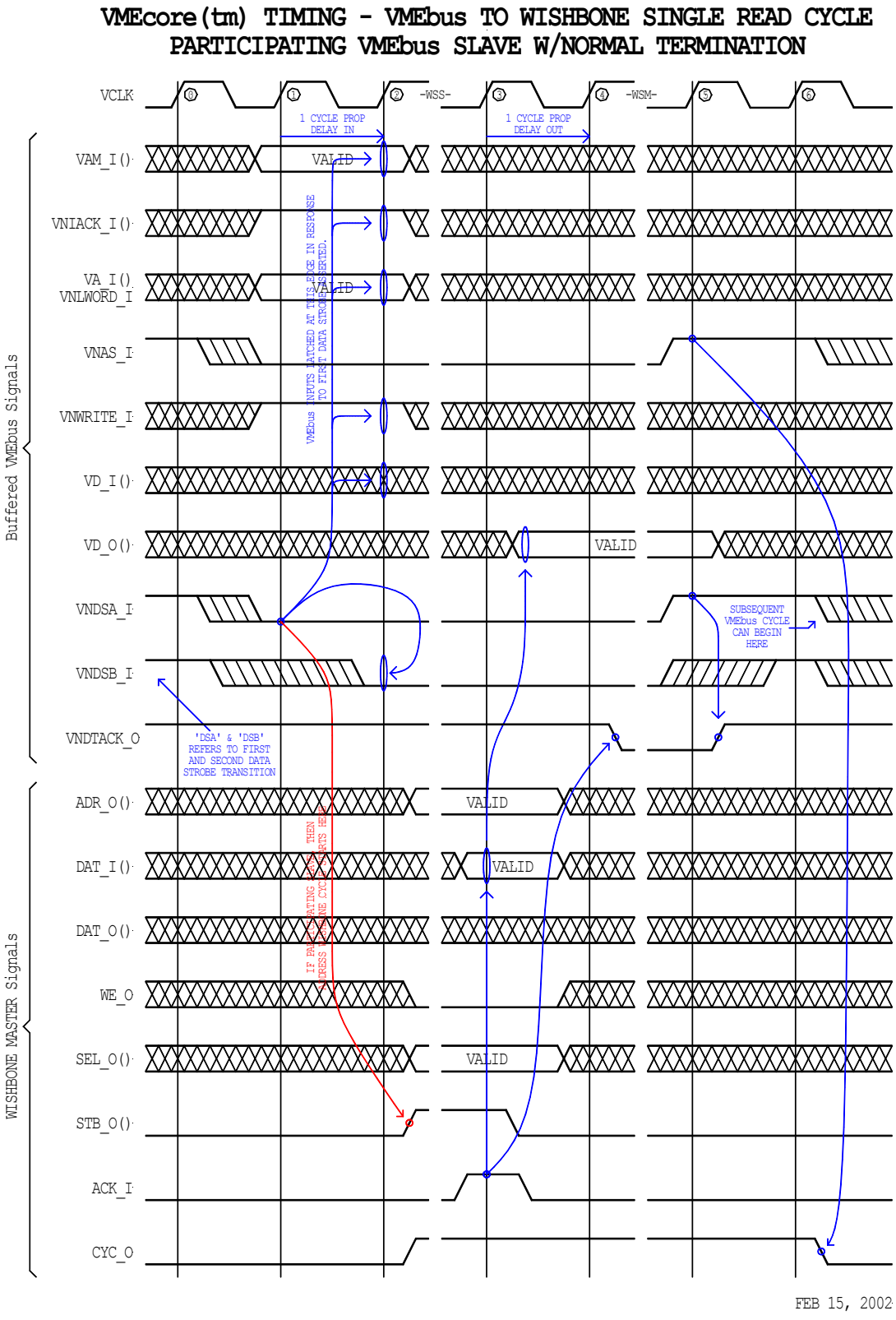


Figure 4-34. VMEbus to WISHBONE read cycle translation.

