# Interfacing the Motorola Coldfire 5307 to the V320USC

Application Note

## 1.1 Objective

This application note shows the interface of the MCF5307 Coldfire processor to the PCI bus using the V320USC Universal System Controller. Basic familiarity with these devices is assumed.

This application note should be used in conjunction with the V320USC and the Coldfire 5307 User's Manuals. If you don't have these documents, you can download them from the following web sites:

| | |
|---|---|
| The V320USC User's Manual: | http://www.quicklogic.com |
| The Motorola Coldfire 5307 User's Manual: | http://www.mot.com |

## 1.2 Overview

This application note focuses on the conversion of the Motorola protocol to a MIPS® protocol, and the conversion of a modulo four-burst protocol to a linear-burst protocol. These conversions are accomplished using an external glue logic PLD (Programmable Logic Device). The PLD takes a Motorola de-multiplexed 32-bit protocol and converts it to a multiplexed MIPS System Interface. Bus switches are used to multiplex the Coldfire address and data lines to the MIPS SYSAD bus. After the conversion to a MIPS protocol, the Coldfire has access to the PCI bus, the M bus, and the peripheral bus on the V320USC. The PLD uses the control signals from the Coldfire and generates the proper SysCmd to initiate an access to the PCI bus or the M bus.

QuickLogic Canada Corporation
250 Consumers Rd., Suite 901
Toronto, Ontario, Canada
Toll Free: 1-877-283-7364
International: 416-497-8884
Fax: 416-497-1160

Using the V320USC as a PCI interface for the Coldfire processor has the following advantages:

- It allows the PCI transactions to/from the SDRAM to achieve transfer speeds up to 132 Mbytes/sec.
- It allows both the local and PCI bus to be used at the Coldfire's maximum local bus speed of 45 MHz, resulting in a higher performance system.

Any peripherals connected to the Coldfire local bus can now be directly connected to the peripheral bus on the V320USC. This takes the load off the Coldfire local bus. In addition, the V320USC supports $I^2C$—this feature makes peripheral connections to the V320USC even more attractive.

## 1.3 V320USC Overview

### 1.3.1 V320USC Processor Interface

When used in MIPS mode, the USC provides a three bus architecture as follows:

- A **PCI Bus,** as defined by the PCI specification.
- A **Local Bus** that employs the protocol of the MIPS processors.
- An **M Bus** — memory and peripheral bus.

Being a private memory bus, the USC provides an asymmetrical bridging operation (see **Figure 1-1)**. Typically, the SDRAM is the main point of intersection between the PCI host and the local CPU. Consequently, both the PCI and the local buses can access the SDRAM and the peripherals on the M bus. The local processor can also access the PCI bus as a bus master; however, external PCI agents cannot generate local bus cycles to a slave device on the local bus.

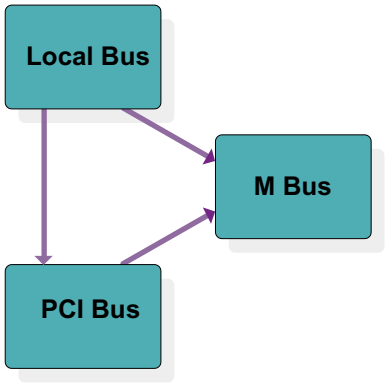


*Figure 1-1: Bus Mastering Relationship—3 Bus Mode (MIPS)*

## 1.3.2 Internal Operation

The USC has five major functional blocks: local processor interface, M bus interface, PCI interface, DMA controller, and internal registers. The three bus interfaces are interconnected by point-to-point highways buffered by FIFOs. **Figure 1-2** presents the internal architecture of the V320USC in MIPS processor mode.

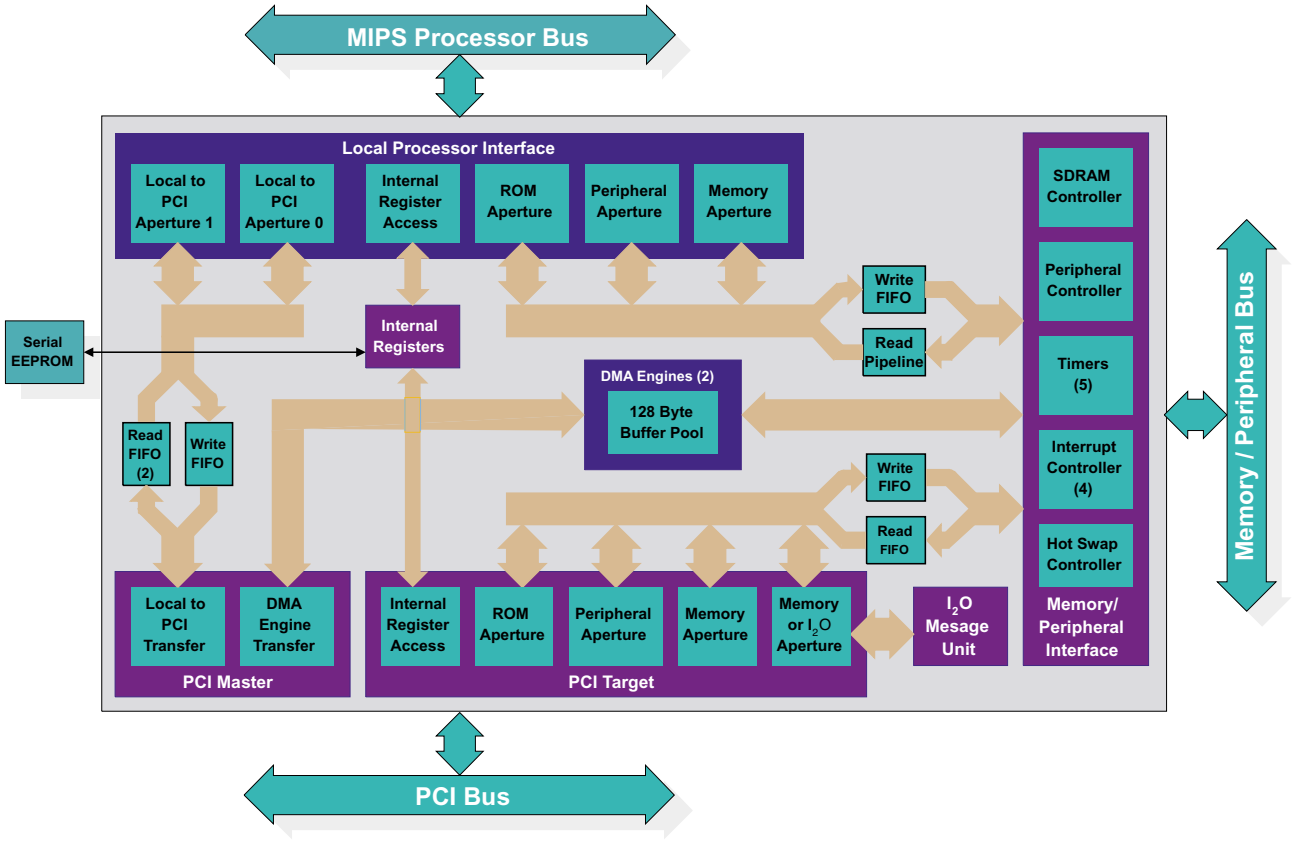


*Figure 1-2: V320USC Internal Block Diagram (MIPS Mode)*

### 1.3.2.1 Access Apertures

The local and PCI buses have a number of apertures that capture addresses placed on the bus by a bus master device. For example, the local bus has some apertures that allow access to the internal registers of the USC. **Figure 1-3** illustrates these apertures and how they are related to the M bus and the internal registers. The address, and, in many cases, the size of these apertures is programmable.

