BU-692XXIX
ACE Flex-Core Intellectual Property: Hardware Guide
MN-692XXIX-002

The information provided in this Manual is believed to be accurate; however, no responsibility is assumed by Data Device Corporation for its use, and no license or rights are granted by implication or otherwise in connection therewith.

Specifications are subject to change without notice.
Please visit our web site at www.ddc-web.com for the latest information.

105 Wilbur Place, Bohemia, New York 11716-2426

For Technical Support - 1-800-DDC-5757 ext. 7771

Headquarters - Tel: (631) 567-5600, Fax: (631) 567-7358
United Kingdom - Tel: +44-(0) 1635-811140, Fax: +44-(0) 1635-32264
France - Tel: +33-(0) 1-41-16-3424, Fax: +33-(0) 1-41-16-3425
Germany - Tel: +49-(0) 8141-349-087, Fax: +49-(0) 8141-349-089
Japan - Tel: +81-(0) 3-3814-7688, Fax: +81-(0) 3-3814-7689
World Wide Web - http://www.ddc-web.com

DATA DEVICE CORPORATION
REGISTERED TO ISO 9001
FILE NO. A5976

REV C – February 2009

RECORD OF CHANGE

Revision	Date	Pages	Description
A	12/2005	All	Original Release
B	02/2008	All	Addition of Supplemental Design Data
C	02/2009	All	Removal of Proprietary Information in footer

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 PREFACE

1.1 Purpose

The purpose of this User's Guide is to provide applicable information to facilitate the simulation and synthesis of the ACE Flex-Core Intellectual Property into a customer's System-on-a-Chip (SOC).

1.2 Organization

First, all ACE Flex-Core signal I/O's are discussed. I/O signals are listed by functional groups, such as local bus I/F support (microprocessor or memory), transceiver interfacing, etc. This section describes the top level I/O, control information, and discusses memory considerations.

Next, the process and techniques of simulation are covered. A detailed listing of all simulation commands and results will be outlined.

Finally synthesis concerns and requirements will be discussed.

This document additionally provides several appendices to assist the designer along the way, including:

- Example synthesis and test bench report, providing the designer with a comparative reference.

- Timing information in the form of tables and diagrams. (Any timing information discussed in this guide refers to clock edges only and not actual delays because Intellectual Property is inherently technology independent. Exact delays, setup times, etc. are only known after synthesis within a library.)

1.3 Conventions Used in This User's Guide

This User's Guide uses typographical and iconic conventions to assist the reader in understanding the content. This section will define the text formatting and icons used in the rest of the document. This User's Guide is formatted with a 'Scholar Margin' where many tips, symbols or icons will be located.

WORD – 16-bit wide data

DWORD – 32-bit wide data

IP Block – ACE Flex-Core Design instantiation

1.4 Text Usage

BOLD ITALIC – will designate DDC Part Numbers.

Courier New – is used to indicate code examples.

<…> - Indicates user entered text or commands.

ITALIC – *is used to indicate signal names – e.g.* clk

BOLD *– is used to indicate ACE Flex-Core block names – e.g.* ***A_Decoder****, or a Test Bench command instruction.*

1.5 Trademarks

All trademarks are the property of their respective owners.

1.6 Special Notes

Unless otherwise noted, an asterisk after a signal name (i.e., MSTCLR*) denotes an active low signal.

1.7 Technical Support

In the event that problems arise beyond the scope of this manual, you can get in touch with DDC by calling:

Customer support:

1-800-DDC-5757, ext. 7771

Headquarters:

1-631-567-5600, ext. 7771

Fax: 1-631-567-5758 to the attention of DATA BUS Applications.

Regional Offices:

USA & Canada:

East of the Mississippi River:
Tel: (631) 567-5600 Fax: (631) 567-7358

West of the Mississippi River:
Tel: (714) 895-9777 Fax: (714) 895-4988

Europe:

Tel: 44 (1635) 811140 Fax: 44 (1635) 32264

Asia/Pacific:

Tel: 81 (3) 3814-7688 Fax: 81 (3) 3814-7689

DDC also has an Internet World Wide Web site, which allows customers to easily download new revisions of software and documentation. The Internet address is **www.ddc-web.com**.

2 INTRODUCTION

2.1 DDC's ACE Flex-Core Intellectual Property

DDC's ACE Flex-Core Intellectual Property product is the next generation of superior MIL-STD-1553 interfacing technology. The design of the ACE Flex-Core stems from DDC's battle tested Enhanced Mini-ACE technology. The ACE Flex-Core offering gives designers a turn-key solution to incorporate DDC's renowned MIL-STD-1553 technology into their own FPGA or ASIC designs.