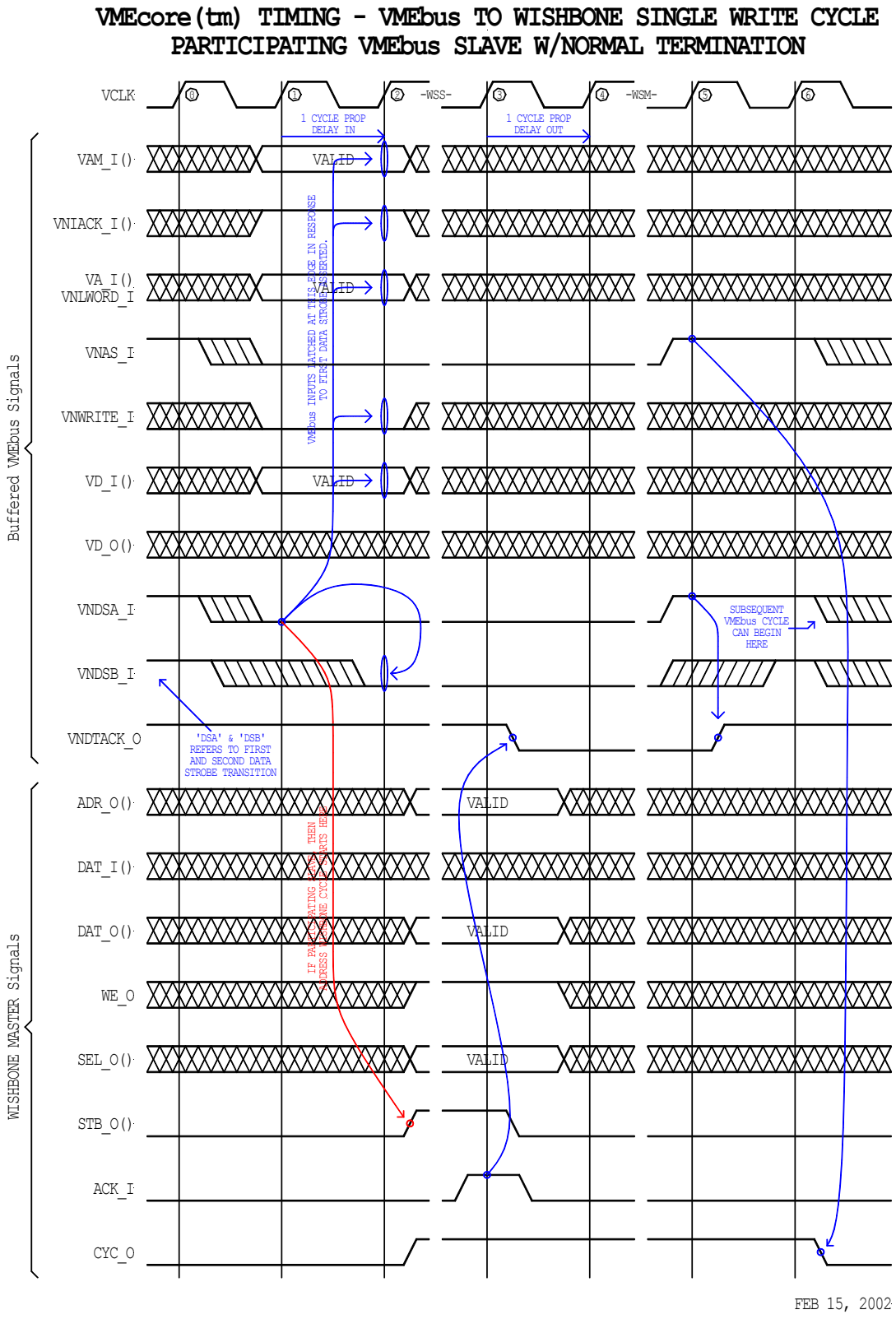


Figure 4-35. VMEbus to WISHBONE write cycle translation.

4.7.5 BUSSCNTC Entity

The BUSSCNTC (bus controller) entity renames signals used in the core.

4.7.6 LOCLADRC Entity

The LOCLADRC entity latches and routes the VME address bus [VA_I(18..02)] to the WISHBONE address bus [ADR_O(18..02)]. The upper VME address bits [VA_I(23..19)] are only used for address decoding purposes, and are not incorporated into the local SoC address bus. The lower two WISHBONE address bits [ADR_O(01..00)] are not present because they are encoded into select signals [SEL_O(3..0)].

4.7.7 LOCLDATC Entity

The LOCLDATC entity provides two functions: (a) it multiplexes VMEBus data to the correct WISHBONE data bank and (b) it latches the data.

4.7.8 SLAVCNTC Entity

The SLAVCNTC (slave controller) performs miscellaneous control functions for the VMEbus slave interface. This includes VMEbus address and data strobe synchronization, logic sequencer, cycle type generator, slave select logic, slave strobe logic, cycle termination generator (for DTACK*, etc.) and other functions. The SLAVCNTC entity includes the following VHDL processes:

- DCYC process
- DTACK_GENR process
- SEL_GENR process
- SEQU process
- STROBES process
- SYNC process
- TERMINATOR process

The DCYC (data cycle function generator) process identifies the type of participating VMEbus data cycle. During D32 cycles, it asserts signal D32. During D16EO cycles, it asserts D16EO. These signals are used by other processes to correctly route data through the interface.

The DTACK_GENR process is drives the VMEbus DTACK* and BERR* signals. The process also informs other parts of the interface that read data is available the [DREAD] signal.

The [DREAD] signal is asserted when local bus data is available during read cycles. This informs other circuits that data should be latched and presented to the VMEbus data lines.

The [VNDTACK_O] signal is connected to DTACK* on the VMEbus interface. During write cycles [VNDTACK_O] is asserted one [CLK_I] edge after WISHBONE acknowledge [ACK_I] is asserted. During read cycles the interface waits for an extra clock cycle before asserting [VNDTACK_O]. This provides extra time to route data through the target device. If the WISHBONE [ERR_I] signal is asserted instead of [ACK_I], then the [VNBERR_O] signal will be asserted instead of [VNDTACK_O]. This indicates that an error occurred during the cycle, and causes the VMEbus [BERR*] signal to be asserted.

The SEL_GENR process drives the WISHBONE MASTER [SEL_O(3..0)] data select signal array. The signal array is used for bank select lines during data transfers. Figure 4-36 shows how the data select signals are asserted during VMEbus transfers.

All of the select signals are asserted during the data load portion of a participating SLAVE cycle. This is indicated by the local [DLOAD] signal, which is generated by the SEQU process. Furthermore, the assertion of an individual signal in the [SEL_O()] array depends on the type of VMEbus data cycle (D32 or D16(EO)) and the address of the transfer. If an individual signal is selected it remains asserted until the cycle is terminated (using [ACK_I], etc.), or when a VMEbus cycle is aborted (indicated by the negation of VMEbus [DS0*] or [DS1*].

VMEcore(tm) WISHBONE MASTER Data Bus Routing - LDSIZE:32

	VMEbus Signals					SEL_O(3)	SEL_O(2)	SEL_O(1)	SEL_O(0)
						DAT_I(31..0) / DAT_O(31..0)			
Cycle	IACK*	DS1*	DS0*	LWORD*	A1	31..24	23..16	15..08	07..00
M/S D32	1	0	0	0	0	BYTE(0) D31..D24	BYTE(1) D23..D16	BYTE(2) D15..D08	BYTE(3) D07..D00
MASTER/SLAVE D16(EO)	1	EVEN	ODD	1	1			BYTE(2) D15..D08	BYTE(3) D07..D00
	1	EVEN	ODD	1	0	BYTE(0) D15..D08	BYTE(1) D07..D00		

NOTES: EVEN - DS1* asserted during D16 or even byte transfer.
ODD - DS0* asserted during D16 or odd byte transfer.
LW* - VMEbus LWORD* Signal
(DB) - Carries Data Bit
SEL_X() - For SLAVE or IREQ: 'X' => 'I'; for MASTER or IHAND: 'X' => '0'.

23 SEP 2002

Figure 4-36. Data select signals during VMEbus transfers.

The SEQU process is a state machine (sequencer) that generates master timing for the core. Figures 4-37 and 4-38 show the state and timing diagrams for the process.

