



*Technical Reference Manual:*

## *WISHBONE Public Domain Library for VHDL*

*Revised: October 8, 2001*

***Silicore Corporation***

*6310 Butterworth Lane – Corcoran, MN 55340*

*TEL: (763) 478-3567 FAX: (763) 478-3568*

*[www.silicore.net](http://www.silicore.net)*



## **Stewardship**

Stewardship for this manual and the associated library files are maintained by Silicore Corporation. Questions, comments and suggestions about this document are welcome and should be directed to:

Wade D. Peterson, Silicore Corporation  
6310 Butterworth Lane – Corcoran, MN USA 55340  
TEL: (763) 478-3567; FAX: (763) 478-3568  
E-MAIL: [wadep@silicore.net](mailto:wadep@silicore.net) URL: [www.silicore.net](http://www.silicore.net)

Silicore Corporation maintains this document as a service to its customers and to the IP core industry as a whole. The intent is to improve the quality of Silicore products, as well as to foster cooperation among the users and suppliers of IP cores.

## **Copyright Release / Royalty Release / Patent Notice**

Notice is hereby given that this document and the associated library files are not copyrighted, and have been placed into the public domain. They may be freely copied and distributed by any means.

This manual and the associated library files may be used for the design and production of System-on-Chip (SoC) components without royalty or other financial obligation to Silicore Corporation.

The author(s) are not aware that the information contained herein cause infringement on the patent, copyright, trademark or trade secret rights of others. However, there is a possibility that such infringement may exist without their knowledge. The user of this document assumes all responsibility for determining if products designed with this information infringe on the intellectual property rights of others.

## **Disclaimer**

In no event shall Silicore Corporation be liable for incidental, consequential, indirect, or special damages resulting from the use of this document or its associated library files. The user assumes all responsibility for their use.

Silicore® is a registered service mark and trademark of Silicore Corporation.  
Xilinx® is a registered trademark of Xilinx, Inc.

# Contents

<b>INTRODUCTION .....</b>	<b>4</b>
<b>VHDL PORTABILITY .....</b>	<b>4</b>
<b>VHDL LIBRARY .....</b>	<b>5</b>
ARB0001A: 4 LEVEL, ROUND-ROBIN ARBITER .....	6
DMA0001A: SIMPLE 32-BIT DMA .....	10
ICN0001A: POINT-TO-POINT INTERCONNECTION .....	15
ICN0002A: 4 X 4 SHARED BUS INTERCONNECTION .....	16
MEM0001A: 8 X 32-BIT MEMORY INTERFACE FOR XILINX .....	19
MEM0002A: 8 X 32-BIT REGISTER BASED MEMORY .....	22
SYC0001A: SIMPLE SYSCON FOR FPGA .....	23

## Introduction

This manual describes various library files contained in the WISHBONE Public Domain Library for VHDL. The latest copy of the library and related documentation are available free of charge from the Silicore website at [www.silicore.net/wishbone.htm](http://www.silicore.net/wishbone.htm).

## VHDL Portability

It is assumed by the Steward that all simulation and synthesis tools conform to the following standards<sup>1</sup>:

- IEEE Standard VHDL Language Reference Manual, IEEE STD 1076-1993.
- IEEE Standard VHDL Synthesis Packages, IEEE STD 1073.3-1997.
- IEEE Standard Multivalued 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 the Steward so that the problem can be resolved (or at least documented).

It is strongly recommended that the user have a set of VHDL simulation tools before using the library components. This helps in two ways: (a) it builds confidence that the core synthesizes correctly and (b) it helps resolve any integration problems. The simulation tools do not need to be fancy...a simple non-graphical simulator is adequate.

All original VHDL source files have been edited with the 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 similar) editor.

---

<sup>1</sup> 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](http://www.ieee.org)

# VHDL Library

Each library file in this document has a core number associated with it. For example, there is a simple 32-bit DMA called ‘DMA0001a’. The first three letters (prefix) of the core number indicate the type of core, and fall into the five categories shown in Table 1.

Table 1. Description of prefixes used in libraries.	
Prefix	Description
ARB	Arbiter
DMA	Direct Memory Access unit
ICN	WISHBONE interconnection (INTERCON)
MEM	Memory element
SYC	WISHBONE system controller (SYSCON)

The last digit in the core number indicates the revision level of the core. Revision levels are applied only to bug fixes. If the form, fit or function of a core changes, then the part number of the core is also changed. All revisions of the core are retained in the library so that users can inspect them.

VHDL source code for each library core is located in the software distribution package. This includes the source code for the core as well as a VHDL test bench. Many of the designs use a test vector file (usually called ‘TESTVECT.TXT’). This is often done because users can re-write the core in another language (such as Verilog®) and still use the same test vectors to verify the design.

**ARB0001a: 4 Level, Round-robin Arbiter**

‘ARB0001a’ is a 4-level, round-robin arbiter. In shared bus systems the arbiter determines which MASTER can use the bus.

Round-robin arbiters give equal priority to all of the MASTERS. They grant the bus on a rotating basis much like the four position rotary switch shown in Figure 1. When a MASTER relinquishes the bus, the switch is turned to the next position and the bus is granted to the MASTER on that level. If a MASTER on any particular level is not making a bus request, then the arbiter skips over that level and continues onto the next one. In this way all of the MASTERS are granted the bus on an equal basis.