2.2 Other Support Documents

This ACE Flex-Core User's Guide has been written to address the unique issues of this Intellectual Property offering.

In addition, information about MIL-STD-1553 can be obtained from the following:

- ACE Flex-Core Intellectual Property: Architectural User's Guide (MN-692XXiX-001)
- DDC's MIL-STD-1553 Designer's Guide
- MIL-STD-1553 Handbook
- MIL-STD-1553B Specification
- Electrical and Layout Considerations for 1553 Terminal design (DDC AN/B27)

2.3 ACE Flex-Core Directories and Files

The Release_Notes.txt file in the ACE Flex-Core directory contains important information about the ACE Flex-Core directories and files and should be read before the files are used. It will supersede this User's Guide.

3 FUNCTIONAL DESCRIPTION

This ACE Flex-Core I/O section introduces the signals, the main requirements and the functions of the I/O groups with which the user will have to directly interface.

The ACE Flex-Core top level block can be found in Figure 1.

Signals can be organized into several functional groups:

- Transceiver Interface Signals
- Host Bus Arbitration and Control Logic Interface
- DMA Bus Arbitration and Control Logic Interface
- Reset and Clocks
- Flex-Core ROM Data Interface
- Flex-Core Synchronous RAM Interface
- Flex-Core Local Scratch RAM Data Interface
- Flex-Core Auxiliary Local Scratch RAM Data Interface (Additional Scratch RAM required for Multi-RT Cores)
- Miscellaneous Signals

3.1 Functional Level Block Diagram

The following is a functional level block diagram of ACE Flex-Core.