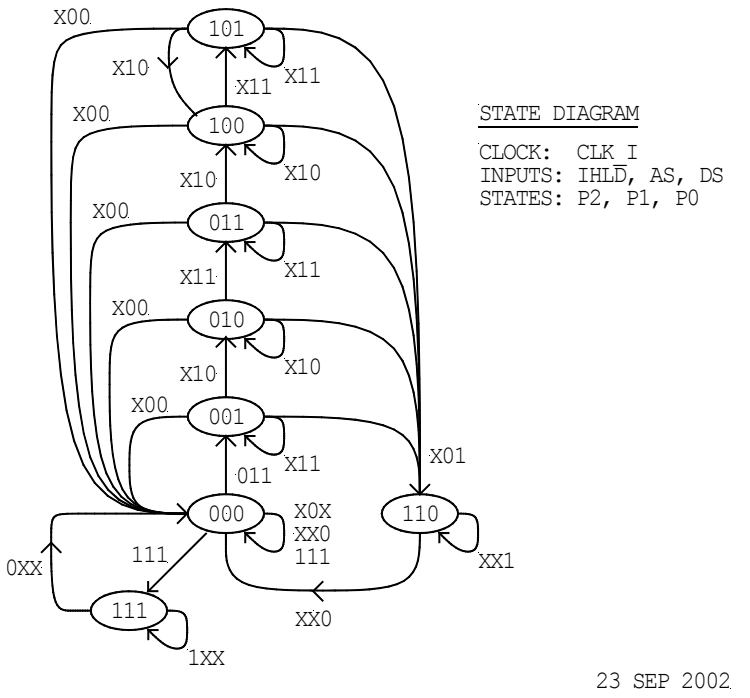


Figure 4-37. State diagram for SEQU process state machine.

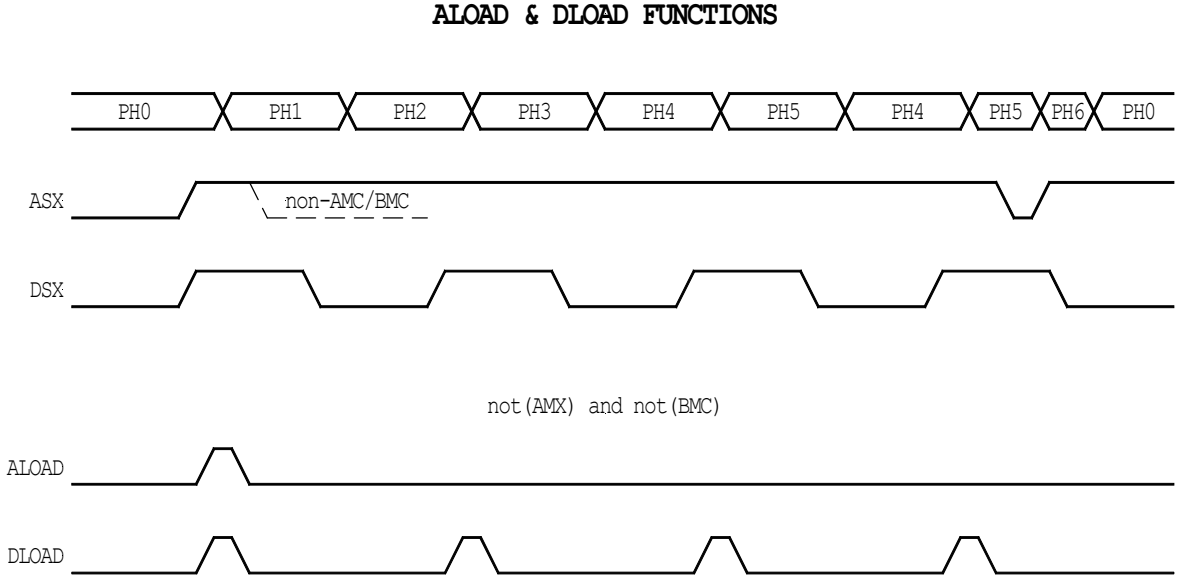


Figure 4-38. Timing diagram (relative) for the SEQU process state machine.

The STROBES process drives the WISHBONE [CYC_O], [STB_O] and [WE_O] signals. For more information about these signals please refer to the WISHBONE specification.

The SYNC process synchronizes the VMEbus AS*, DS0* and DS1* signals. It generates local signals [AS], [DS] and [SLDS].

The TERMINATOR process drives a common local signal [ACK] in response to the assertion of [ACK_I] or [ERR_I]. In general, either of these signals can be used to terminate a WISHBONE bus cycle.

4.7.9 SLAVDECC Entity

The SLAVDECC entity decodes the upper five bits of the VMEbus address bus and the address modifier code. When a participating VMEbus interface is selected, the entity asserts the address compare signal [ADC]. The circuit responds to address modifier codes 0x3E, 0x3D, 0x3A and 0x39.

The upper five VMEbus address bits [VA_I(23..19)] are compared to local address compare bits [SAC24_I(23..19)]. When these match (along with the address modifier codes), the interface is selected for a bus cycle.

The interface may be enabled or disabled with signal [SAE24_I]. If the interface is to be permanently enabled, tie [SAE24_I] to logic '1'. Otherwise, it may be used as a generic control signal to enable or disable the interface.

4.7.10 VMEBADBC Entity

The VMEBADBC entity is the data multiplexor and register for the VMEbus data out (VD_O()) signal array. During read cycles, VMEbus output data is routed (using multiplexors) from the WISHBONE data input signal array [DAT_I()] to the correct VMEbus output data signal array [VD_O()]. The routing depends upon the type of VMEbus data transaction (i.e. D32, D16(E) WORD or D16(O) WORD).

4.8 VMEPCIBR Entity

The VMEPCIBR entity is a top level, dual, WISHBONE System-on-Chip (SoC). It is a classic public domain WISHBONE shared bus with multiplexor interconnections that has been modified in several ways. These modifications include:

- Single WISHBONE MASTER operation.
- Dual WISHBONE interconnection buses for VMEbus and PCI sides of the bridge.
- Re-encoded, variable address decoder.

The VMEPCIBR entity is the highest 'RTL' level of the system. A description of the system hierarchy can be found with the VMEPCIBR_SOC description located elsewhere in this manual.

Figure 4-39 is representative of the WISHBONE SoC for both the VMEbus and PCI sides of the bridge. These are called the 'A SIDE' and 'B SIDE' system interconnections, respectively. On the 'A SIDE' interconnection the VMEbus SLAVE (VMEcore) is the WISHBONE MASTER. On the 'B SIDE' interconnection the PCI target (the Xilinx LogiCORE PCI core) is the WISHBONE MASTER. Both sides of the bridge are interconnected through the WISHBONE SLAVEs (i.e. registers and memory buffers).

The WISHBONE variable address decoder was also modified. WISHBONE system address decoders are simplest (and work fastest) when all of the SLAVEs are decoded at locations that are 'powers-of-two' (2, 4, 8, 16, etc.). However, in this design the SLAVES are located at other addresses, so a *re-encoded variable address decoder* is used. Functionally, this is a standard address decoder except that the decoded outputs are again re-encoded to generate evenly spaced select signals for the [ACK_O] and [DRD()] multiplexors. This will slow the system down somewhat, but is necessary in order to generate the correct address map.