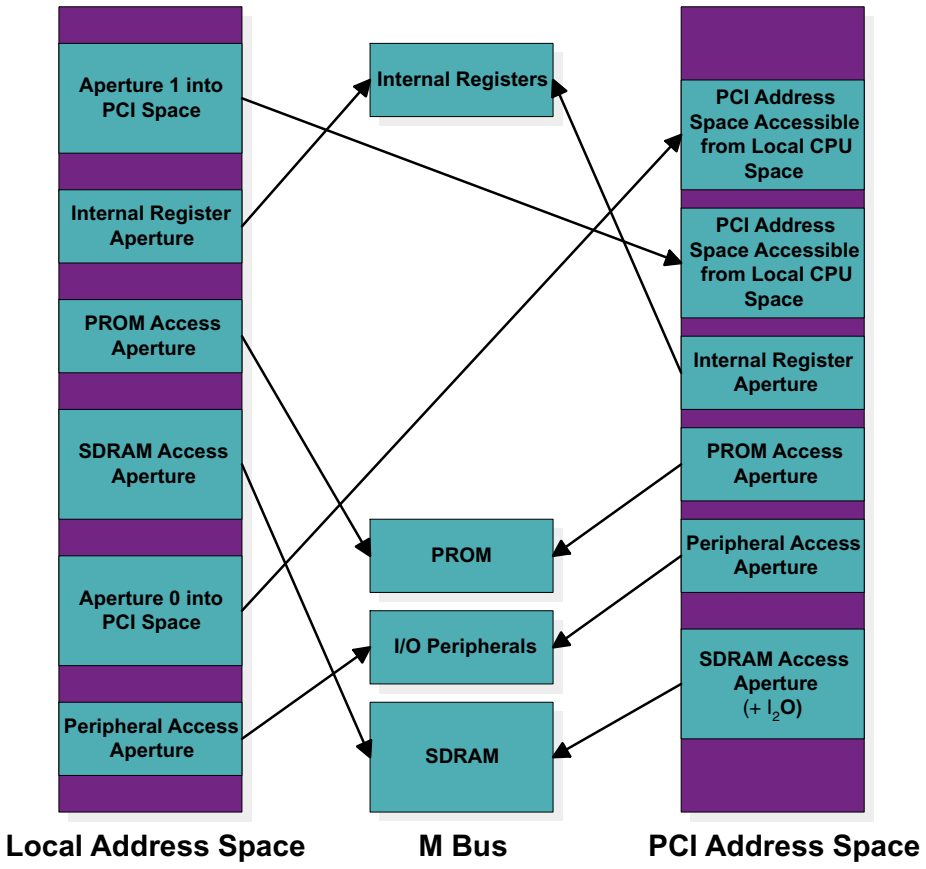


*Figure 1-3: Bridging between PCI, Local, and M buses*

### 1.3.2.2 Address Translation

Address translation is the process of mapping one address space to another. This feature is very important in systems where the mapping of the PCI space cannot be controlled—as in the case of PCI add-in cards. In such cases, address translation allows convenient *holes* in local space to be mapped to any arbitrary PCI address range.

#### 1.3.2.3 PCI Bus Interface

Providing a direct connection with the industry standard PCI, the V320USC can act either as a master initiating a PCI bus operation, or as a target responding to a PCI bus operation. 288-bytes of posted write and read prefetch buffers are provided for efficient data transfer between the CPU bus and the PCI.

As a PCI bus master, the USC supports all currently defined bus cycles with the exception of the dual address cycle.

As a target, the USC can be programmed to respond to either I/O or memory cycles through standard PCI-defined Base Address Registers. In the case of internal registers, the USC also responds to configuration cycles. All internal registers, including the standard PCI configuration registers, can be accessed from both the local bus and the PCI bus. The USC configuration register set is PC plug-and-play compatible.

#### 1.3.2.4 Local Bus Interface

The USC's local bus is designed to connect directly to 32-bit members of the MIPS processor family, or any other local bus device that can emulate the bus interface. The devices on the local bus can access the PCI bus, the memory, or the peripherals on the M bus. The MIPS bus protocol supports 8-, 16-, 24-, and 32-bit word operations with burst lengths up to eight words. With a maximum frequency of 75 MHz, the CPU can transfer up to 264 Mbytes/sec.

#### 1.3.2.5 Memory and Peripheral Bus Interface

Having a separate memory/peripheral bus, the USC can optimize the local processor memory performance by implementing a 16 word write buffer. This way, the $\overline{\text{WRRDY}}$ remains asserted most of the time and the CPU writes are absorbed by the FIFO at the full bus speed of the processor (zero wait state). On the M bus, SDRAM bursts are performed at zero wait states and peripheral devices are written at a rate determined by a set of peripheral control registers.

The peripheral controller functions support a variety of memory and I/O devices by providing fine grain programmability of timing control signals. Consequently, devices such as flash, EEPROMs, SRAMs, FIFOs, and I/O controllers are easily accommodated. Device widths from 8-bits to 32-bits are directly supported.

#### 1.3.2.6 On-Chip Memory Buffers

The USC contains a number of on-chip FIFO buffers. These buffers are designed to allow simultaneous fill and drain so that long back-to-back bursts can be sustained.

**Figure 1-4** shows how the on-chip buffer memory is allocated.

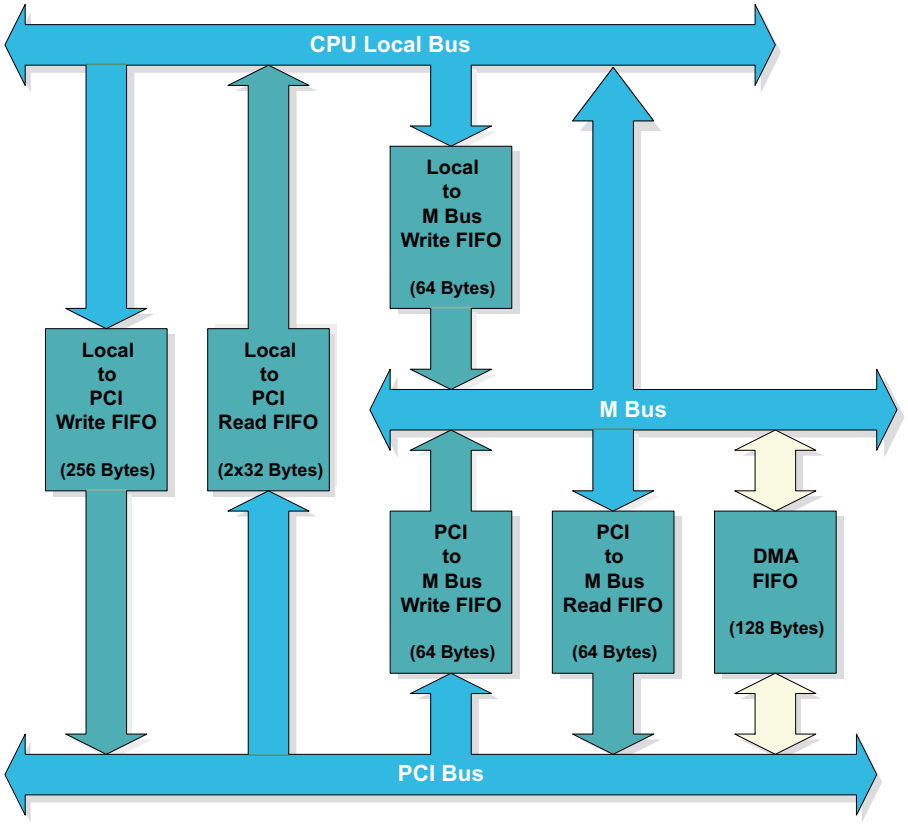


*Figure 1-4: Data Buffering between Various Buses*

### 1.3.2.7 DMA Controller

The USC houses two independently controlled DMA engines[1]. These engines contain a number of advanced features such as block transfers, shuttle mode, and DMA chaining. The architecture supports simultaneous fill and drain from/to the M bus to/from the PCI bus to allow long burst transactions to occur. This maximizes the PCI bus utilization.

The following list presents the advanced features available for the V320USC DMA engines:

- Anywhere-to-anywhere DMA: (PCI, SDRAM, peripheral) to (PCI, SDRAM, peripheral).
- Descriptor based scatter/gather capability.
- Descriptor write-back to indicate completion of an individual descriptor link and to allow looped descriptors.
- Block fill.
- Interrupt on End-of-Transfer (EOT) or End-of-Link (EOL).
- Programmable byte lane swapping.
- Burst loading of descriptors.

1. Not available with revision A0 silicon.

## 1.4 Motorola Coldfire 5307 Overview

This section covers information about the Motorola Coldfire that is relevant for the purpose of this application note[2]. We will discuss how transfer size, bursts, and burst starting addresses are determined in the Motorola protocol.

The burst functionality of the MCF5307 is similar to that of the M680x0 series: the processor asserts SIZ[1:0] at the beginning of a bus cycle to indicate the transfer length (see **Table 1-1**).

Table 1-1: MCF5307 Transfer Size

| SIZ[1:0] | Port Size | Transfer Size |
|---|---|---|
| 00 | Longword | Single Cycle |
| 01 | Byte | Single Cycle |
| 10 | Word | Single Cycle |
| 11 | Line | Burst of 4 |

Please note that the MCF5307 wraps a burst on a four word modulo boundary. That means that if a burst of four words begins on an address in which bits A[3:2] are non-zero, the burst address goes from 0xnnnnnnnC back to 0xnnnnnnn0 where nnnnnnn are the upper 28 bits of the address. The approach used in this application note is to break the burst cycles that start with A[3:2] = '01' or '11'. When the value of SIZ[1:0] = '11', the value of A[3:2] determines where in the cycle the burst of four words starts and ends (see **Table 1-2**)

Table 1-2: Four Modulo Boundary Burst

| SIZ[1:0] | Addr[3:2] | Longword |
|---|---|---|
| 11 | 00 | 0-4-8-C |
| 11 | 01 | 4-8-C-0 |
| 11 | 10 | 8-C-0-4 |
| 11 | 11 | C-0-4-8 |

2. For more information on the Coldfire 5307 processor please check the *Motorola Coldfire User's Manual*.

# 1.5 Application Note - Interfacing the Motorola Coldfire to the V320USC

This application note has two parts. The first part discusses the conversion of the Motorola protocol to a MIPS protocol, and the second part presents the conversion of a modulo four-style bursting to a linear-style bursting.