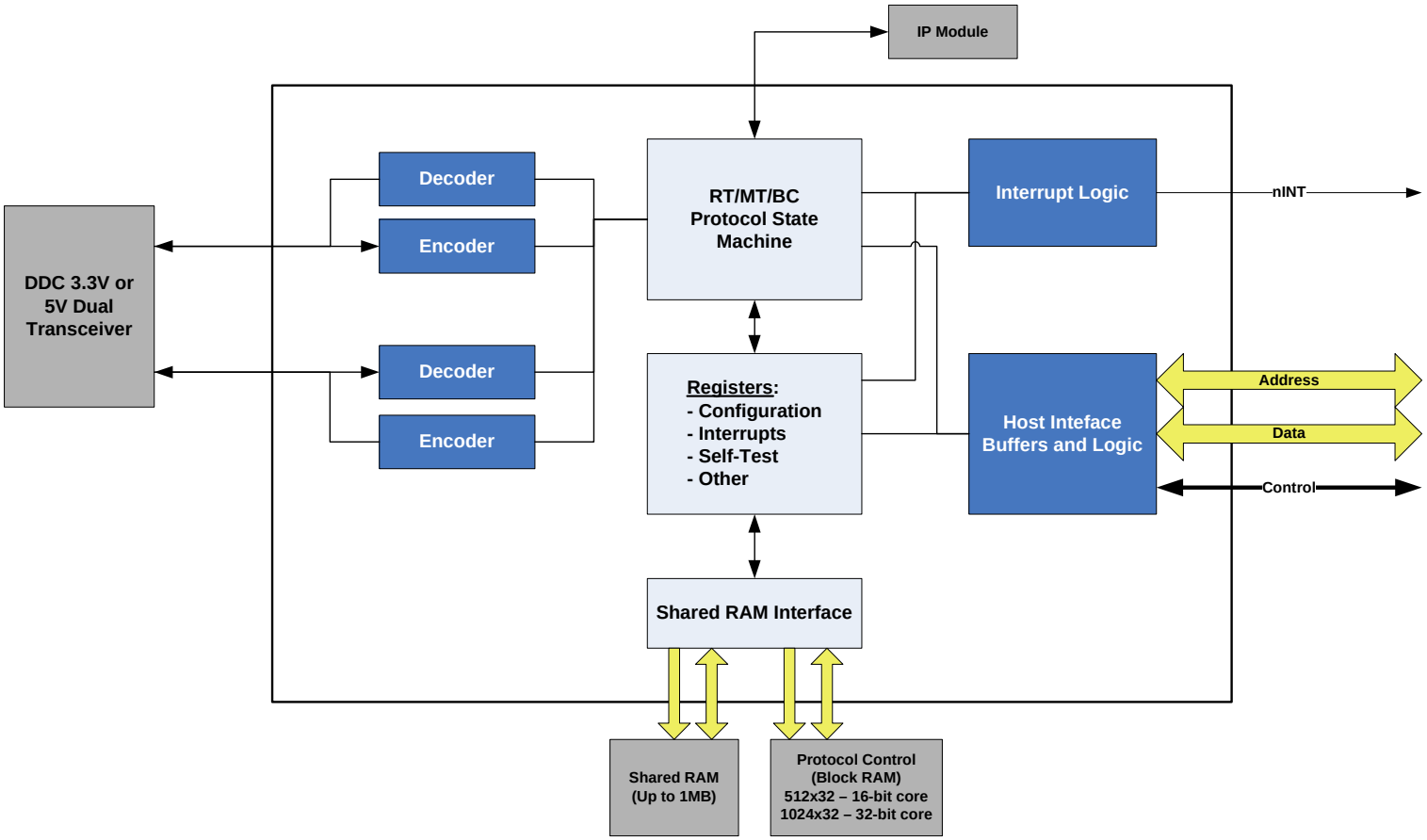


Figure 1: ACE Flex-Core Functional Level Block Diagram

3.2 ACE Flex-Core Signal Descriptions

The ACE Flex Core breaks out the following external signals. Any signal groups or individual signals denoted by ** do not exist in BC only implementations. *** denotes only exists on Multi-RT core.

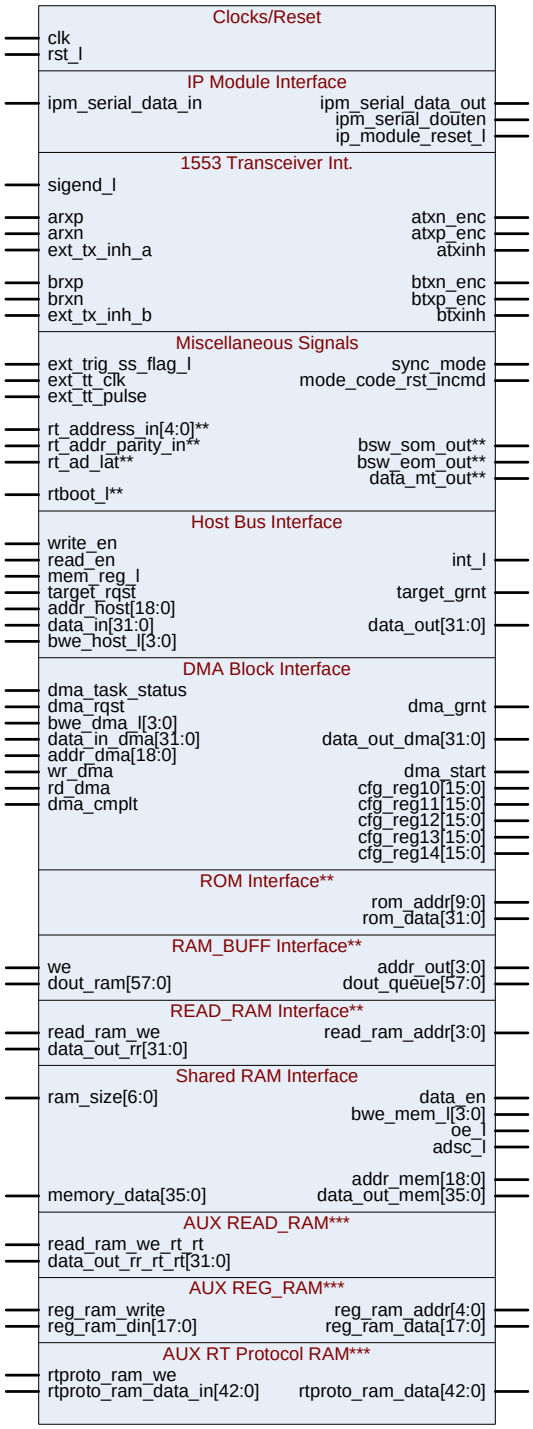


Figure 2: FlexCore Signals

3.2.1 MIL-STD-1553 Transceiver Interface

This is the standard MIL 1553 Transceiver interface. These signals should be connected to the 1553 Transceivers as per section 3.4