Figure 4-40 shows the address decoding for the VMEbus (A_SIDE) WISHBONE interconnection. The PCI side of the bridge has a similar map that is not shown.

Figure 4-41 shows the error decoding for the VMEbus side of the bridge. One requirement of the system is that all non-decoded addresses return an error [ERR_I]. The error decoder output is asserted whenever an error address is selected.

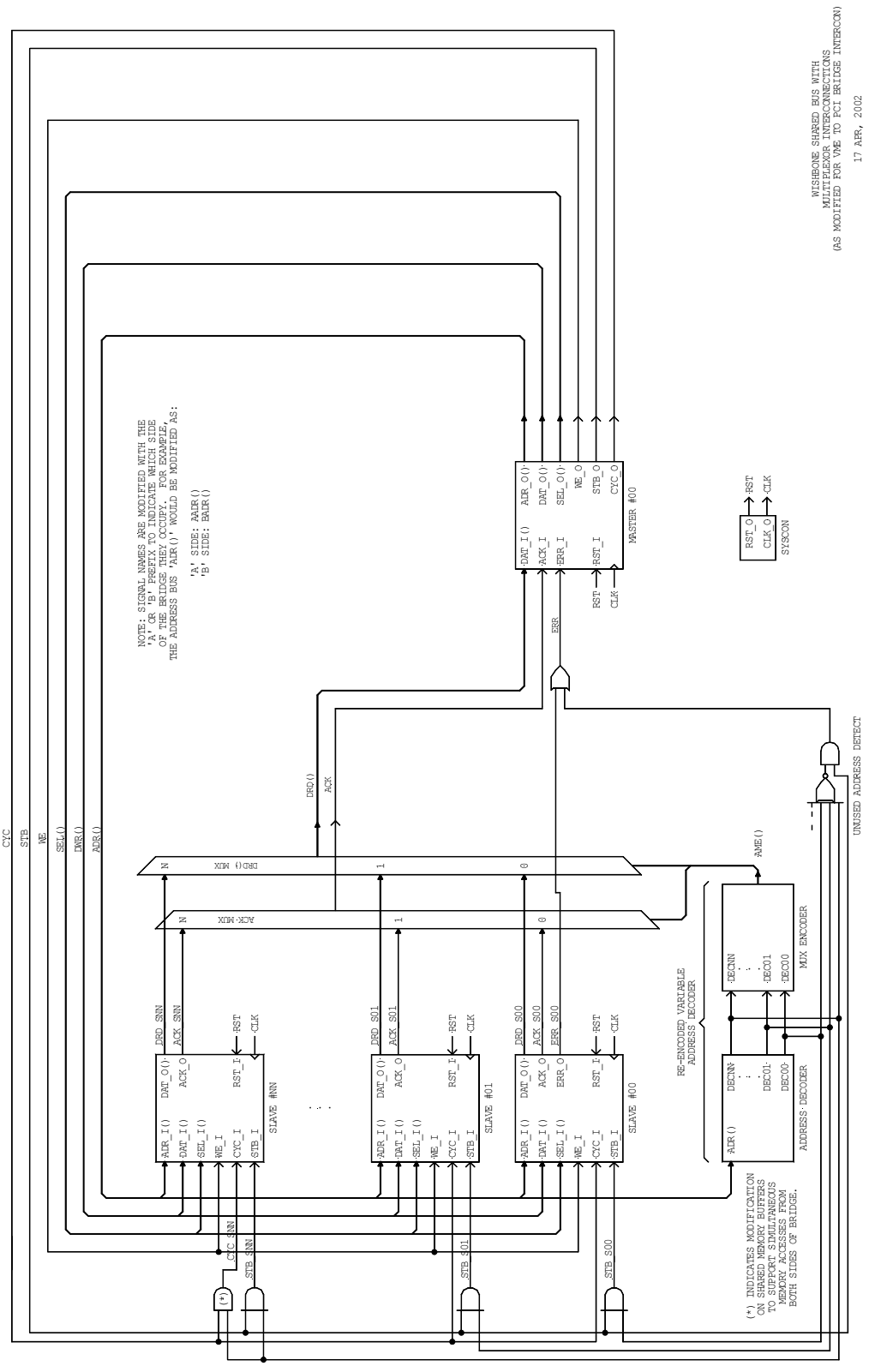


Figure 4-39. Modified public domain WISHBONE shared bus.