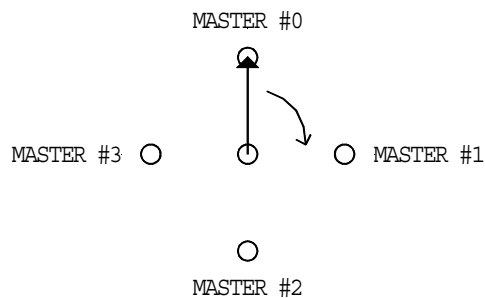


Figure 1. Round-robin arbiters grant the bus on a rotating basis much like a rotary switch.

Round-robin arbiters are popular in data acquisition systems where data is collected and placed into shared memory. Often the peripherals in those systems must off-load data to memory on an equal basis. Priority arbiters (where each MASTER is assigned a higher or lower level of priority) do not work well in these applications because some peripherals receive more bus bandwidth than others, thereby causing data ‘gridlock’.

One disadvantage of the round-robin arbiter is that it consumes a full clock cycle to perform the arbitration. Other WISHBONE arbiters, such as the priority arbiter or systems with no arbitration whatsoever, can be made to operate with a zero clock overhead.

**Arbiter Topology**

A block diagram of the arbiter is shown in Figure 2. Bus requests from each of the four MASTERs arrive at inputs [CYC0] to [CYC3]. When asserted, these signals indicate that a MASTER wants to use the shared bus. If the bus is in use, then the request is ignored. If the bus is free, then it asserts one of four grant lines called [GNT0] to [GNT3]. These correspond to the request signals [CYC0] to [CYC3] respectively.

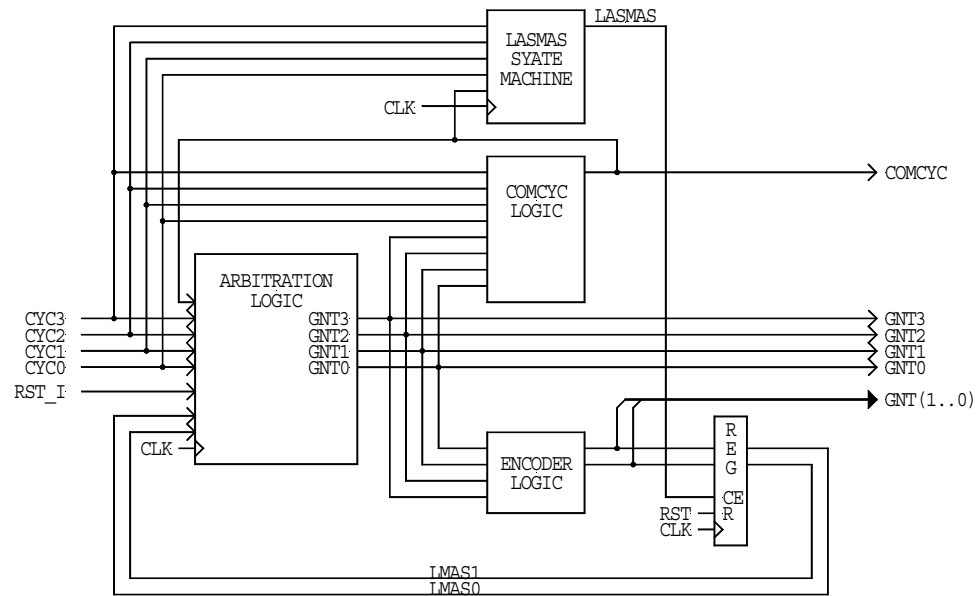
The [COMCYC] signal indicates whether the bus is busy or free. [COMCYC] is asserted whenever a MASTER has both requested the bus, and has been granted the bus by the arbiter. This signal is also used by the interconnection (external to the arbiter). For this reason, [COMCYC] is made visible outside of the arbiter.

The grant lines [GNT0] to [GNT3] are also encoded as [GNT(1..0)] and made available outside of the arbiter. These can be used by the interconnection in conjunction with the [COMCYC] signal to indicate which MASTER has been granted the bus. When [COMCYC] is asserted, then the MASTER located on [GNT(1..0)] is granted the bus.

Round-robin arbiters also keep track of the level of the previous MASTER. This is the main reason why they require a full clock cycle for arbitration. That's because the level to be granted is determined by the level of the previous MASTER. The level of the previous MASTER is saved in a register that latches the state of grant signals [GNT(1..0)]. The register latches the grant level when indicated by the 'LASMAS' state machine.

The timing diagram of Figure 3 shows a bus arbitration example. There, the MASTER on level 1 asserts a bus request on [CYC1] after edge 0. At the following edge (edge 1) the arbiter asserts grant signals [GNT(1..0)], and the [COMCYC] signal. This indicates to external logic that a valid arbitration has occurred. The arbitration level is latched using signal [LASMAS] at edge 2. This information is used later to determine which MASTER gets the bus.

When the MASTER on level 1 is done with the bus, it negates [CYC1] (after edge 2). In response to this, the arbiter negates [COMCYC], which initiates another arbitration. This second arbitration (after edge 3) is slightly more complex because three MASTERS are now requesting the bus. Since the first arbitration was granted to the MASTER on level #1, the second arbitration is granted to MASTER #2. The other MASTERS are ignored because the level 2 MASTER is next in the round-robin.