Table 1: MIL-STD-1553 Transceiver Interface signals

Port Name	Port Type	Description
arxp	in *std_logic*	**Channel A +RX Input** Manchester receive data positive differential input from 1553 Transceiver.
arxn	in *std_logic*	**Channel A -RX Input** Manchester receive data negative differential input from 1553 Transceiver.
brxp	in *std_logic*	**Channel B +RX Input** Manchester receive data positive differential input from 1553 Transceiver.
brxn	in *std_logic*	**Channel B -RX Input** Manchester receive data negative differential input from 1553 Transceiver.
atxp_enc	out *std_logic*	**Channel A +TX Output** Manchester transmit data positive differential output to 1553 Transceiver.
atxn_enc	out *std_logic*	**Channel A -TX Output** Manchester transmit data negative differential output to 1553 Transceiver.
btxp_enc	out *std_logic*	**Channel B +TX Output** Manchester transmit data positive differential output to 1553 Transceiver.
btxn_enc	out *std_logic*	**Channel B -TX Output** Manchester transmit data negative differential output to 1553 Transceiver.
atxinh	out *std_logic*	**Channel A Transceiver TX inhibit** Connect to TX_INH_OUTA input of a MIL-STD-1553 transceiver. Asserted high to inhibit transmission.
btxinh	out *std_logic*	**Channel B Transceiver TX inhibit** Connect to TX_INH_OUTB input of a MIL-STD-1553 transceiver. Asserted high to inhibit transmission.
ext_tx_inh_a	in *std_logic*	**Channel A TX_ENC Inhibit** When asserted, no data is sent out on the atxp/n_enc outputs. Configuration Register #5, bit 13 tracks the input of this signal.
ext_tx_inh_b	in *std_logic*	**Channel B TX_ENC Inhibit** When asserted, no data is sent out on the btxp/n_enc outputs. Configuration Register #5, bit 12 tracks the input of this signal.
sigend_l	in *std_logic*	**Single Ended Transceiver Enable** This input determines if the input 1553 bus will be single ended (e.g. to support a MIL-STD-1773 fiber optic interface). Configuration Register #5, bit 14 tracks the input of this signal. Active Low signal. Connect this input to logic 1 for conventional MIL-STD-1553 interfaces.

3.2.2 Host Bus Interface

The signal list below is the Host Bus interface signals. The Host may access the Shared RAM and IP block registers through this interface. If a 32-bit core is implemented, the data bus width is 32-bits and DWORD addressed. If a 16-bit core is implemented, then the data bus width is 16-bits and WORD addressed.

Table 2: Host Bus Interface signals

Port Name	Port Type	Description
addr_host [18:0]	in *std_logic_vector*	**Host Bus Address Input** 19-bit Address input for Memory and Registers based on the state of the mem_reg_l signal. 16-Bit Core = WORD Addressed 32-Bit Core = DWORD Addressed
data_in [31:0]	in *std_logic_vector*	**Host Bus Data Input** 32-bit wide input data bus for Host Data Writes. Note: When using the 16-bit core bits [31:16] are considered to be Don't Care.
data_out [31:0]	out *std_logic_vector*	**Host Bus Data Output** 32-bit wide output data bus for Host Data Reads. Note: When using the 16-bit core bits [31:16] are considered to be Don't Care.
mem_reg_l	in *std_logic*	**Memory/Register Select** Selects between memory access (mem_reg_l = '1') or register access (mem_reg_l='0'). Generally connected to either a CPU address line or address decoder output. Note: Register locations are held inside the core. RAM access are always external to the core.
write_en	in *std_logic*	**Host Bus Write Strobe** Active high signal. When asserted by host, the data on the data_in[31:0] signals is latched into memory/register location selected by addr_host[18:0] and mem_reg_l.
read_en	in *std_logic*	**Host Bus Read Strobe** Active high signal. When asserted by host, the memory/register data selected by addr_host[18:0] and mem_reg_l is driven on the data_out[31:0] signals.

Port Name	Port Type	Description
bwe_host_l [3:0]	in *std_logic_vector*	**Host Bus Byte Write Enables** Active low Byte Write enables for memory transactions. If asserted, enables associated data_in byte lane to be written to the same memory_data byte lane. bwe_host_l[0] = data_in[7..0] bwe_host_l[1] = data_in[15..8] bwe_host_l[2] = data_in[23..16] bwe_host_l[3] = data_in[31..24] Note: A 32-bit transfer to memory will occur on a memory write transfer when bwe_host_l[3:0] = 0000, while a 16-bit transfer will occur when bwe_host_l[3:0] = 0011 or 1100. Any combination of the four byte lanes may be enabled during a memory write access.
int_l	out *std_logic*	**Interrupt request** Pulse or level interrupt will occur based on the selection in Configuration Register #2, bit 3. Active Low signal.
target_rqst	in *std_logic*	**Host Bus Data Transfer Request** Active high input signal used by the host to request access to the Flex-Core's internal shared RAM. When asserted, indicates that the host is requesting access to the shared RAM. The Flex-Core will arbitrate for access to the shared RAM and will assert the target_grnt once arbitration is complete and access is given to the host. The host must ensure that the target_grnt is not asserted active high for more than 1 usec. The host is not required to assert the target_rqst signal to perform register transfers.
target_grnt	out *std_logic*	**Host Bus Data Transfer Grant** Active high output signal from the Flex-Core used to indicate the status of host arbitration for host access to the shared RAM. When asserted, indicates that the Flex-Core has given access to the shared RAM over to the Host bus and the host may perform the requested transactions.