DESTINATION ENTITY / FILE NAME	DECODE SIGNAL NAME	DECODE REGION - VMEbus BYTE ADDRESSING	VA_I(18)	VA_I(17)	VA_I(16)	VA_I(15)	VA_I(14)	VA_I(13)	VA_I(12)	VA_I(11)	VA_I(10)	VA_I(09)	VA_I(08)	VA_I(07)	VA_I(06)	VA_I(05)	VA_I(04)	VA_I(03)	VA_I(02)	SEL(3.0)	MUX ENCODING AME()
			AADR(17)	AADR(16)	AADR(15)	AADR(14)	AADR(13)	AADR(12)	AADR(11)	AADR(10)	AADR(09)	AADR(08)	AADR(07)	AADR(06)	AADR(05)	AADR(04)	AADR(03)	AADR(02)	AADR(01)		
MISCREGc.VHD	ADEC00	0x00000	00000					0	0	0	0	0	0	0	0	0	0	0	0	00	0000
		0x0001F	00000					0	0	0	0	0	0	0	0	0	1	1	1	11	
		DECODE	00000					0	0	0	0	0	0	0	0	0	X	X	X	--	
SEMAREGc.VHD 'A'	ADEC01	0x00020	00000					0	0	0	0	0	0	0	0	1	0	0	0	00	0001
		0x00023	00000					0	0	0	0	0	0	0	0	1	0	0	0	11	
		DECODE	00000					0	0	0	0	0	0	0	0	1	0	0	0	--	
SEMAREGc.VHD 'B'	ADEC02	0x00024	00000					0	0	0	0	0	0	0	0	1	0	0	1	00	0010
		0x00027	00000					0	0	0	0	0	0	0	0	1	0	0	1	11	
		DECODE	00000					0	0	0	0	0	0	0	0	1	0	0	1	--	
SEMAREGc.VHD 'C'	ADEC03	0x00028	00000					0	0	0	0	0	0	0	0	1	0	1	0	00	0011
		0x0002B	00000					0	0	0	0	0	0	0	0	1	0	1	0	11	
		DECODE	00000					0	0	0	0	0	0	0	0	1	0	1	0	--	
SEMAREGc.VHD 'D'	ADEC04	0x0002C	00000					0	0	0	0	0	0	0	0	1	0	1	1	00	0100
		0x0002F	00000					0	0	0	0	0	0	0	0	1	0	1	1	11	
		DECODE	00000					0	0	0	0	0	0	0	0	1	0	1	1	--	
SEMAREGc.VHD 'E'	ADEC05	0x00030	00000					0	0	0	0	0	0	0	0	1	1	0	0	00	0101
		0x00033	00000					0	0	0	0	0	0	0	0	1	1	0	0	11	
		DECODE	00000					0	0	0	0	0	0	0	0	1	1	0	0	--	
SEMAREGc.VHD 'F'	ADEC06	0x00034	00000					0	0	0	0	0	0	0	0	1	1	0	1	00	0110
		0x00037	00000					0	0	0	0	0	0	0	0	1	1	0	1	11	
		DECODE	00000					0	0	0	0	0	0	0	0	1	1	0	1	--	
SEMAREGc.VHD 'G'	ADEC07	0x00038	00000					0	0	0	0	0	0	0	0	1	1	1	0	00	0111
		0x0003B	00000					0	0	0	0	0	0	0	0	1	1	1	0	11	
		DECODE	00000					0	0	0	0	0	0	0	0	1	1	1	0	--	
SEMABUFc.VHD 'A'	ADEC08	0x00800	00000					0	0	1	0	0	0	0	0	0	0	0	0	00	1000
		0x00BFF	00000					0	0	1	0	1	1	1	1	1	1	1	1	11	
		DECODE	00000					0	0	1	0	X	X	X	X	X	X	X	X	--	
SEMABUFc.VHD 'B'	ADEC09	0x00C00	00000					0	0	1	1	0	0	0	0	0	0	0	0	00	1001
		0x00FFF	00000					0	0	1	1	1	1	1	1	1	1	1	1	11	
		DECODE	00000					0	0	1	1	X	X	X	X	X	X	X	X	--	
SEMABUFc.VHD 'C'	ADEC0A	0x01000	00000					0	1	0	0	0	0	0	0	0	0	0	0	00	1010
		0x013FF	00000					0	1	0	0	1	1	1	1	1	1	1	1	11	
		DECODE	00000					0	1	0	0	X	X	X	X	X	X	X	X	--	
SEMABUFc.VHD 'D'	ADEC0B	0x01400	00000					0	1	0	1	0	0	0	0	0	0	0	0	00	1011
		0x017FF	00000					0	1	0	1	1	1	1	1	1	1	1	1	11	
		DECODE	00000					0	1	0	1	X	X	X	X	X	X	X	X	--	
SEMABUFc.VHD 'E'	ADEC0C	0x01800	00000					0	1	1	0	0	0	0	0	0	0	0	0	00	1100
		0x01BFF	00000					0	1	1	0	1	1	1	1	1	1	1	1	11	
		DECODE	00000					0	1	1	0	X	X	X	X	X	X	X	X	--	
SEMABUFc.VHD 'F'	ADEC0D	0x01C00	00000					0	1	1	1	0	0	0	0	0	0	0	0	00	1101
		0x01FFF	00000					0	1	1	1	1	1	1	1	1	1	1	1	11	
		DECODE	00000					0	1	1	1	X	X	X	X	X	X	X	X	--	
SEMABUFc.VHD 'G'	ADEC0E	0x02000	00000					1	0	0	0	0	0	0	0	0	0	0	0	00	1110
		0x023FF	00000					1	0	0	0	1	1	1	1	1	1	1	1	11	
		DECODE	00000					1	0	0	0	X	X	X	X	X	X	X	X	--	
SEMABUFc.VHD 'H'	ADEC0F	0x02400	00000					1	0	0	1	0	0	0	0	0	0	0	0	00	1111
		0x027FF	00000					1	0	0	1	1	1	1	1	1	1	1	1	11	
		DECODE	00000					1	0	0	1	X	X	X	X	X	X	X	X	--	

'A' SIDE ADDRESS DECODING
25 JUL 2002

Figure 4-40. VMEbus (A_SIDE) address decoder.

LOCATION NAME	DECODE REGION - VMEbus BYTE ADDRESSING	VA_I(18)	VA_I(17)	VA_I(16)	VA_I(15)	VA_I(14)	VA_I(13)	VA_I(12)	VA_I(11)	VA_I(10)	VA_I(09)	VA_I(08)	VA_I(07)	VA_I(06)	VA_I(05)	VA_I(04)	VA_I(03)	VA_I(02)	VA_I(01)	SEL(3.0)
		AADR(17)	AADR(16)	AADR(15)	AADR(14)	AADR(13)	AADR(12)	AADR(11)	AADR(10)	AADR(09)	AADR(08)	AADR(07)	AADR(06)	AADR(05)	AADR(04)	AADR(03)	AADR(02)	AADR(01)		
HIGH REGISTERS	0x0003C	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	00	
	0x0003F	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	11	
	DECODE	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	--	
	0x00040	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	00	
	0x0007F	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	11	
	DECODE	0	0	0	0	0	0	0	0	0	0	0	0	1	X	X	X	X	--	
	0x00080	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	00	
	0x000FF	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	11	
	DECODE	0	0	0	0	0	0	0	0	0	0	0	1	X	X	X	X	X	--	
	0x00100	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	00	
	0x001FF	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	11	
	DECODE	0	0	0	0	0	0	0	0	0	0	1	X	X	X	X	X	X	--	
	0x00200	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	00	
	0x003FF	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	11	
	DECODE	0	0	0	0	0	0	0	0	0	1	X	X	X	X	X	X	X	--	
	0x00400	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	00	
	0x007FF	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	11	
	DECODE	0	0	0	0	0	0	0	0	1	X	X	X	X	X	X	X	X	--	
HIGH MEMORY	0x02400	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	00	
	0x027FF	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	11	
	DECODE	0	0	0	0	0	1	0	0	1	X	X	X	X	X	X	X	X	--	
	0x02800	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	00	
	0x02FFF	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	11	
	DECODE	0	0	0	0	0	1	0	1	X	X	X	X	X	X	X	X	X	--	
	0x03000	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	00	
	0x03FFF	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	11	
	DECODE	0	0	0	0	0	1	1	X	X	X	X	X	X	X	X	X	X	--	
	0x04000	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	00	
	0x07FFF	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	11	
	DECODE	0	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	--	
	0x08000	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	00	
	0x0FFFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	11	
	DECODE	0	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	--	
	0x10000	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	
	0x1FFFF	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	11	
	DECODE	0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	--	
	0x20000	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	
	0x3FFFF	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	11	
	DECODE	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	--	
	0x40000	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	
	0x7FFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	11	
	DECODE	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	--	

'A' SIDE ERROR DECODING
04 OCT 2002

Figure 4-41. VMEbus (A_SIDE) error decoder.