## 1.5.1 System Overview

Transactions from the Coldfire to the PCI or the M bus are done through the V320USC (see **Figure 1-5**). Coldfire transactions are converted to MIPS style protocol. Transactions from the PCI bus to the SDRAM are done directly by the V320USC.

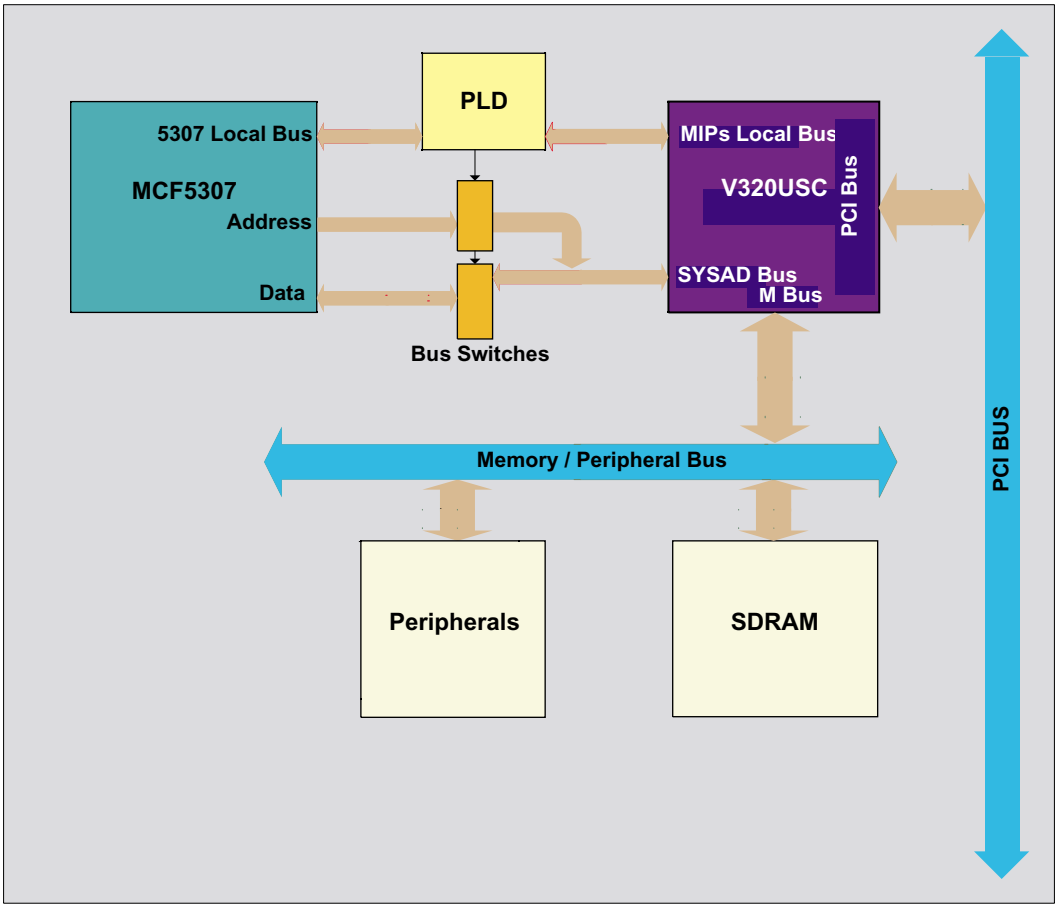


*Figure 1-5: Motorola Coldfire to V320USC Block Diagram*

The use of the V320USC allows the PCI transactions to the SDRAM to have transfer speeds up to 132 Mbytes/sec. **Figure 1-6** presents the interfaces between the V320USC and the MCF5307.

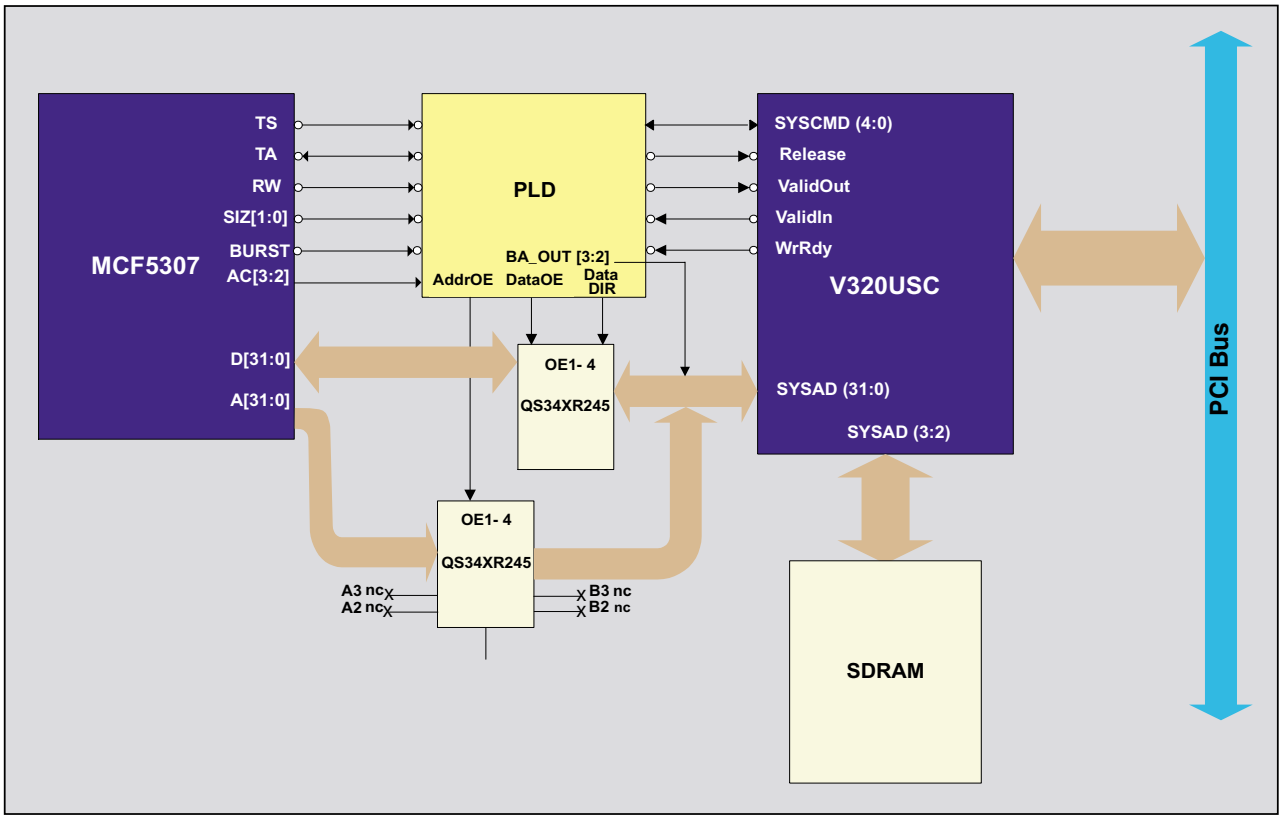


*Figure 1-6: Motorola Coldfire to V320USC Signal Interface*

## 1.5.2 Motorola to MIPS Protocol Conversion

This section discusses the conversion of modulo four-style bursting to a linear-style bursting.