3.2.3 Resets and Clocks

Below are the Reset and Clock inputs to the IP core

Table 3: Reset and Clock signals

Port Name	Port Type	Description
rst_l	in *std_logic*	**Master reset input** Active low input. When asserted, will hold the Flex-Core in reset state. Must be asserted low following power turn on to initialize the Flex-Core.
clk	in *std_logic*	**Clock Input** 40 MHz input clock.

3.2.4 ROM Data Interface Signals

The interface connects to the Flex-Core ROM for RT operation. This interface only exists when RT functionality has been implemented in

the IP core. Please see section 3.3.4 for further information regarding size and configuration.

Table 4: ROM Data Interface Signals

Port Name	Port Type	Description
rom_addr [9:0]	out *std_logic_vector*	**ROM Address Bus** Address bus for internal ROM. ROM should be created as an internal design block.
rom_data [31:0]	in *std_logic_vector*	**ROM Data Bus** Data bus for internal ROM. ROM should be created as an internal design block

3.2.5 Shared RAM Interface

These signals interface to the Shared RAM memory (up to 4Mbit). The width and depth of this memory are user selectable based on functionality requirements along with whether the 16-bit or 32-bit core has been selected. Please see section 3.3.5 for further information.

Table 5: Shared RAM Interface Signals

Port Name	Port Type	Description
memory_data [35:0]	in *std_logic_vector*	**Shared RAM Data Input** Data Input from Shared RAM. memory_data[35:32] contain parity bits for memory_data[31:0]. memory_data[32] is the parity for memory_data[7:0] memory_data[33] is the parity for memory_data[15:8] memory_data[34] is the parity for memory_data[23:16] memory_data[35] is the parity for memory_data[31:24]
data_out_mem [35:0]	out *std_logic_vector*	**Shared RAM Data Ouput** Data Output to Shared RAM. Data_out_mem[35:32] contain parity bits for data_out_mem[31:0]. data_out_mem[32] is the parity for data_out_mem[7:0] data_out_mem[33] is the parity for data_out_mem[15:8] data_out_mem[34] is the parity for data_out_mem[23:16] data_out_mem[35] is the parity for data_out_mem[31:24]
addr_mem [18:0]	out *std_logic_vector*	**Shared RAM Address** 19-bit Address bus output to Shared RAM
data_en	out *std_logic*	**Shared RAM Data Write Enable** Active high write enable output to Shared RAM. When asserted, indicates an active write cycle to Shared RAM.

Port Name	Port Type	Description
bwe_mem_l [3:0]	out *std_logic_vector*	**Shared RAM Byte Write Enables** Active low Byte Write enables to Shared RAM. If asserted, enables write on associated data_out_mem byte. **bwe_mem_l[0]** = data_out_mem [7..0] **bwe_mem_l[1]** = data_out_mem [15..8] **bwe_mem_l[2]** = data_out_mem [23..16] **bwe_mem_l[3]** = data_out_mem [31..24] Note: A 32-bit transfer to memory will occur on a memory write transfer when bwe_mem_l[3:0] = 0000, while a 16-bit transfer will occur when bwe_mem_l[3:0] = 0011 or 1100. Any combination of the four byte lanes may be enabled during a memory write access.
oe_l	out *std_logic*	**Shared RAM Output Enable** Active low Output enable to Shared RAM. Asserted during read cycles
adsc_l	out *std_logic*	**Shared RAM Address Status Control** Inverted version of data_en. Used for specific SRAM models.
ram_size [6:0]	in *std_logic_vector*	**Shared RAM size control** Static Indication of Local RAM size. Used by the Flex-Core to perform RAM self-test. Local RAM size = (ram_size + 1)*4K x 32/16*{based on 32-bit or 16-bit core}.* If parity is used, the memory width will increase by 1-bit for each Byte lane. See 3.2.5 for further information with regards to memory configurations.

3.2.6 RAM BUFF Interface

These signals interface to the RAM BUFF block. Configuration information for this memory block can be found in section 3.3.3.1.