## ARBITRATION LOGIC (REGISTERED OUTPUT):

```

GNT0 := ( /RST * /COMCYC * /LMAS1 * /LMAS0 * /CYC3 * /CYC2 * /CYC1 * CYC0 )
+ ( /RST * /COMCYC * /LMAS1 * /LMAS0 * /CYC3 * /CYC2 * CYC0 )
+ ( /RST * /COMCYC * /LMAS1 * /LMAS0 * /CYC3 * CYC0 )
+ ( /RST * /COMCYC * /LMAS1 * /LMAS0 * CYC0 )
+ ( /RST * /COMCYC * GNT0 );

GNT1 := ( /RST * /COMCYC * /LMAS1 * /LMAS0 * CYC1 )
+ ( /RST * /COMCYC * /LMAS1 * /LMAS0 * /CYC3 * /CYC2 * CYC1 * /CYC0 )
+ ( /RST * /COMCYC * /LMAS1 * /LMAS0 * /CYC3 * CYC1 * /CYC0 )
+ ( /RST * /COMCYC * /LMAS1 * /LMAS0 * CYC1 * /CYC0 )
+ ( /RST * /COMCYC * GNT1 );

GNT2 := ( /RST * /COMCYC * /LMAS1 * /LMAS0 * CYC2 * /CYC1 )
+ ( /RST * /COMCYC * /LMAS1 * /LMAS0 * CYC2 )
+ ( /RST * /COMCYC * /LMAS1 * /LMAS0 * /CYC3 * CYC2 * /CYC1 * /CYC0 )
+ ( /RST * /COMCYC * /LMAS1 * /LMAS0 * CYC2 * /CYC1 * /CYC0 )
+ ( /RST * /COMCYC * GNT2 );

GNT3 := ( /RST * /COMCYC * /LMAS1 * /LMAS0 * CYC3 * /CYC2 * /CYC1 )
+ ( /RST * /COMCYC * /LMAS1 * /LMAS0 * CYC3 * /CYC2 )
+ ( /RST * /COMCYC * /LMAS1 * /LMAS0 * CYC3 * /CYC2 )
+ ( /RST * /COMCYC * /LMAS1 * /LMAS0 * CYC3 * /CYC2 * /CYC1 * /CYC0 )
+ ( /RST * /COMCYC * GNT3 );
    
```

## COMCYC LOGIC:

```

COMCYC = ( CYC3 * GNT3 )
+ ( CYC2 * GNT2 )
+ ( CYC1 * GNT1 )
+ ( CYC0 * GNT0 );
    
```

## ENCODER LOGIC:

```

GNT(1) = GNT3 + GNT2;
GNT(0) = GNT3 + GNT1;
    
```

## SHORTHAND NOTATION:

```

= COMBINATORIAL OUTPUT
:= REGISTERED OUTPUT
+ LOGICAL 'OR'
* LOGICAL 'AND'
/ LOGICAL 'NOT'
    
```

## 'LASMAS' STATE MACHINE:

GENERATES A CLOCK ENABLE TO THE LAST MASTER REGISTER.

INPUTS: BEGIN  
STATES: EDGE, LASMAS

RESET: ALL 'CYC' INPUTS FORCED LOW, THEREBY  
RESETTING THE ARBITER.

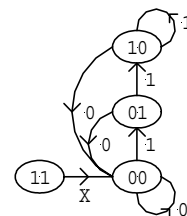
## STATE MACHINE EQUATIONS:

```

BEG = (CYC0 + CYC1 + CYC2 + CYC3) * /COMCYC;

LASMAS := ( BEG * /EDGE * /LASMAS );

EDGE := ( BEG * /EDGE * LASMAS )
+ ( BEG * EDGE * /LASMAS );
    
```



STATE DIAGRAM

Figure 2. Block diagram of ARB0001a.



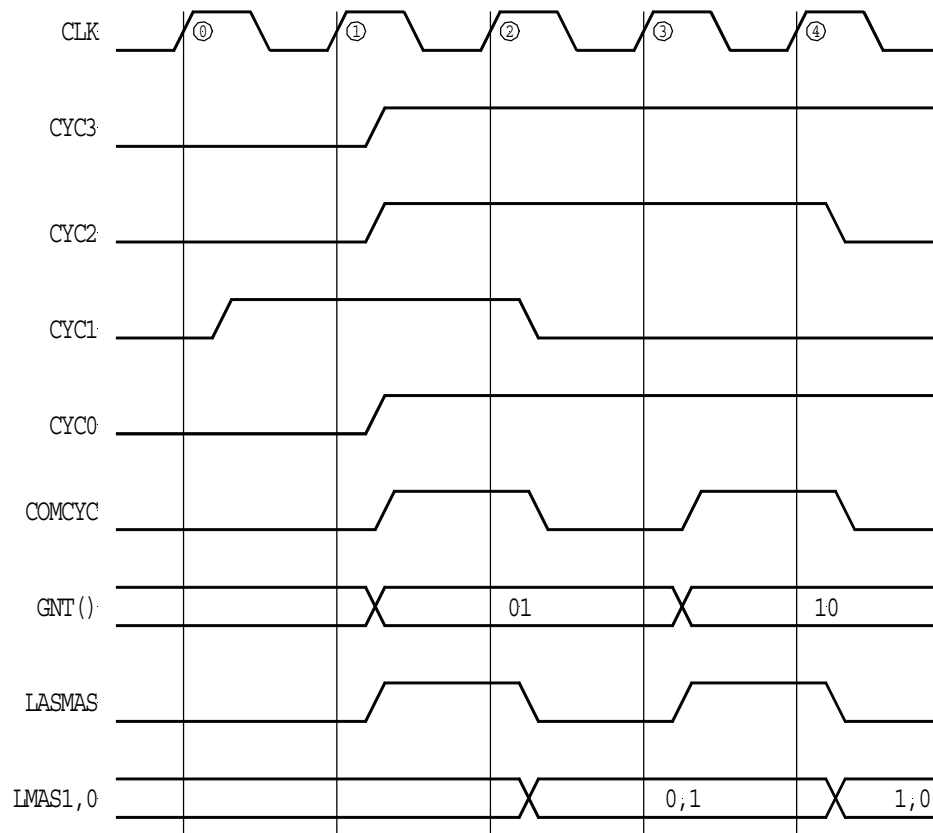


Figure 3. ARB0001a timing example.