4.9 VMEPCIBR_SOC Entity

As shown in Figure 4-42, the highest level entity in the SoC hierarchy is VMEPCIBR_SOC. The entity has two functions: (1) to define the VMEPCIBR entity as a component (the highest level RTL in the SoC) and (2) to specify all of the application specific I/O pads. With the exception of the PCI I/O pads (which Xilinx locates in their PCI core), all I/O pads are located here.

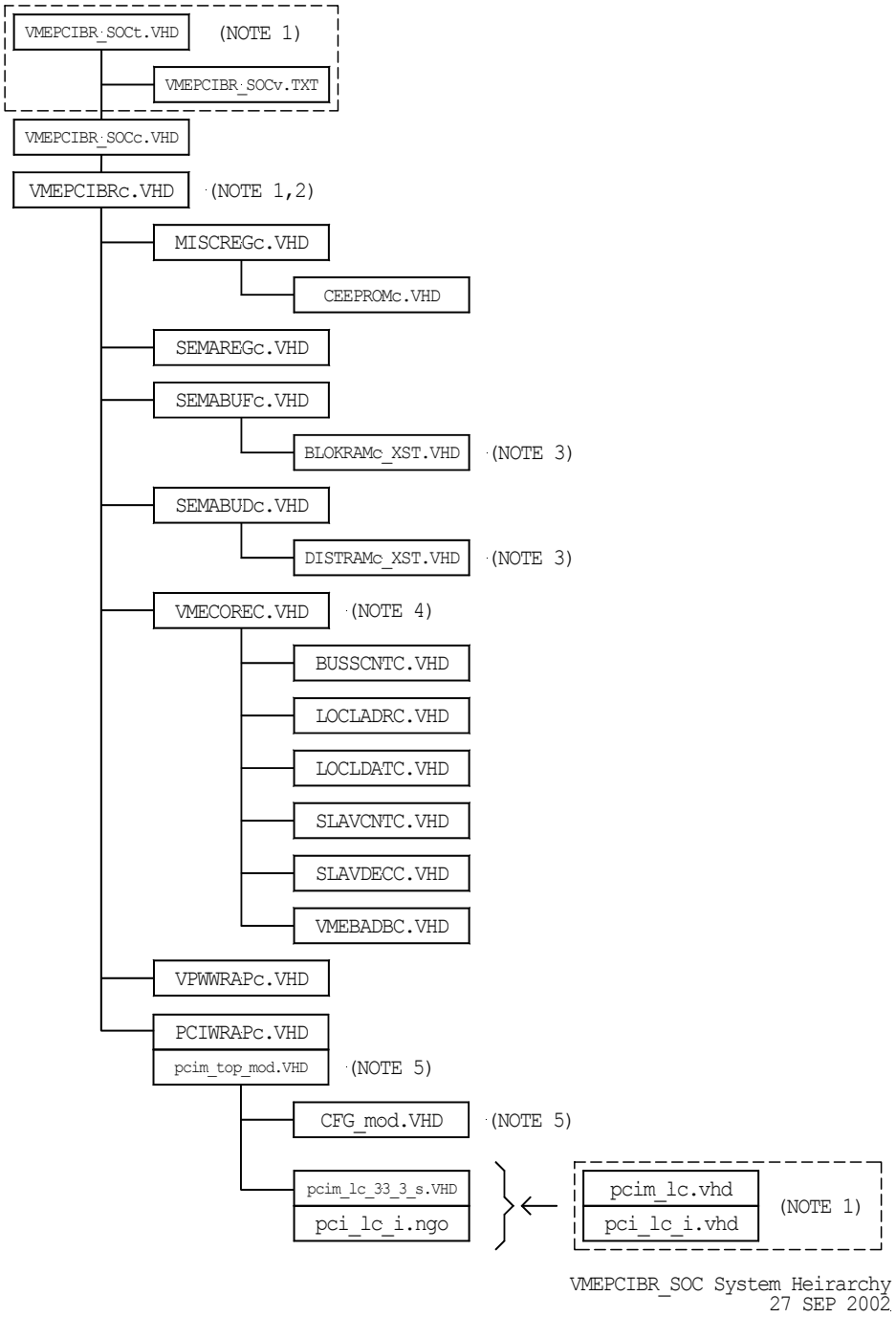


Figure 4-42(a). VMEPCIBR_SOC system hierarchy.

NOTES:

(1) Add or substitute the files shown during pre-synthesis simulation.

The top level test bench named 'VMETOPCI_SOCt.VHD' is used for both the pre-synthesis and post-routing top level simulation. Also note that the test bench uses (reads) a test vector file named 'VMETOPCI_SOCv.TXT'. If the simulator can't find the file, it probably means that its path (as declared in 'VMETOPCI_SOCt.VHD') is wrong.

The unmodified Xilinx PCI wrapper named 'pcim_lc_33_3_s.VHD' is used for synthesis. Substitute file 'pcim_lc.vhd' for pre-synthesis simulation.

The Xilinx PCI core black-box primative named 'pcim_lc_i.ngo' is incorporated into the system by the Xilinx router. Substitute file 'pcim_lc_i.vhd' for pre-synthesis simulation.

Special instructions when using ModelSim:
(a) After creating the project, be sure to check the "Use 1993 Language Syntax" under Options >> Compile.
(b) Compile all files from their source directory.

(2) During synthesis, the VHDL source file 'VMETOPCI_SOCc.VHD' is the top level entity.

Route the system with modified Xilinx user constraint file named 2s200pq208_32_33_mod.ucf.

(3) Contains block or distributed RAM primatives generated by Xilinx XST synthesis software.

(4) Silicore VMEcore(tm). Also note that all VMEcore(tm) files are in the VMECORE directory.

(5) Xilinx LogiCORE(tm) PCI file modified for this application. File is located under directory named 'PCIMODS'.

VMEPCIBR_SOC System Heirarchy
27 SEP 2002

Figure 4-42(b). VMEPCIBR_SOC system hierarchy (con't).