Table 6: RAM BUFF Interface

Port Name	Port Type	Description
addr_out [3:0]	out *std_logic_vector*	**Address Bus** Internal connection to RAM BUFF address bus
dout_queue [57:0]	out *std_logic_vector*	**Data Output** Internal connection to RAM BUFF data input for writes.
dout_ram [57:0]	in *std_logic_vector*	**Data Input** Internal connection to RAM BUFF data output for reads.
we	out *std_logic*	**Write Enable** Internal connection to RAM BUFF Write Enable.

3.2.7 READ RAM Interface

These signals interface to the READ RAM block. Configuration information for this memory block can be found in section 3.3.3.1.

Table 7: READ RAM Interface

Port Name	Port Type	Description
read_ram_addr [3:0]	out *std_logic_vector*	**Address Bus** Internal connection to READ RAM address bus
data_out_rr [31:0]	in *std_logic_vector*	**Data Input** Internal connection to READ RAM data output for reads.
read_ram_we	out *std_logic*	**Write Enable** Internal connection to READ RAM Write Enable.

3.2.8 DMA Bus Interface

These signals are used to interface to the DMA interface module for PCI bus operation. This is an optional configuration. Please see section 6 for further information on connection and configuration of this block.

NOTE: This block is intended to operate with the Xilinx PCI IP core.

Table 8: DMA Bus Interface signals

Port Name	Port Type	Description
addr_dma [18:0]	in *std_logic_vector*	**DMA Address** 19-bit address for Memory/Register Access.
data_in_dma [31:0]	in *std_logic_vector*	**DMA Data Input** Data for Memory/Register Writes to Flexcore.
wr_dma	in *std_logic*	**DMA Write strobe** DMA Bus Write strobe. Enables write operations for a DMA module access.
rd_dma	in *std_logic*	**DMA Read strobe** DMA Bus Read strobe. Enables read operations for a DMA module access.
bwe_dma_l [3:0]	in *std_logic_vector*	**DMA Byte Write Enables** Active low signals used by DMA Bus to enable byte lanes during write operations.
dma_rqst	in *std_logic*	**(Target MODE)DMA Bus Data Transfer Request** Active high signal indicating a request for a DMA transfer request to Memory or Register space from an external PCI host. The host must wait until the dma_grnt is asserted before proceeding with the requested transfer.

Port Name	Port Type	Description
dma_grnt	out *std_logic*	**(Target MODE)DMA Data Transfer Grant** Active high signal indicating that the DMA transfer requested by the external PCI host may now be performed. NOTE: The host must monitor this signal in the event the Arbiter rescinds host access.
dma_cmplt	in *std_logic*	**(Initiator MODE)DMA Transfer Complete** Active high indication to ACE Flex-Core that the requested DMA transfer has completed. This signal is output from the DDC supplied PCI – DMA interface. Used when the ACE FlexCore is performing an Initiator MODE DMA Transfer.
dma_start	out *std_logic*	**(Initiator MODE)DMA Transfer Start** Active high request from ACE Flex-Core to the DDC supplied PCI – DMA interface block to arbitrate for an Initiator MODE DMA Transfers originating from the ACE FlexCore IP block.
dma_task_status [3:0]	in *std_logic_vector*	**DMA Transfer Task Status** Indicates the status of the DMA transfer as per the list below. -- 000 = Successful -- 001 = Master Abort -- 010 = Target Abort
cfg_reg10 [15:0]	out *std_logic_vector*	**Configuration Register #10 output** 15:1 = DMA Transfer Starting PCI Address [16:2] 0 = Write/Read*
cfg_reg11 [15:0]	out *std_logic_vector*	**Configuration Register #11 output** 15:0 = DMA Transfer Starting PCI Address [31:17]
cfg_reg12 [15:0]	out *std_logic_vector*	**Configuration Register #12 output** 15:0 = DMA Transfer Starting Memory Address [15:0]
cfg_reg13 [15:0]	out *std_logic_vector*	**Configuration Register #13 output** 2:0 = DMA Transfer Starting Memory Address [18:16]
cfg_reg14 [15:0]	out *std_logic_vector*	**Configuration Register #14 output** 14:0 = DMA Transfer Size 14:0 (Word Oriented)

3.2.9 Auxiliary Scratch RAM Interfaces

These signals interface to additional scratch RAM only required when the Multi-RT core is utilized.

3.2.9.1 Auxiliary READ RAM

Table 9: Auxiliary READ RAM Interface

Port Name	Port Type	Description
data_out_rr_rt_rt [31:0]	in *std_logic_vector*	Data Bus Input (for internal use only)
read_ram_we_rt_rt	out *std_logic*	Write Enable (for internal use only)

3.2.9.2 Auxiliary REG RAM

Used to hold local Register contents for individual RT Addresses.