To generate a SysCmd from a Motorola transaction, the PLD uses the following signals (see **Figure 1-7**):

- R/W determines if it is a Read or Write
- SIZ[1:0] determines transfer size (byte, 2 bytes, 4 bytes, four-word burst)
- A[3:2] determines the starting address in a transaction

Once these signals have been sampled, the corresponding SysCmd is generated for the address and data phase of the transaction. For example, a Motorola single-word write transaction generates the following SysCmd in the address phase: '01011' (0x0B) and the following SysCmd in the data phase: '10001' (0x11)[3].

| | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| Singe Read Command | 0 | 0 | 0 | 0=1 Byte 1=2 Bytes 2=3 Bytes 3=4 Bytes | |
| Burst Read Command | 0 | 0 | 1 | 0=2 words 1=4 words 2=8 words 3=reserved | |
| Singe Write Command | 0 | 1 | 0 | 0=1 Byte 1=2 Bytes 2=3 Bytes 3=4 Bytes | |
| Burst Write Command | 0 | 1 | 1 | 0=2 words 1=4 words 2=8 words 3=reserved | |
| Processor Data Identifier | 1 | not last | 0 | data error | disable data check |
| External Data Identifier | 1 | not last | not resp. data | data error | disable data check |

*Figure 1-7: MIPs SysCmd Encoding (5 bit)*

## 1.5.3 Burst Conversion

In response to a Motorola initiated burst, the PLD initiates two different types of bursts. The first burst is a linear burst of four words; the second burst is broken into intermediate cycles. A broken burst consists of three intermediate cycles: a single cycle, a burst of two words, and a single cycle (see **Table 1-3**). The value of A[3:2] determines the type of access the USC executes on the PCI bus or M bus.

Table 1-3: Single Cycle Broken Burst

| A[3:2] | Longword Access | Cycle of Access on PCI Bus |
|---|---|---|
| 00 | 0-4-8-C | Burst of 4 words |
| 01 | 4-8-C-0 | Single Word - Burst of 2 - Single Word |
| 10 | 8-C-0-4 | Burst of 4 words |
| 11 | C-0-4-8 | Single Word - Burst of 2 - Single Word |

The MCF5307's bursts are always four words long; however, the V320USC sees the MCF5307 burst as a burst of four words only when:

- the burst is aligned so that A[3:2] = '00'
- the USC performs a toggle burst so that A[3:2] = '10'

3. Refer to the *Read and Write Waveforms* section of this document for different examples of SysCmd generations.

The V320USC sees the MCF5307 burst as a burst of four words due to the fact that the V320USC is designed to be glueless to the MIPS processor, and the MIPS processor can perform toggle bursts when in cache mode. **Figure 1-8** shows how the toggle burst mechanism works. **Table 1-4** shows how the V320USC interprets A[3:2] from the Coldfire processor and how it converts a half-word aligned burst into an aligned burst.

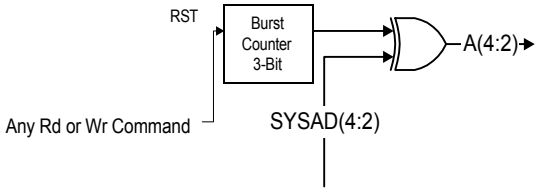


*Figure 1-8: Internal Toggle Burst Mechanism*

Table 1-4: V320USC Interpretation of the Coldfire A[3:2]

| Coldfire A[3:2] SYSAD[3:2] | USC A[3:2] |
|---|---|
| 10 | 00 |
| 11 | 01 |
| 00 | 10 |
| 01 | 11 |

The signals used for breaking the Motorola burst are the BREAK_BURST, LAST, and SYS_BURST signals. The BREAK_BURST signal indicates that an intermediate cycle should end and a new intermediate cycle should start. The SYS_BURST signal represents the end of a burst of two words. The LAST signal is the signal used to end the entire cycle.

Burst generation is accomplished through generating the signal *LAST* at the end of a burst. LAST is always asserted for non-burst access and SIZ[1:0]= '11'. If the LAST signal is not asserted, it crosses a modulo four boundary and the cycle starts again. When the LAST signal crosses a modulo four boundary, the PLD breaks the burst into three intermediate cycles.

**Table 1-5** lists the sequences and cycles involved in generating a LAST signal at the end of a burst.

Table 1-5: Generating the LAST signal at the end of a Burst

| Start Address | Sequence | Cycle 0 | Cycle 1 | Cycle2 | Cycle 3 |
|---|---|---|---|---|---|
| 00 | Burst Address Sequence | 00 | 01 | 10 | 11 |
| | LAST Sequence | 0 | 0 | 0 | 1 |
| | SYS_BURST Sequence | 1 | 1 | 1 | 0 |
| | BREAK_BURST Sequence | 0 | 0 | 0 | 0 |
| 01 | Burst Address Sequence | 01 | 10 | 11 | 00 |
| | LAST Sequence | 0 | 0 | 0 | 1 |
| | SYS_BURST Sequence | 0 | 1 | 0 | 0 |
| | BREAK_BURST Sequence | 1 | 0 | 1 | 0 |
| 10 | Burst Address Sequence | 10 | 11 | 00 | 01 |
| | LAST Sequence | 0 | 0 | 0 | 1 |
| | SYS_BURST Sequence | 1 | 1 | 1 | 0 |
| | BREAK_BURST Sequence | 0 | 0 | 0 | 0 |
| 11 | Burst Address Sequence | 11 | 00 | 01 | 10 |
| | LAST Sequence | 0 | 0 | 0 | 1 |
| | SYS_BURST Sequence | 0 | 1 | 0 | 0 |
| | BREAK_BURST Sequence | 1 | 0 | 1 | 0 |

## 1.5.4 PLD Overview

For this application note we have used the Altera's Max+ plus II software and the EPM7032AELC44-5 PLD[4]. However, only high speed grade FPGA will suffice.

**Figure 1-9** presents descriptions of PLD signals

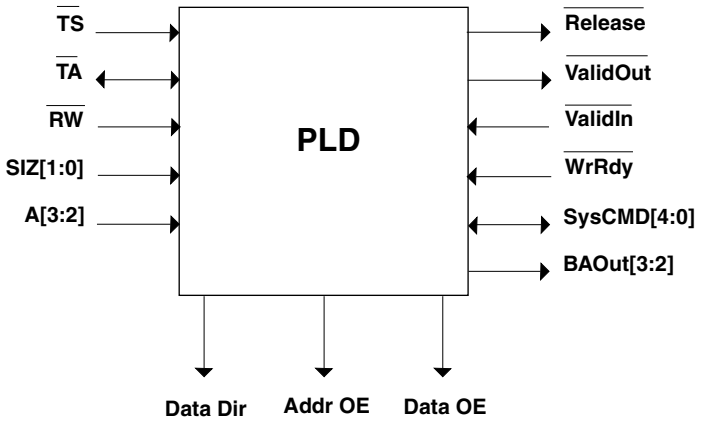


*Figure 1-9: PLD Signals*

4. You can buy this part from a local distributor.

**Table 1-6** presents descriptions of PLD signals.

Table 1-6: PLD Signal Descriptions

| Signal | Device | Definition | Direction | Description |
|---|---|---|---|---|
| TS | MOT | Transfer Start | Input | Indicates the beginning of a bus transfer |
| TA | MOT | Transfer Acknowledge | In/Out | Assertion terminates transfer synchronously |
| R/W | MOT | Read/Write | Input | Identifies read and write transfers |
| SIZ[1:0] | MOT | Size | Input | Indicates data transfer size |
| A[3:2] | MOT | Address | Input | Shows the start address of a transfer |
| Release | USC | Release Interface | Output | Indicates when the USC should command data on the system bus in response |
| ValidOut | USC | Valid Output | Output | Used by the USC to qualify all commands and data from the system bus |
| ValidIn | USC | Valid Input | Input | Driven by the USC in response to a Release during a read cycle to indicate read data is ready |
| WrRdy | USC | Write Ready | Input | Driven by the USC when it cannot accept any additional write commands |
| SysCMD[4:0] | USC | System Command | In/Out | A 5 bit bus for command and data identifier transmission used by the USC to initiate cycles on to the PCI or M Bus. |
| Data Dir | Bus Switches | Data Direction | Output | Indicates direction of data for reads or writes |
| Data OE | Bus Switches | Data Output Enable | Output | Enable data |
| Address OE | Bus Switches | Address Output Enable | Output | Enable address |
| BA_Out [3:2] | USC | Burst Address [3:2] | Output | Driven by the PLD during the address phase as SysAD[3:2] |

**Figure 1-10** is the state diagram of the state machine present in the PLD. It starts with the assertion of $\overline{TS}$. At the same time, SIZ[1:0] = '01' and A[3:2] = '01'.These registers generate a SysCmd for a single byte write, starting at the address '01' (0x04). During the assertion of $\overline{TS}$, DATA_OE is de-asserted and ADDRESS_OE is asserted so that the address lines from the Coldfire will assert on the SYSAD bus. On the next cycle, a SysCmd for the data phase is generated. During this data phase, ADDRESS_OE is de-asserted and DATA_OE is asserted to allow data from the Coldfire to appear on the SYSAD bus.