**DMA0001a: Simple 32-bit DMA**

‘DMA0001a’ is a simple 32-bit DMA unit. It operates as a WISHBONE MASTER, and transfers using two methods: as SINGLE READ/WRITE cycles or as BLOCK READ/WRITE cycles. The type of cycle is selected by the [DMODE]<sup>2</sup> signal input. The block diagram of the DMA is shown in Figure 4, and the WISHBONE DATASHEET in Table 2. The DMA has a 5-bit addressing range.

Because of its very simple operation, this DMA is intended only for educational and benchmarking purposes. However, it can be adapted for more serious work.

When [RST\_I] is asserted, the DMA resets its control state machine and all related registers and counters. Furthermore, the initial address [IA()] and initial data [ID()] states are latched. This gives the system integrator some control over the destination address and initial write data values.

If the [DMODE] input is negated, the DMA generates SINGLE READ/WRITE cycles. In this mode, the DMA initiates a SINGLE WRITE cycle beginning at the [CLK\_I] rising edge immediately after the reset input [RST\_I] is negated. After the write cycle is completed, the DMA initiates a SINGLE READ cycle. The write / read cycles repeat indefinitely.

After each bus cycle is completed, the DMA increments its address. The highest two address bits generated by the DMA are latched in response to a reset. This allows each DMA (in a benchmarking system) to target a unique SLAVE. The lower three address bits are generated by a counter that counts from 0x0 to 0x7, and rolls over from 0x7 to 0x0.

If the [DMODE] input is asserted, the DMA generates BLOCK READ/WRITE cycles. In this mode, the DMA initiates a BLOCK WRITE cycle with eight phases. After the BLOCK WRITE cycle is completed, the DMA generates a similar BLOCK READ cycle. Timing diagrams for the BLOCK WRITE and BLOCK READ cycles are shown in Figures 5 and 6 respectively.

---

<sup>2</sup> [DMODE] is a non-WISHBONE signal.

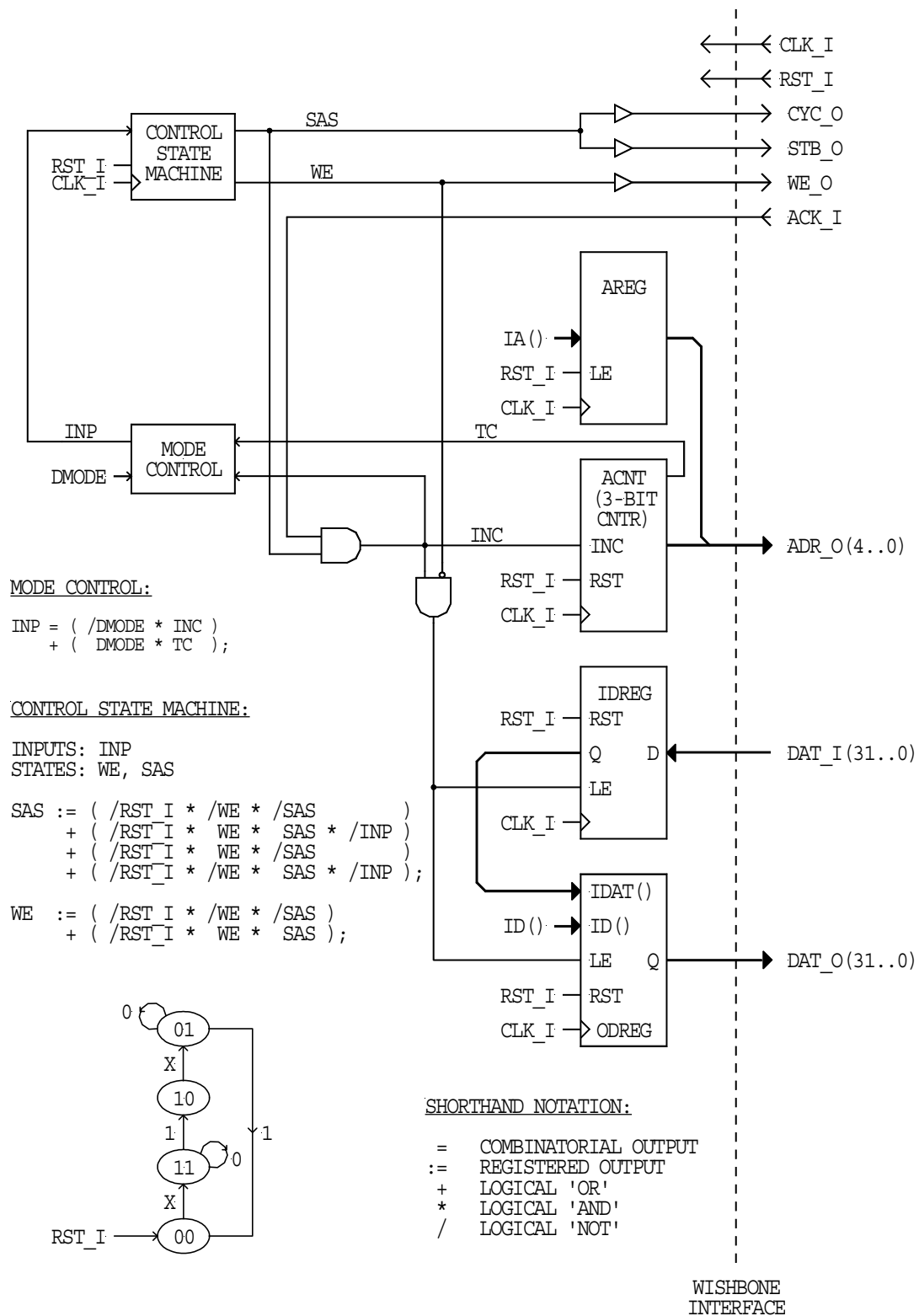


Figure 4. Block diagram for DMA0001a.