4.10 VPWWRAP Entity

VPWWRAP is a VMEbus posted write wrapper for the VMEcore entity. As shown in the functional diagram of Figure 4-43, it contains a register that holds data written from the VMEbus side of the bridge. Immediately after latching the write data, the entity simultaneously: (1) acknowledges the VMEbus cycle by asserting [ACK_O], and (2) begins a WISHBONE write cycle with the latched data. Figure 4-44 and 4-45 shows the timing diagram for read and write cycles (respectively).

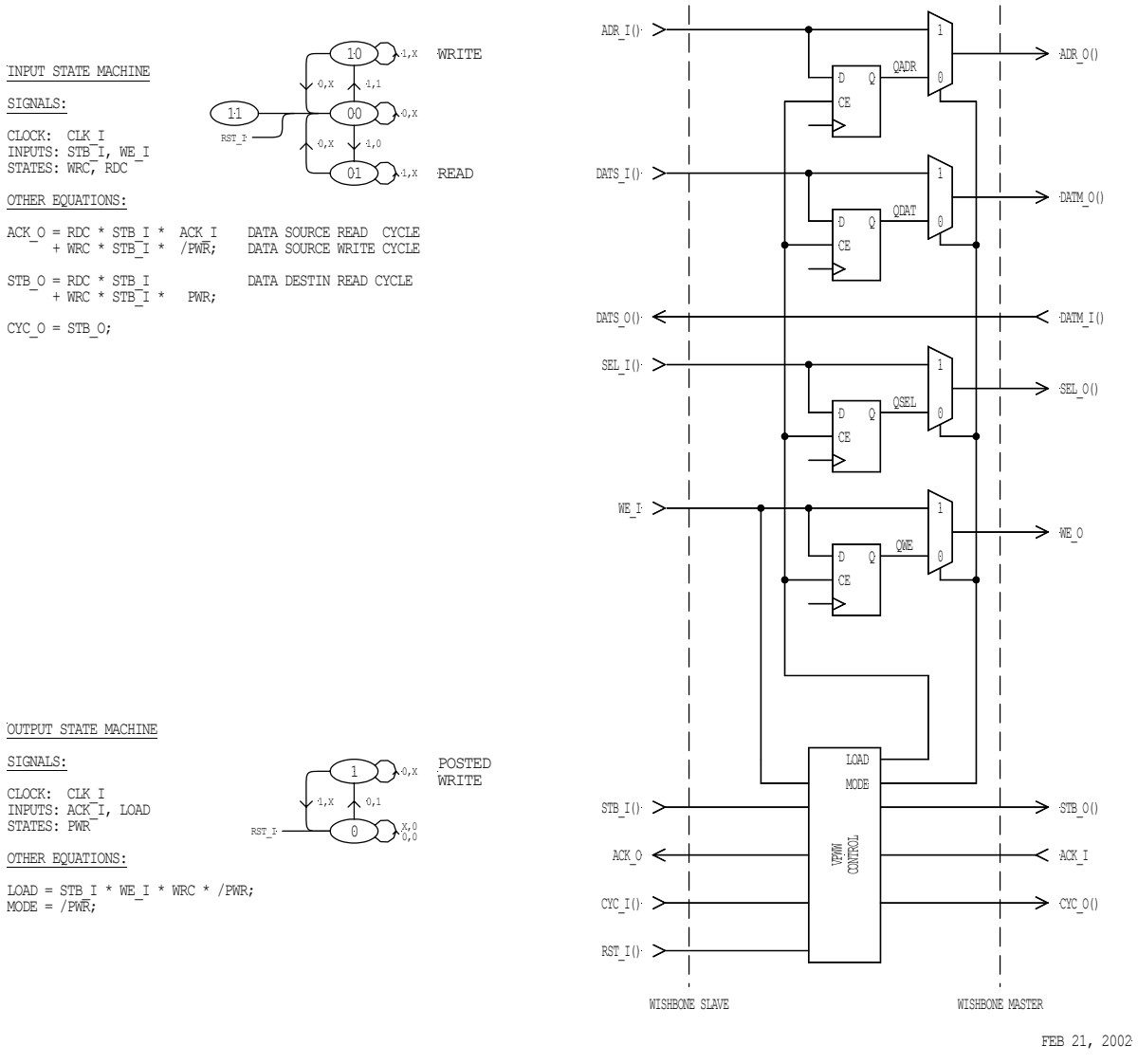


Figure 4-43. Functional diagram of the VPWWRAP entity.