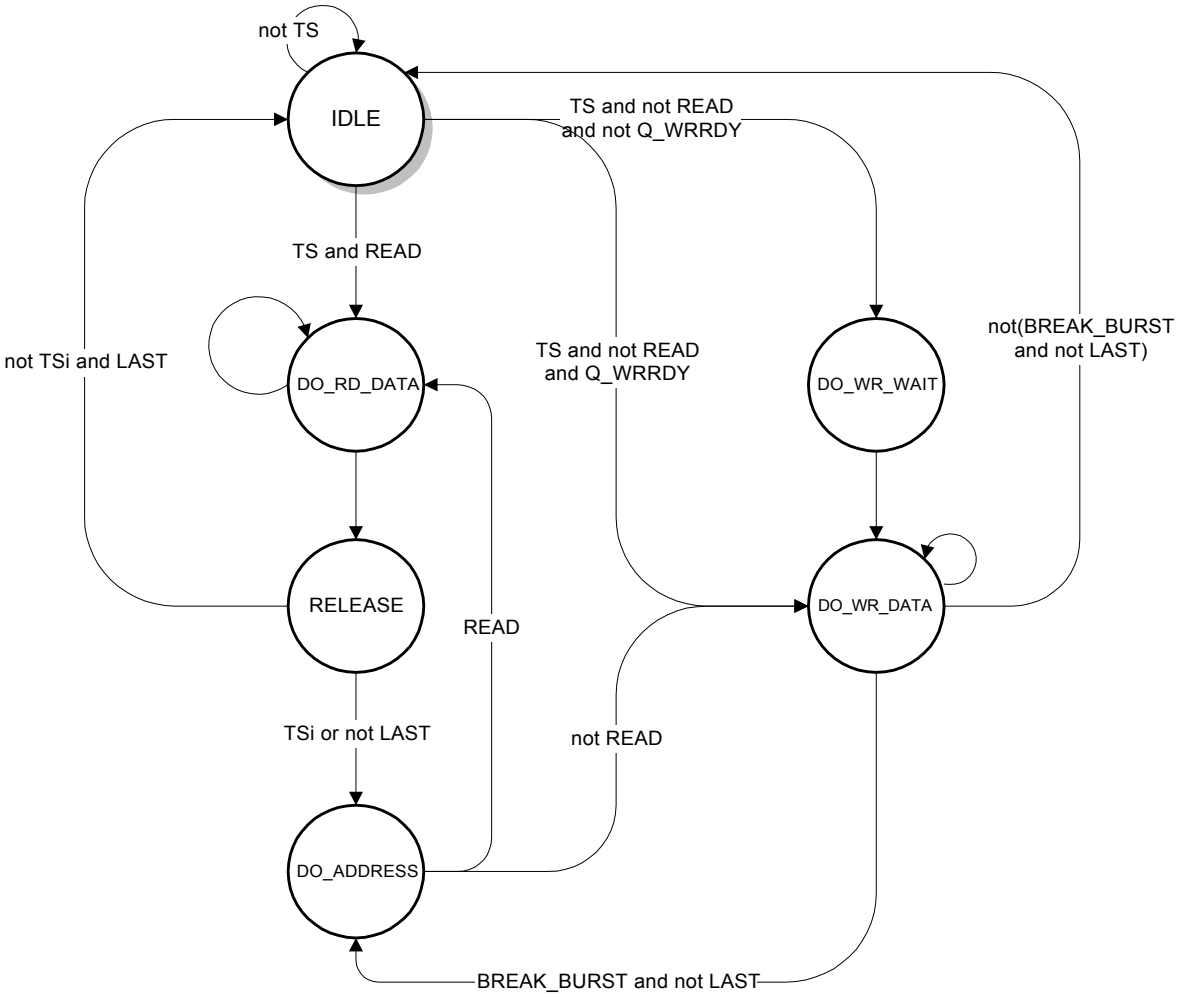


*Figure 1-10: State Diagram for PLD*

### 1.5.5 Read and Write Waveforms

**Figure 1-11** starts with the assertion of $\overline{TS}$. At the same time, SIZ[1:0] = '01' and A[3:2] = '01'. These registers generate a SysCmd for a single byte write, starting at the address '01' (0x04). During the assertion of $\overline{TS}$, DATA_OE is de-asserted and ADDRESS_OE is asserted so that the address lines from the Coldfire will assert on the SYSAD bus. On the next cycle, a SysCmd for the data phase is generated. During this data phase, ADDRESS_OE is de-asserted and DATA_OE is asserted to allow data from the Coldfire to appear on the SYSAD bus.

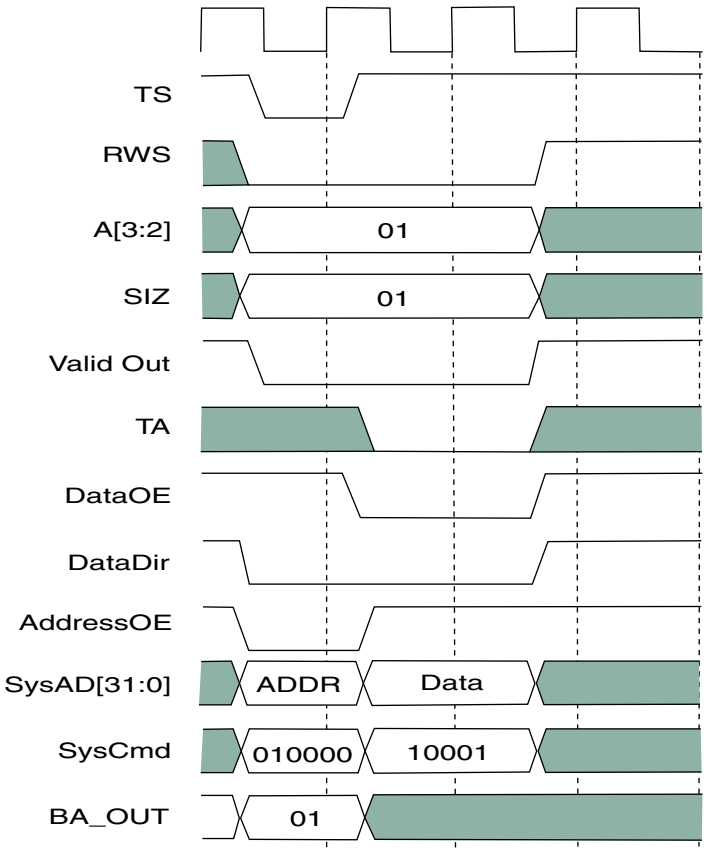


*Figure 1-11: Single Cycle Byte Write Access*

**Figure 1-12** shows a burst transaction in which SIZ[1:0] = '11'; this figure indicates that the burst is aligned and A[3:2] = '00'. Therefore, the PLD generates a four word burst SysCmd. During the data phase, the PLD generates a SysCmd to indicate that the data is being transferred. The last SysCmd indicates that this is the last data to be transferred in the burst.

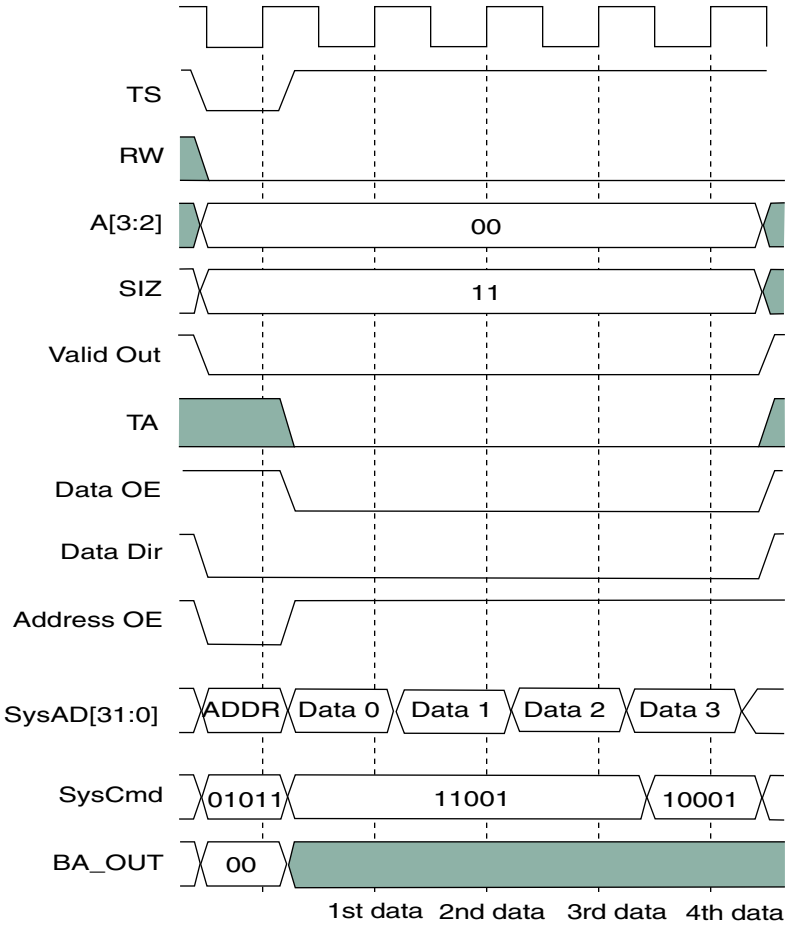


*Figure 1-12: Aligned Burst of Four Write*

In **Figure 1-13**, SIZ[1:0] = '11' and the cycle starts at the end of a modulo burst (A[3:2] ='11'). Therefore, the burst is broken up. When TS is asserted RW, A[3:2] and Siz[1:0] are sampled and the proper SysCmd is outputted. The first cycle that begins is a single cycle. On the next clock cycle, data is clocked in with the appropriate SysCmd, which is a burst of two words. This burst is followed by two cycles of data. The last cycle to be initiated is another single cycle.