Table 2. WISHBONE DATASHEET for DMA0001a		
Description	Specification	
General description:	Simple 32-bit DMA with 5-bit addressing range. Intended for educational and benchmarking purposes.	
Supported cycles:	MASTER, SINGLE READ/WRITE MASTER, BLOCK READ/WRITE	
Data port, size:	32-bit	
Data port, granularity:	32-bit	
Data port, maximum operand size:	32-bit	
Data transfer ordering:	Big endian and/or little endian	
Data transfer sequencing:	Undefined	
Supported signal list and cross reference to equivalent WISHBONE signals:	<u>Signal Name</u>	<u>WISHBONE Equiv.</u>
	ACK_I	ACK_I
	ADR_O(4..0)	ADR_O()
	CLK_I	CLK_I
	DAT_I(31..0)	DAT_I()
	DAT_O(31..0)	DAT_O()
	RST_I	RST_I
	STB_O	STB_O
	WE_O	WE_O

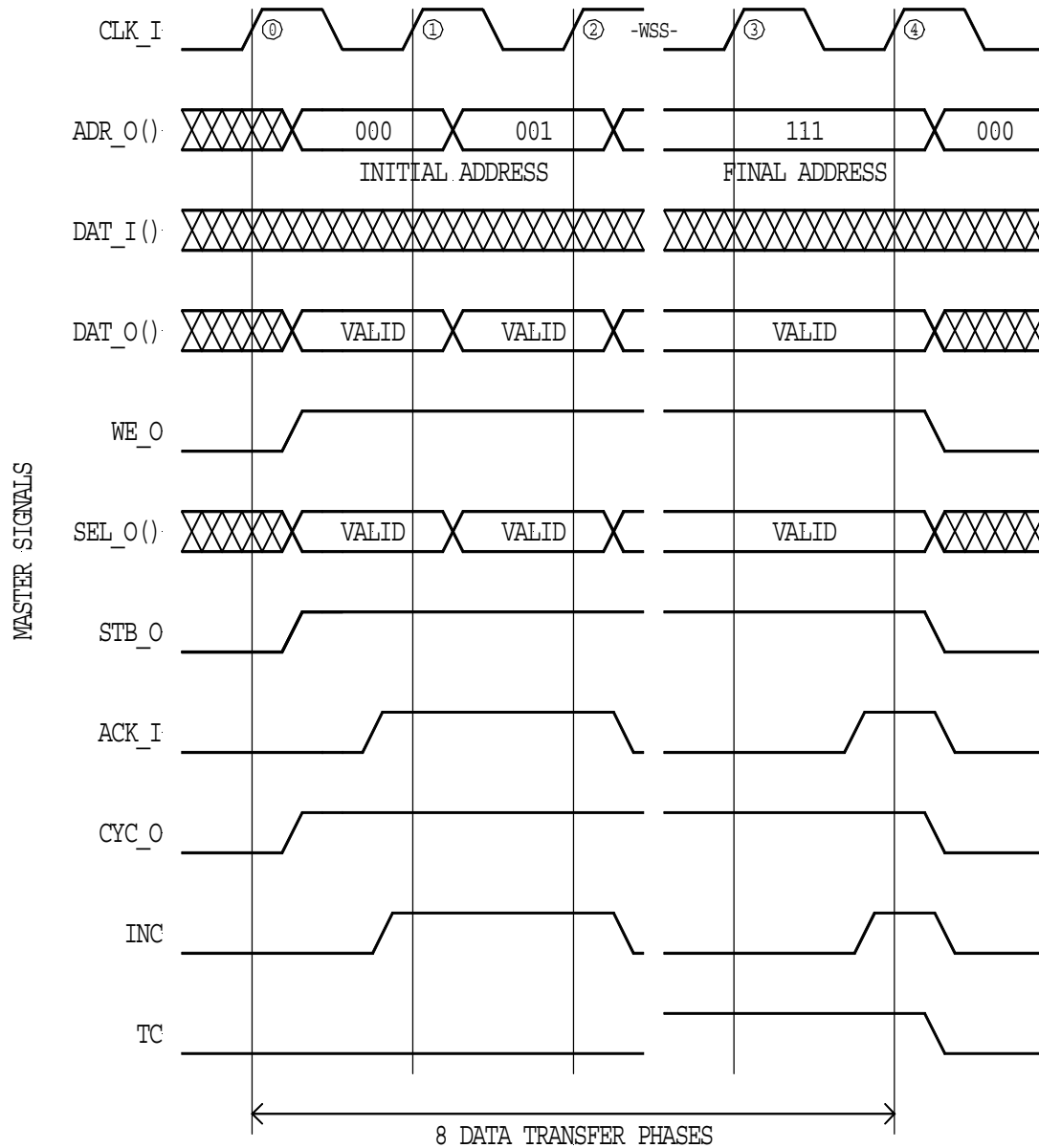


Figure 5. DMA BLOCK WRITE cycle.