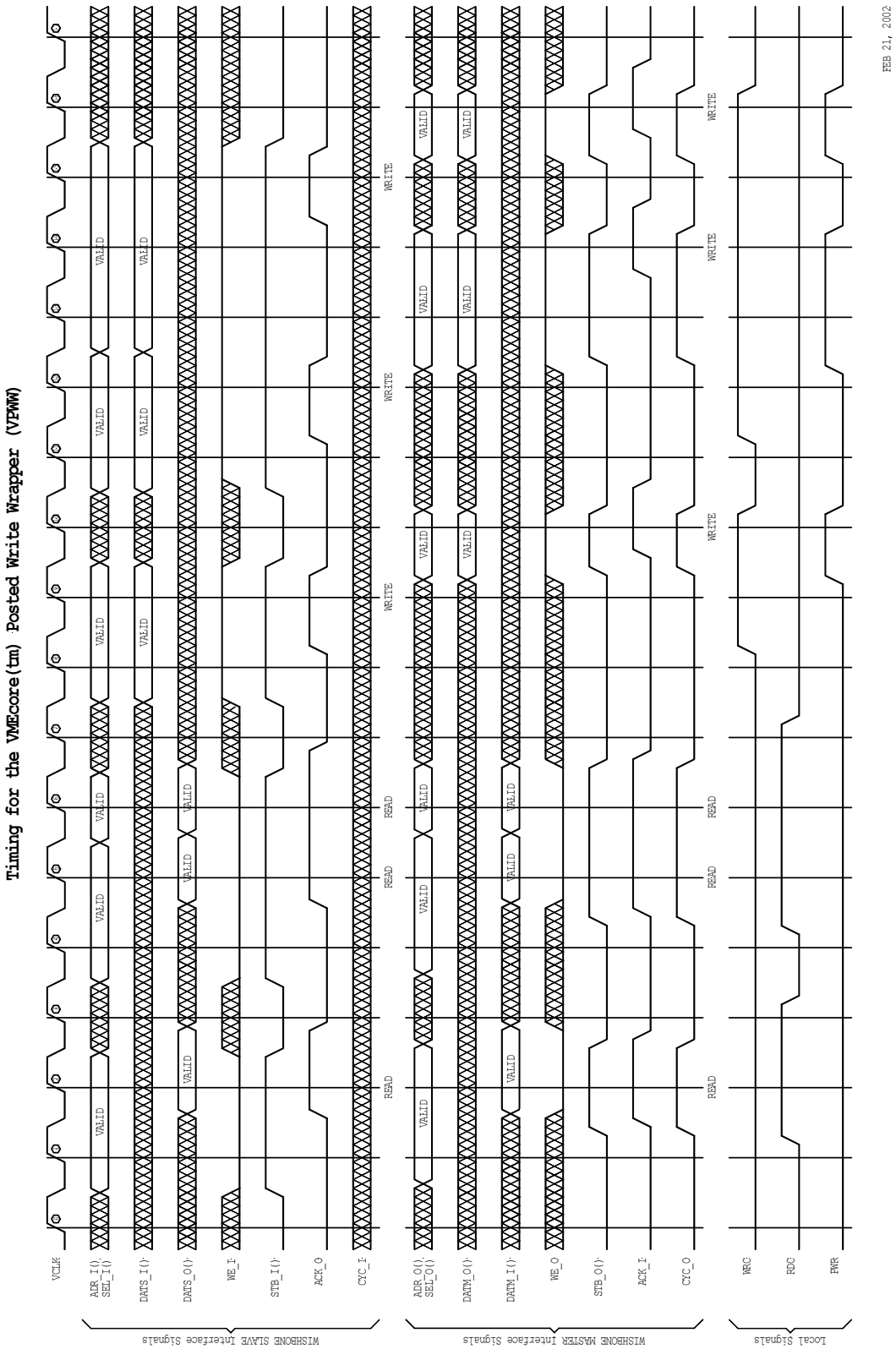


Figure 4-44. VPWWRAP read cycle.

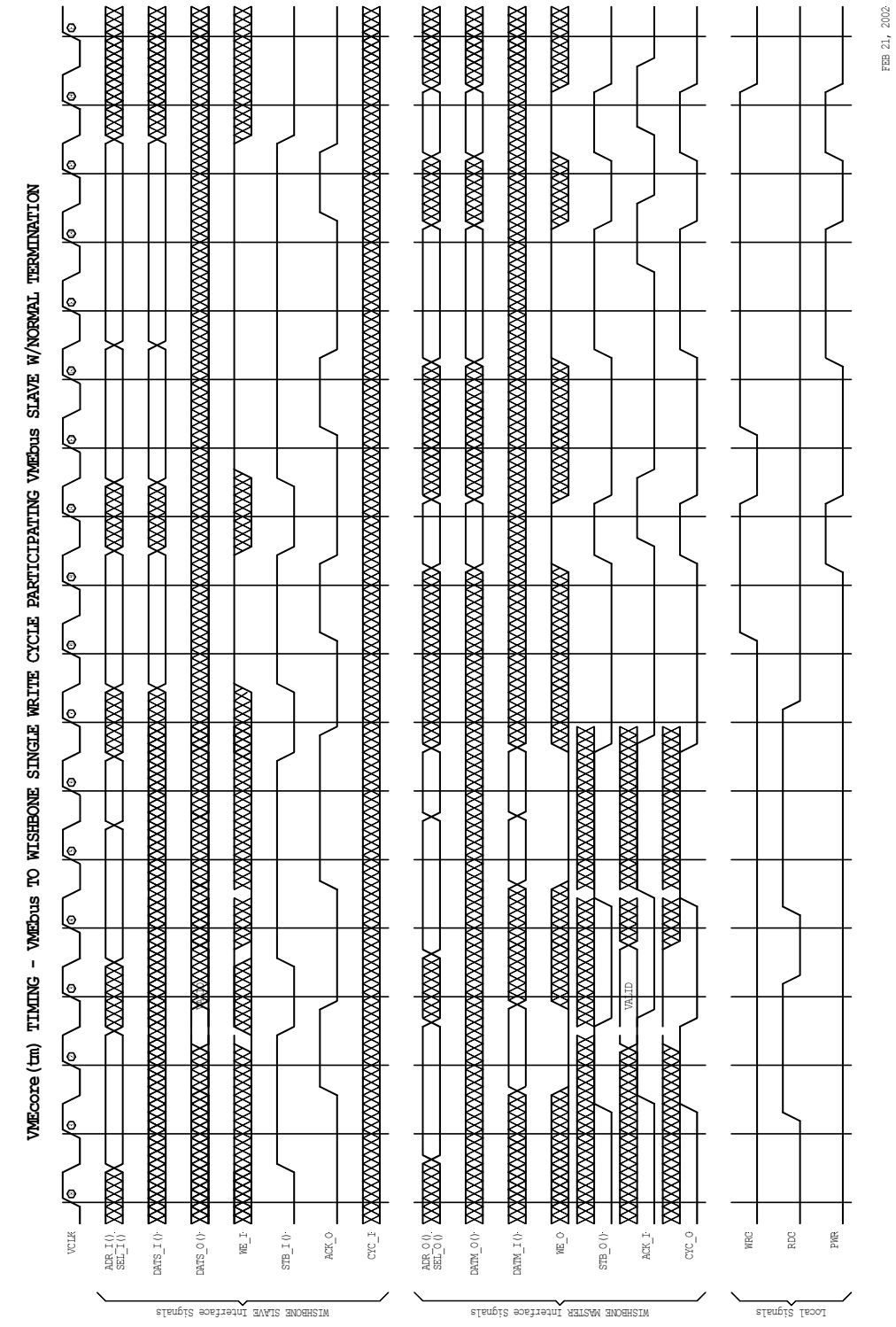


Figure 4-45. VPWWRAP write cycle.

Appendix A – GNU LESSER GENERAL PUBLIC LICENSE

GNU LESSER GENERAL PUBLIC LICENSE - Version 2.1, February 1999

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

PREAMBLE

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of
it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should

know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for

the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the au-

thor/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE

LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

HOW TO APPLY THESE TERMS TO YOUR NEW LIBRARIES

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the library's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>
>
> This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.
>
> This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.
>
> You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
>
> Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, ifnecessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library `Frob' (a library for tweaking knobs) written by James Random Hacker.
<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice

That's all there is to it!

Appendix C – GNU Free Documentation License

GNU Free Documentation License - Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connec-

tion with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by

reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the
Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this
License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index