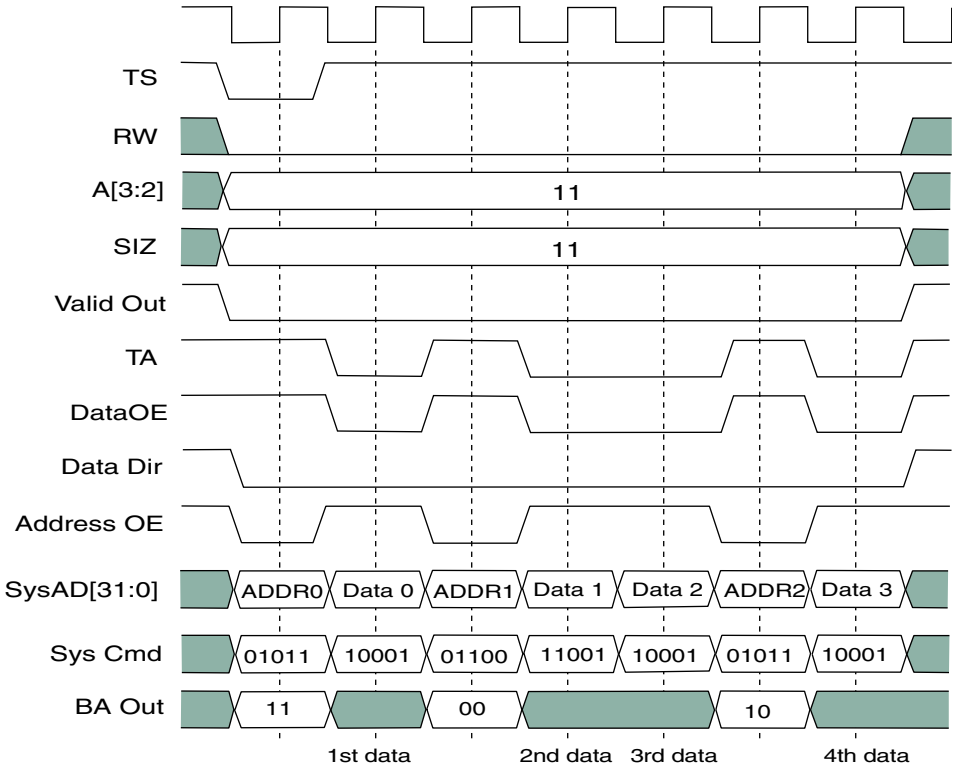


*Figure 1-13: Unaligned Burst Write*

Read Cycles are similar to the PLD write cycles except that the DATA_DIR signal is reversed to allow the data to pass to the Coldfire, and the PLD does not drive out any SysCmd's during the data cycles. **Figure 1-14** depicts these differences.

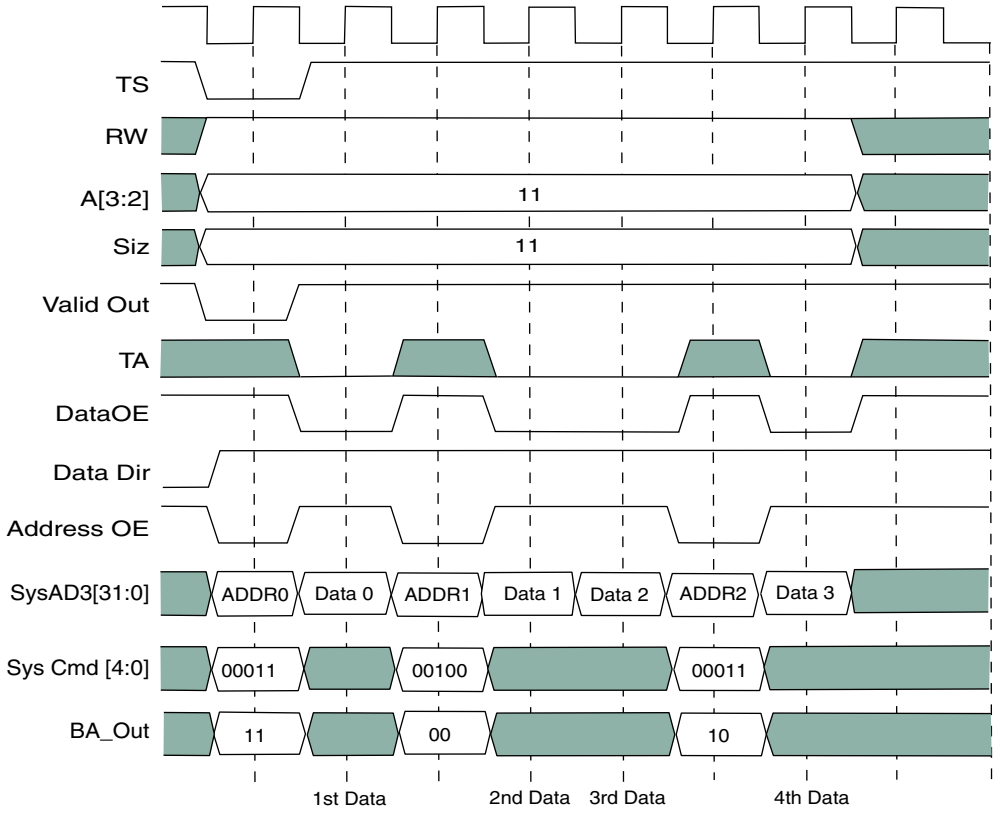


*Figure 1-14: Unaligned Burst Read*

# 1.6 PLD Source Code

The source code presented in **Section 1.6.2 , "-VHDL Code" on page 20,** was written in VHDL (Very High Speed Integrated Circuit Hardware Description Language).

## 1.6.1 VHDL Conventions

To make the VHDL code easier to follow, we are presenting some conventions we have used while writing the code:

- Signal names are all in upper case with the following exceptions: signals that are active low are indicated by a lower case 'b' at the end of the signal name (TSb, TAb, BWEb, BURSTb). All active low signals are converted to active high when used in the logic generation portion of the VHDL code.
- Top level signals pass through an I/O buffer stage as shown in **Figure 1-15.** This buffer stage converts the signals to positive true logic and converts weak input highs/lows 'H'/'L' to'1'/'0'.
- All VHDL core logic was written in positive true logic. This means that an active low signal (such as $\overline{TS}$, called TSb in VHDL) is converted to an active high equivalent (as in **Figure 1-15**), and then used by the core logic in the active high sense (TSi ='1' means $\overline{TS}$ is active).