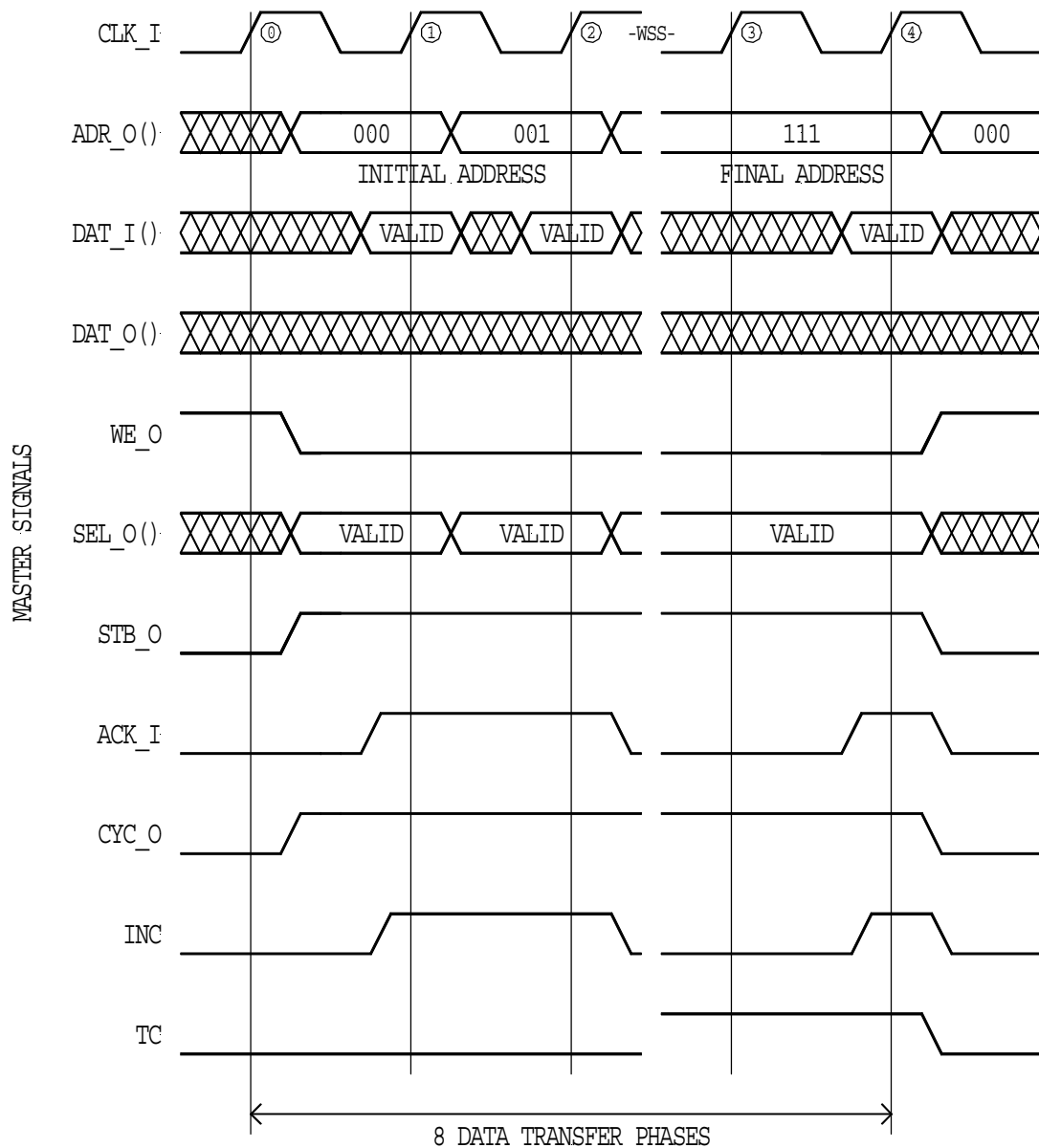


Figure 6. DMA BLOCK READ cycle.

## ICN0001a: Point-to-point Interconnection

‘ICN0001a’ is a point-to-point WISHBONE interconnection. As shown in the block diagram of Figure 7, it has a single MASTER, a single SLAVE and a SYSCON interface. These work together to form a system with a 32-bit data bus and a three-bit address bus.

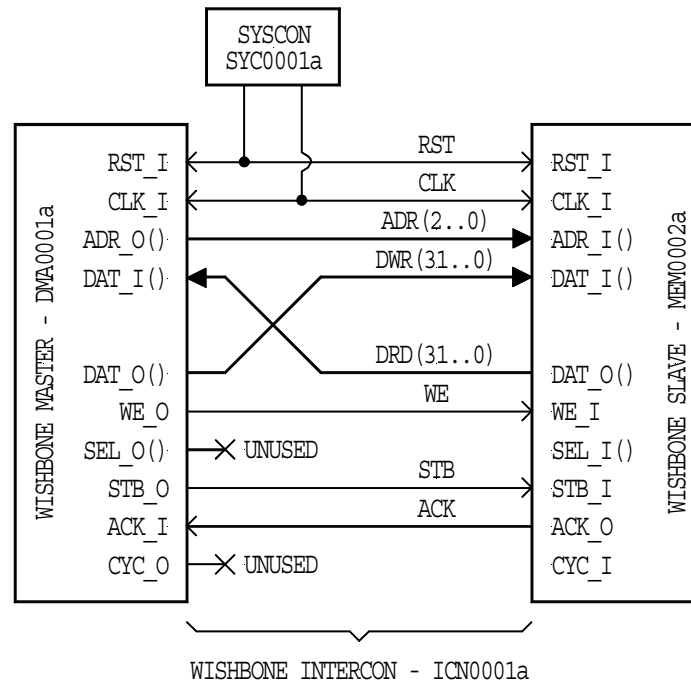


Figure 7. Block diagram of ICN0001a.