Table 10: Auxiliary REG RAM Interface

Port Name	Port Type	Description
reg_ram_write	out *std_logic*	**REG RAM Write Enable**
reg_ram_din [17:0]	in *std_logic_vector*	**REG RAM Data Bus Input**
reg_ram_addr [4:0]	out *std_logic_vector*	**REG RAM Address Bus Output**
reg_ram_data [17:0]	out *std_logic_vector*	**REG RAM Data Bus Output**

3.2.9.3 Auxiliary RT Protocol RAM

Table 11: Auxiliary RT Protocol RAM Interface

Port Name	Port Type	Description
rtproto_ram_data [42:0]	buffer *std_logic_vector*	“RT PROTOCOL RAM”
rtproto_ram_data_in [42:0]	in *std_logic_vector*	“RT PROTOCOL RAM”
rtproto_ram_we	out *std_logic*	“RT PROTOCOL RAM”

3.2.10 IP Module interface

Signals used to communicate with DDC's "IP Module". A Bi-Directional IO buffer must be used to create the IPM_SERIAL_DIO and connected to the "Data" port on the BU-69299 "IP Module". Based on the IP core used, each instance of the Flex-Core IP may require the use of a BU-69299 "IP Module". For IP cores requiring the module, they will not operate for longer than 65 mS without an IP Module. Following a hardware reset, the Flex-Core will establish communication with the IP Module over this single wire "data" path.

Please refer to section 7 for proper connection and usage of these signals.

Table 12: IP Module Interface

Port Name	Port Type	Description
ipm_serial_douten	out *std_logic*	**IP Module serial data buffer enable** Should be connected to the enable line of a Bi-directional IO buffer.
Ipm_serial_data_out	out *std_logic*	**IP Module serial data output** Should be connected to the input to a Bi-directional IO buffer.
ipm_serial_data_in	in *std_logic*	**IP Module serial data input** Should be connected to the output of a Bi-directional IO buffer
ip_module_reset_l	out *std_logic*	**IP Module reset** Active low output. Must be connected to the "reset_l" port on the BU-69299 "IP Module".

3.2.11 Miscellaneous

These signals are utilized for various functionality that may or may not be required based on the specific application.

Table 13: IP Miscellaneous Signals

Port Name	Port Type	Description
rt_address_in [4:0]	in *std_logic_vector*	**RT Address Input** This is the 5-bit external RT Address input. It is used in various configurations based on “rt_ad_lat” and internal register configurations. Please see the description of “rt_ad_lat” below and the Architectural reference for proper usage. Note: Non-Multi-RT Cores only.
rt_addr_parity_in	in *std_logic*	**RT Address Parity Input** This is the odd-parrity signal for the “rtaddress_in[4:0]” input. If parity is incorrect, the device will only respond to Broadcast messages. Please see the description of “rt_ad_lat” below and the Architectural reference for proper usage. Note: Non-Multi-RT Cores only.
rt_ad_lat	in *std_logic*	**RT Address Latch** Active low RT Address latch control input. When cleared to 0, the latch becomes transparent, allowing the rt_address_in[4:0] and rt_addr_parity_in to be routed to the RT Address control logic. NOTE: This signal functions only when bit 5 of Configuration register #6 is cleared to a 0. If this register bit is set to a 1, the latching of the RT Address is controlled by Host register writes. Please see the Architectural reference for proper usage. Note: Non-Multi-RT Cores only.
rt_multi_en [31:0]	in *std_logic_vector*	**Multi-RT Address Enable input** Each bit corresponds to a respective RT address. To enable a given RT, in Multi-RT Core, set corresponding bits = ‘1’. Note: Only required for Multi-RT Core.
rtboot_l	in *std_logic*	**RT Auto Boot input** Active Low signal. If asserted, the ACE Flex-Core will automatically initialize as an RT capable of responding to valid messages with the busy bit set, after reset is removed. Otherwise the core is come out of reset in an Idle state.