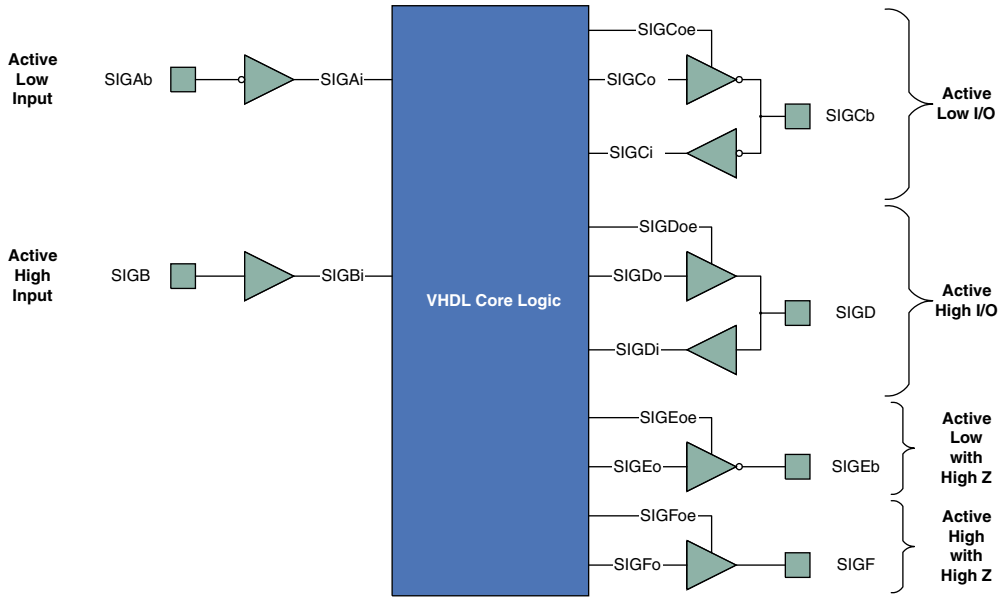


*Figure 1-15: Signal Names in Relation to I/O Buffering*

## 1.6.2 -VHDL Code

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
--=============================================================================--
-- MODULE << coldfire_usc >>
-- Description: This module interfaces a Coldfire (or other Motorola CPU)
--               to the V320USC
--=============================================================================--
entity coldfire_usc is
    port(
        CLK       : in     STD_LOGIC;
        --
        -- Main system Reset
        RESETb    : in     STD_LOGIC;
        --================================================
        -- Coldfire Signals
        --================================================
        TSb  : in     STD_LOGIC;
        TAb          : inout STD_LOGIC;
        A            : in     STD_LOGIC_VECTOR(3 downto 2);
        RW       : in     STD_LOGIC;
        SIZ          : in     STD_LOGIC_VECTOR(1 downto 0);
        --
        --================================================
        -- V320USC Signals
        --================================================
        RELEASEb : out    STD_LOGIC;
        VALIDOUTb : out    STD_LOGIC;
        VALIDINb : in     STD_LOGIC;
        WRRDYb       : in     STD_LOGIC;
        SYSCMD       : inout  STD_LOGIC_VECTOR(4 downto 0);
        --
        -- This is the version of address that is counted
--      LA   :inoutSTD_LOGIC_VECTOR(3 downto 2);
        --
        --================================================
        -- Buffer Control Signals
        --================================================
        DATA_OEb : out    STD_LOGIC;
        DATA_DIR : out    STD_LOGIC;
        ADDRESS_OEb : out    STD_LOGIC
--================================================
--External Address Signal
--================================================
BA_OUTb      :out STD_LOGIC_VECTOR(3 downto 2)
);
end coldfire_usc;
architecture BEHAVIOUR of coldfire_usc is
-----------------------------------------------------------------------
-- "+" Increment function
-----------------------------------------------------------------------
function "+"(L: STD_LOGIC_VECTOR; R: STD_LOGIC) return STD_LOGIC_VECTOR is
    variable Q: STD_LOGIC_VECTOR(L'range);
    variable A: STD_LOGIC;
    begin
        A := R;
        for i in L'low to L'high loop
            Q(i) := L(i) xor A;
            A := A and L(i);
        end loop;
        return Q;
    end;

-----------------------------------------------------------------------
-- Convert boolean to STD_ULOGIC
-----------------------------------------------------------------------
function to_mvl ( b: in boolean ) return STD_LOGIC is
    begin
        if ( b = TRUE ) then
            return ('1');
            else
            return ('0');
        end if;
            end to_mvl;

-------------------------------------------------------------------------------
-- Signal Declarations
```

```
--------------------------------------------------------------------------------
-- I/O signals for Coldfire
    signal RESETi: STD_LOGIC;
    signal TSi  : STD_LOGIC;
    signal TAi : STD_LOGIC;
    signal TAo : STD_LOGIC;
    signal TAoe : STD_LOGIC;
    signal Ai : STD_LOGIC_VECTOR(A'range);
     -- signal TTi:STD_LOGIC_VECTOR(TT' range);
    signal READ : STD_LOGIC;
    signal SIZi : STD_LOGIC_VECTOR(SIZ'range);


-- I/O signals for V320USC
    signal SYSCMDi : STD_LOGIC_VECTOR(SYSCMD'range);
    signal SYSCMDo : STD_LOGIC_VECTOR(SYSCMD'range);
    signal SYSCMDoe : STD_LOGIC;
    signal RELEASEo : STD_LOGIC;
    signal VALIDINi     : STD_LOGIC;
    signal VALIDOUTo    : STD_LOGIC;
    signal WRRDYi       : STD_LOGIC;
--  signal LAo          : STD_LOGIC_VECTOR(LA'range);
--  signal LAoe         : STD_LOGIC;


-- Data Buffer Control
    signal DATA_OEo     : STD_LOGIC;
    signal DATA_DIRo    : STD_LOGIC;
    signal ADDRESS_OEo  : STD_LOGIC;
--BA Buffer Control
signal BA_OUTo   :STD_LOGIC_VECTOR(BA_OUTb'range);
signal BA_OUToe  :STD_LOGIC;
-- Main State Machine
    -- decoded state
    signal SYS_STATE    : STD_LOGIC_VECTOR(5 downto 0);
    constant IDLE       : integer := 0;
    constant DO_ADDRESS : integer := 1;
    constant DO_RD_DATA : integer := 2;
    constant DO_WR_WAIT : integer := 3;
    constant DO_WR_DATA : integer := 4;
    constant RECOVER    : integer := 5;
    --
    -- State Register
    signal SYS_STATE_REG : STD_LOGIC_VECTOR(3 downto 0);
--  signal NEXT_SYS_STATE: STD_LOGIC_VECTOR(3 downto 0);
    --                                                      3210
    --               -------
    constant NS_IDLE       : STD_LOGIC_VECTOR(3 downto 0) :="1100";
    constant NS_DO_RD_DATA : STD_LOGIC_VECTOR(3 downto 0) :="0010";
    constant NS_DO_WR_WAIT : STD_LOGIC_VECTOR(3 downto 0) :="0110";
    constant NS_DO_WR_DATA : STD_LOGIC_VECTOR(3 downto 0) :="0011";
    constant NS_RECOVER    : STD_LOGIC_VECTOR(3 downto 0) :="0000";
    constant NS_DO_ADDRESS : STD_LOGIC_VECTOR(3 downto 0) :="0101";
    --                                                        |||| | | |
    --                                              CS IDLE -- / | | |
    --                                            ADDRESS_OEo - - / | |
    --                                              DATA_OEo ---- / |
    --                                             Q_VALIDOUT ----- /


-- Intermediate signals
    signal BURST_OF_2   : STD_LOGIC;
    signal BURST_OF_4   : STD_LOGIC;
    signal Q_VALIDOUT   : STD_LOGIC;
    signal CS_IDLE: STD_LOGIC;
    signal BREAK_BURST  : STD_LOGIC;
    signal INCREMENT_BA : STD_LOGIC;
    signal Q_WRRDY      : STD_LOGIC;

-- Burst Address Counter
    signal BA           : STD_LOGIC_VECTOR(3 downto 2);

-- Indicates the last of a burst (inverse of BURST)
    signal LAST         : STD_LOGIC;
    signal SYS_BURST    : STD_LOGIC;


begin
--#############################################################################--
```

```
--##                                                                          ##--
--## I/O Buffers: Take the external I/O and create signals for internal   ##--
--##              use that are active high.                                ##--
--##                                                                          ##--
--##############################################################################--


    RESETi <= not To_X01(RESETb);
--================================================
-- Coldfire Signals
--================================================
    TSi     <= not (TSb);
    TAb     <= not TAo when (TAoe = '0') else 'Z';
    TAi     <= not TAb;
    Ai      <= A;
TTi     <=To_StdUlogicVector(To_X01 (TT));
    READ    <= (RW);
    SIZi    <= SIZ;


--================================================
-- V320USC Signals
--================================================

    RELEASEb   <= not RELEASEo;

    VALIDOUTb  <= not VALIDOUTo;


    VALIDINi   <= not VALIDINb;
    WRRDYi     <= not WRRDYb;

    SYSCMDi    <= SYSCMD;
    SYSCMD     <= (SYSCMDo) when (SYSCMDoe = '1') else (others => 'Z');


--================================================
-- Buffer Control Signals
--================================================
        DATA_OEb    <= not DATA_OEo;
    DATA_DIR    <= DATA_DIRo;
    ADDRESS_OEb <= not ADDRESS_OEo;
--================================================
External Address Signal
--================================================
BA_OUTb      <BA_OUTo when (BA_OUToe='0') else (others=>'Z');


--##############################################################################--
--##                                                                        ##--
--##         This is the core logic using all active high signals         ##--
--##                                                                        ##--
--##############################################################################--

---================================================
-- Generate SYSCMD
--================================================

    SYSCMDo(0) <= (not to_mvl(SIZi = "01") and (not BURST_OF_2)) or not ADDRESS_OEo;

    SYSCMDo(1) <= (to_mvl(SIZi = "00") and not to_mvl(SIZi = "11") or (to_mvl(SIZi = "11") and A(2)
and not BURST_OF_2)) and
    ADDRESS_OEo;