This interconnection is used for evaluation and benchmarking purposes. The MASTER interface is connected to direct memory access unit ‘DMA0001a’. The SLAVE is an 8 x 32-bit memory. For simulation purposes the ‘MEM0002a’ memory can be used. That memory is portable across all FPGA and ASIC target devices. However, when implementing this system on a specific target device, it is recommended that the device specific memory be used. For example, if this system were implemented on a Xilinx Spartan 2 or FPGA, then modify the file for use with ‘MEM0001a’. Figure 8 shows the hierarchy diagram for simulating the system.

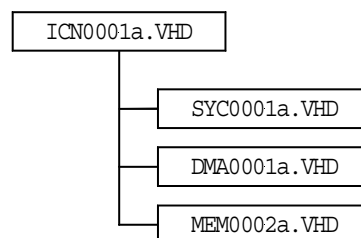


Figure 8. Hierarchy diagram for simulating the ICN0001a.

## ICN0002a: 4 x 4 Shared Bus Interconnection

'ICN0002A' is a shared bus interconnection with 4 MASTERS and 4 SLAVES. It uses multiplexor style interconnections as shown in the (generic) block diagram of Figure 9. It also has a four level round-robin arbiter.

This interconnection is used for evaluation and benchmarking purposes. All four MASTER interfaces are connected to direct memory access units of type 'DMA0001a'. The SLAVE is an 8 x 32-bit memory. For simulation purposes the 'MEM0002a' memory can be used. That memory is portable across all FPGA and ASIC target devices. However, when implementing this system on a target device it is recommended that a device specific memory be used. For example, if this system were implemented on a Xilinx Spartan 2, then modify the file for use with 'MEM0001a'. Figure 10 shows the hierarchy diagram for simulating the system.

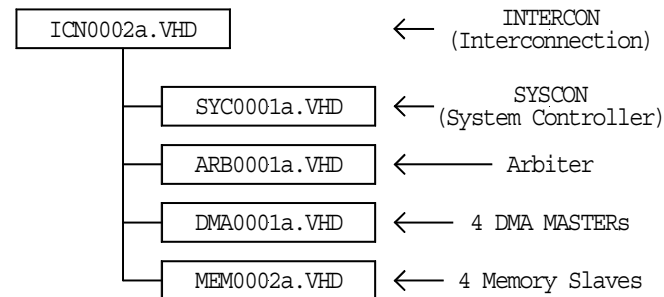


Figure 10. Hierarchy diagram for simulating the ICN0002a.

Some of the features of the interconnection have been eliminated because they were not needed. These include:

- The (optional) retry [RTY] and bus error [ERR] signals were eliminated because they were not needed.
- The (optional) [SEL] interconnections have been eliminated. This is because all of the MASTERS and SLAVES have identical port sizes and granularities. [They could have been left in, but the synthesis and router tools would have ripped them out anyway].

The system includes a partial address decoder for the SLAVES. This is shown as 'ADDRESS COMPARATOR' in the block diagram, and creates a system address space as shown in Figure 11. For the benchmark tests, each MASTER accesses the regions shown in Table 3.

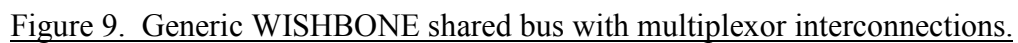


0x1F	SLAVE 3
0x18	
0x17	SLAVE 2
0x10	
0x0F	SLAVE 1
0x08	
0x07	SLAVE 0
0x00	

Figure 11. Address map used by ICN0002a.

Table 3. Address spaces decoded by ICN0002a.			
DMA Master:	DMA's To:	At Addresses	Using Cycles
MASTER #0	SLAVE #0	0x00 – 0x07	BLOCK READ/WRITE
MASTER #1	SLAVE #1	0x08 – 0x0F	BLOCK READ/WRITE
MASTER #2	SLAVE #2	0x10 – 0x17	BLOCK READ/WRITE
MASTER #3	SLAVE #3	0x18 – 0x1F	SINGLE READ/WRITE

In this application, the round-arbiter was chosen because all of the MASTERS are DMA controllers. That means that all four MASTERS continuously vie for the bus. If a priority arbiter were used, then only the one or two highest priority MASTERS would ever get the bus.



## MEM0001a: 8 x 32-bit Memory Interface for Xilinx

'MEM0001a' is a simple, 8 X 32-bit WISHBONE memory interface for Xilinx FPGA. As shown in Figure 12, this file is actually a 'wrapper' that interfaces the Xilinx RAM to a WISHBONE SLAVE interface. Table 4 shows the WISHBONE DATASHEET for the module. The interface has been tested with Xilinx Spartan 2 and Virtex 2 products, but will probably work on other devices and brands as well.

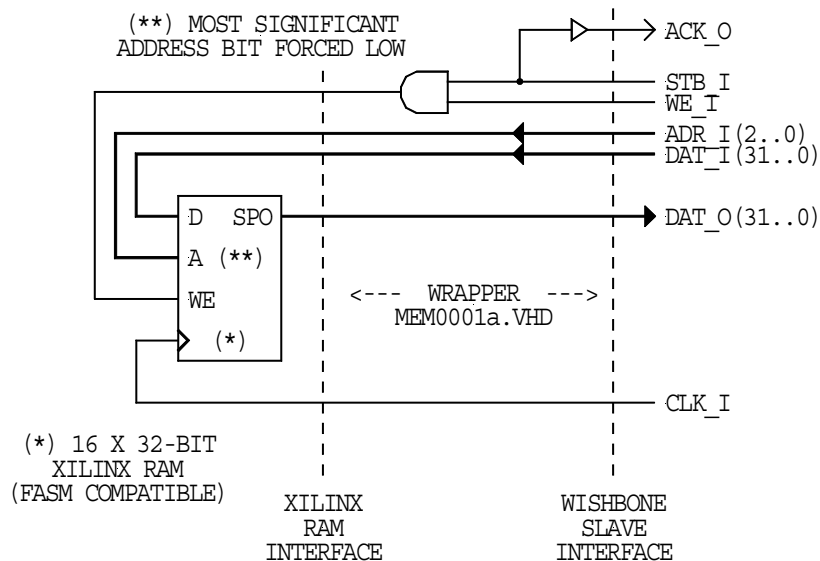


Figure 12. Block diagram of MEM0001a.

Since Xilinx copyrights their memory models, they cannot be shared in a public domain library. For this reason only the wrapper is included in the file. The RAM element itself must be created with the Xilinx Core Generator tool (which is supplied with the Xilinx Alliance router). This is a parametric core generator that creates memory elements to the users' specification.

'MEM0002a' is a similar, register based memory that can be used for simulation purposes. That memory can be used on all brands of FPGA and ASIC.

Once synthesized, the 'MEM0001a.VHD' wrapper and Xilinx RAM looks like a memory block sitting on WISHBONE. As can be seen in the Figure, the overhead glue logic needed by the wrapper is very small. Most of file is signal routing and name changes, with only a single 'AND' gate needed to operate the write enable [WE] signal.

The Xilinx Core Generator is set up to create a RAM that is an 8 x 32-bit synchronous memory. It conforms to the FASM<sup>3</sup> guidelines described in the Appendix of the WISHBONE specification. There are many ways to build memories, but the simplest is to use the automatic memory generation software that is supplied with the Xilinx router.

The RAM is formed from the Xilinx distributed RAM, meaning that the FPGA logic look up tables (LUTs) are re-configured as RAM. This is opposed to block memory, which is formed from dedicated memory cells on the FPGA. Using the Xilinx CORE Generator tool, create the distributed RAM thusly:

- 1) Create a directory called 'RAM08X32'.
- 2) Open the Xilinx CORE Generator tool, and set it up to create a synchronous RAM in the RAM08X32 folder. Set up the options thusly:

Device type: Spartan 2  
File format: VHDL  
Tool type: Other (or enter the name of your synthesis tool)  
Netlist bus format: B(I) (if required)  
Memory type: distributed  
Component name: ram08x32  
Depth: 16  
Data width: 32  
Memory type: single port RAM  
MUX construction: LUT based  
Input options: non-registered  
Layout: create RPM

- 3) Generate the RAM.
- 4) Verify that the RAM08X32 directory that you created has a file named 'ram08x32.edn' in it. This is the EDIF file for the RAM. This file will be used later by VHDL synthesis tools to create the SoC.

Also note that this example shows the procedure for Spartan 2 FPGA. Other devices, such as Virtex 2, can be made in a similar manner.

It should be also noted that Xilinx does not allow the creation of RAM memories smaller than 16 x 32-bit. For this reason, the most significant address bit of the RAM is tied low. This disables one bit, and allows the formation of the smaller memory element.

---

<sup>3</sup> FASM: FPGA and ASIC Subset Model.

Table 4. WISHBONE DATASHEET for MEM0001a	
Description	Specification
General description:	FASM compatible synchronous RAM. A wrapper for Xilinx and similar FPGAs.
Supported cycles:	SLAVE, READ/WRITE SLAVE, BLOCK READ/WRITE SLAVE, RMW
Data port, size:	32-bit
Data port, granularity:	32-bit
Data port, maximum operand size:	32-bit
Data transfer ordering:	Big endian and/or little endian
Data transfer sequencing:	Undefined
Supported signal list and cross reference to equivalent WISHBONE signals:	<u>Signal Name</u>
	ACK_O
	ADR_I(2..0)
	CLK_I
	DAT_I(31..0)
	DAT_O(31..0)
	STB_I
	WE_I
	<u>WISHBONE Equiv.</u>
	ACK_O
	ADR_I()
	CLK_I
	DAT_I()
	DAT_O()
	STB_I
	WE_I

### **MEM0002a: 8 x 32-bit Register Based Memory**

‘MEM0002a’ is a special register based memory that facilitates the testing of WISHBONE interconnections. Although this memory can be synthesized in hardware, it is generally not recommended because it will synthesize as discrete flip-flops, multiplexors and other logic. However, if slower speed and larger size are acceptable, then it is perfectly okay to use this memory.

This memory is intended to be used for the portable simulation of WISHBONE interconnections. It is portable across all FPGA and ASIC target devices. However, it is recommended that a target specific memory be used before implementation on a final target device. For example, ‘MEM0001a’ is functionally identical to this memory, except that it is optimized for use with Xilinx and similar FPGA parts.

This memory operates as a FASM compatible synchronous RAM.