    SYSCMDo(2) <= ADDRESS_OEo and to_mvl(SIZi ="11")) and (A(2) xnor (BURST_OF_2));

    SYSCMDo(3) <= (not READ and ADDRESS_OEo) or (not ADDRESS_OEo and SYS_BURST);

    SYSCMDo(4) <= not ADDRESS_OEo;

    VALIDOUTo <= (TSi and SYS_STATE(IDLE)) or (Q_VALIDOUT and not READ) or (Read and
to_mvl(SIZi="11") and SYS_STATE(DO_ADDRESS));
    RELEASEo  <= READ; -- RELEASEb can be tied to ground
    TAo <= not ValidIni when (Read = '1') else   DATA_OEo;
    TAoe <= DATA_OEo;
BA_OUTo  <=Ai when TSi='1' else BA;

BA_OUToe<not ADDRESS_OEo;
```

```
process
    begin
        wait until(clk'event and (CLK = '1'));
        Q_WRRDY <= WRRDYi;
end process;

--================================================
-- V320USC Bus State Machine
--================================================

-- Pre-decode the state for convenience
    SYS_STATE(IDLE)       <= to_mvl(SYS_STATE_REG = NS_IDLE);
    SYS_STATE(DO_RD_DATA) <= to_mvl(SYS_STATE_REG = NS_DO_RD_DATA);
    SYS_STATE(DO_WR_WAIT) <= to_mvl(SYS_STATE_REG = NS_DO_WR_WAIT);
    SYS_STATE(DO_WR_DATA) <= to_mvl(SYS_STATE_REG = NS_DO_WR_DATA);
    SYS_STATE(RECOVER)    <= to_mvl(SYS_STATE_REG = NS_RECOVER);
    SYS_STATE(DO_ADDRESS) <= to_mvl(SYS_STATE_REG = NS_DO_ADDRESS);

process
    variable NS : STD_LOGIC_VECTOR(SYS_STATE_REG'range);
    begin
        wait until(clk'event and (CLK = '1'));

        if(RESETi = '1') then
            NS := NS_IDLE;
        else
            NS := (others => '0');

            case SYS_STATE_REG is
            when NS_IDLE =>
                if(TSi = '1') then
                    if(READ = '1') then NS := NS_DO_RD_DATA;
                    elsif(Q_WRRDY = '1') then NS := NS_DO_WR_DATA;
                    else NS := NS_DO_WR_WAIT;
                    end if;
                else                                                NS := NS_IDLE;
                end if;

            when NS_DO_RD_DATA =>
                if (VALIDINi='1') then
                    if (LAST='1') then

                if((VALIDINi and (not LAST)) = '0') thenNS := NS or NS_RECOVER;
                elsif(BREAK_BURST = '1') then NS := NS or NS_DO_ADDRESs;
                    else     NS := NS or NS_DO_RD_DATA;
                end if;
            else
            NS:= NS or NS_DO_RD_DATA;
            end if;

            when NS_DO_WR_WAIT =>
                if(Q_WRRDY = '1') then NS := NS_DO_WR_DATA;
                    else         NS := NS_DO_WR_WAIT;
                end if;

            when NS_DO_WR_DATA =>
                if(LAST = '1') then NS := NS_IDLE;
                elsif(BREAK_BURST = '1') then NS := NS_DO_ADDRESs;
                    else     NS := NS_DO_WR_DATA;
                end if;

            when NS_RECOVER =>
                if((LAST and TSi) = '1') then NS := NS_DO_ADDRESS;
                    else     NS := NS_IDLE;
                end if;

            when NS_DO_ADDRESS=>
                if(READ = '1') then                                                                 NS
:= NS_DO_RD_DATA;
                    else     NS := NS_DO_WR_DATA;
                end if;

            when others =>
                        NS := NS_IDLE;
            end case;

        end if;

        SYS_STATE_REG <= NS;
        SYSCMDoe <= NS(2) or NS(0);
```

```
    end process;

    Q_VALIDOUT    <= SYS_STATE_REG(0);
    DATA_OEo      <= SYS_STATE_REG(1);
    ADDRESS_OEo   <= SYS_STATE_REG(2);
    CS_IDLE       <= SYS_STATE_REG(3);
    DATA_DIRo     <= READ;  -- This may need to be changed depending on the type of buffer used

--=================================================
-- Burst Counter
--=================================================
    process
    begin
        wait until(clk'event and (CLK = '1'));

        if(RESETi = '1') then
            BA <= (others => '0');
        else
            -- This is the modulo burst counter
            if(TSi = '1') then -- load the counter
                BA <= Ai(BA'range);
            elsif(INCREMENT_BA = '1') then -- increment the counter
                BA <= BA + '1';
            end if;
        end if;

    end process;

    INCREMENT_BA <= SYS_STATE(DO_WR_DATA)
                or (SYS_STATE(DO_RD_DATA) and VALIDINi);

process(BA, SIZi, DATA_OEo)
begin
    case Ai(3 downto 2) is
        when "00"   =>
            LAST <= to_mvl(BA = "11") or (not to_mvl(SIZi = "11") and to_mvl(BA="00"));
            SYS_BURST <= not to_mvl(BA = "11");
            BREAK_BURST <= '0';
            BURST_OF_2 <= '0';

        when "01"   =>
            LAST <= to_mvl(BA = "00") or (not to_mvl(SIZi = "11") and to_mvl(BA="01"));
            SYS_BURST <= to_mvl(BA = "10") and DATA_OEo;
            BREAK_BURST <= (to_mvl(BA = "01")or to_mvl(BA = "11")) and to_mvl(SIZi = "11");
            BURST_OF_2 <= to_mvl(BA = "10") and to_mvl(SIZi = "11");

        when "10"   =>
            LAST <= to_mvl(BA = "01") or (not to_mvl(SIZi = "11") and to_mvl(BA="10"));
            SYS_BURST <= not to_mvl(BA = "01") and to_mlv(SIZi=”11”);
            BREAK_BURST <= '0';
            BURST_OF_2 <= '0';

        when others => -- "11"
            LAST <= to_mvl(BA = "10") or (not to_mvl(SIZi = "11") and to_mvl(BA="11"));
            SYS_BURST <= to_mvl(BA = "00") and DATA_OEo;
            BREAK_BURST <= (to_mvl(BA = "11") or to_mvl(BA = "01")) and to_mvl(SIZi = "11");
            BURST_OF_2 <= to_mvl(BA = "00") and ADDRESS_OEo and not CS_IDLE;
    end case;

end process;

BURST_OF_4 <= not Ai(2) and to_mvl(SIZi = "11");

end BEHAVIOUR;
```

## 1.7 Performance

The performance of the various alternatives has been calculated in **Table 1-7** and **Table 1-8.** Real world performance depends on a number of factors:

**Bursting:** The best performance depends on the ability to burst long data transfers with no wait states. Because the Coldfire processors are limited to four-word bursts, it is better to use the DMA controller on the V320USC to transfer large volumes of data. The DMA controller is capable of sustaining up to 1 KB bursts. Also, setting the PBRST_MAX to 256 words provides better performance for the PCI to/from SDRAM.

**Data Flow:** It is better to push/ write data rather than pull/ read data. Pushing takes advantage of the large 256 byte write FIFOs on the V320USC. Writing data can also be posted so that the final target device doesn't have to respond for the originating master to start data transfer. Reading non-consecutive locations requires a new address to be transferred to the target before any progress can be made with data movement.

**Sequential Reads:** The data flow cannot be accommodated in writing. To take advantage of the read prefetching buffers, reading should be done as sequential bursts. When possible, use the dual apertures because they have their own prefetch buffers.

**System Bottlenecks:** Performance is affected while using non-bursting or slow target devices. Also, additional traffic on the PCI or local bus allows less time for V320USC transfers. The PCI system's main memory can also be a bottleneck because the V320USC must compete with the host CPU to get access to this shared resource. Host CPU intensive applications that are cache bound allow improved PCI data movement because the CPU doesn't need access to the main memory very often.

Table 1-7: Local to SDRAM Burst Read

| | Number of LCLKs | LCLK = 45 MHz Rate (MB/sec) |
|---|---|---|
| Aligned Burst | 9 | 80 |
| Non-aligned Burst | 19 | 37.9 |

Table 1-8: PCI to SDRAM Burst Write

| | Number of PCLKs | PCLK = 33 MHz Rate (MB/sec) | PCLK = 45 MHz Rate (MB/sec) |
|---|---|---|---|
| 4 Word Burst | 5 | 105.6 | 144 |
| 8 Word Burst | 9 | 117.3 | 160 |
| 16 Word Burst | 17 | 124.2 | 169.4 |
| 256 Word Burst | 257 | 131.5 | 179.3 |