Port Name	Port Type	Description
ext_trig	in *std_logic*	**External BC Trigger** Rise Edge triggered control. If the IP block, configured as a BC, encounters a Wait for External Trigger (WTG) instruction it will wait for a rising edge here before proceeding to the next instruction. This input has no effect in Message Monitor or RT modes.
ext_tt_clk	in *std_logic*	**External time tag clock Input** External clock that may be used to increment the Time Tag Register. This option is controlled by Configuration register #2. Please see the Architectural reference for proper usage.
ext_tt_pulse	in *std_logic*	**External time tag Pulse Input** Active high level sensitive input. Based on the setting of Configuration Register #6, this signal can cause one of three results. 1) Cause the contents of the TimeTag Register to be loaded into the TimeTag counter. 2) Cause The TimeTag counter to reset. 3) Cause the value previously written by host to the TimeTag Register(05h) to be loaded into the time tag. Please see the Architectural reference for proper usage. NOTE : external pulse s/b at least two 40MHz clock ticks wide.
sync_mode	out *std_logic*	**Synchronize Mode Code Detected** Active high signal indicating a ***Synchronize*** Mode Code occurred.
Mode_code_rst_incmd	out *std_logic*	RT Mode (only): This signal can performs the following: 1) If Configuration Register #7, Bit 0 is programmed to a value of '0' this signal = '1' while the RT is currently processing a message. 2) If Configuration Register #7, Bit 0 is programmed to a value of '1' this signal = '1' following receipt of a Mode Code Reset command.

3.3 Functional Description

3.3.1 Resets

Three different reset's are utilized internally based on reset conditions. The following shows each configuration.

rst_l = External HW reset
sw_rst_l = Internal SW Reset from Reg03h
protocol_rst_l = RT 1553 protocol initiated reset

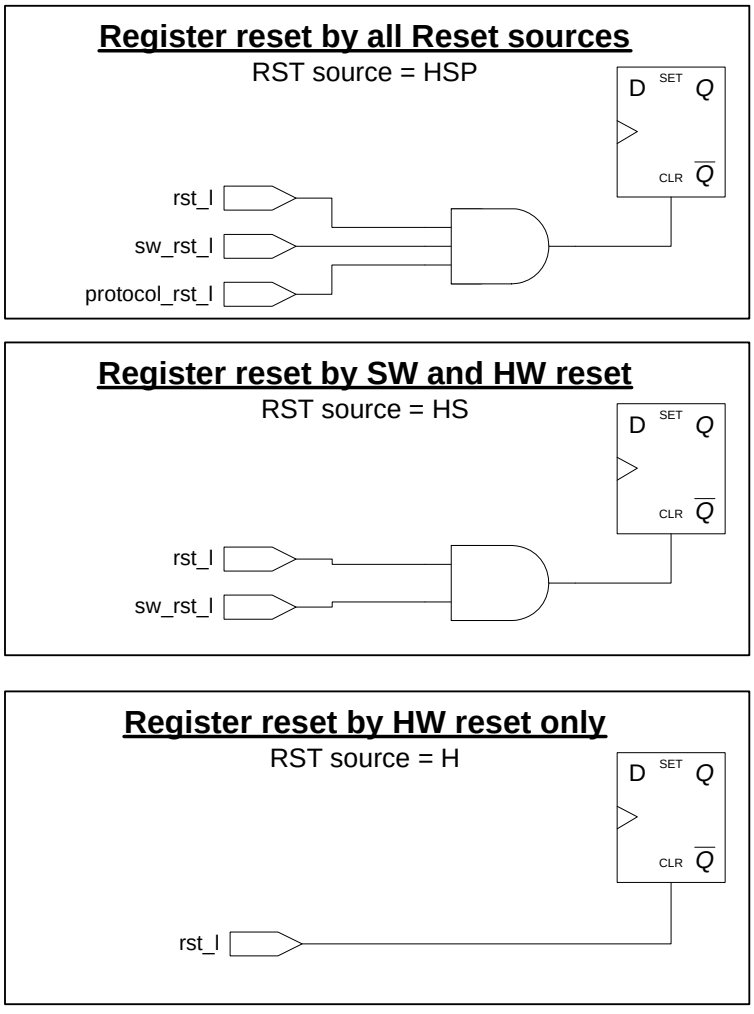


Figure 3: Reset types

The *rst_l* signal should be kept at a low level until power is stable to ensure no spontaneous transmission on the 1553 bus. A low level on *rst_l* asynchronously forces all resets active. The resets release on the third rising edge of *clk* after a *rst_l* has gone to a '1'. The software reset is generated on a write to the Start/Reset Register bit 0. The

protocol reset is generated when ACE Flex-Core is switched between modes of operation. More specifically, a protocol reset occurs when in BC, MT, or RT mode and a switch is made among the three modes. The software and protocol resets are completely synchronous with *clk*.

3.3.2 Clocks

The signal *clk* is the master clock for the system and is used to clock all the D flip-flop's in the ACE Flex-Core code. The falling edge is used only in the A_Decoder and B_Decoder sections and this technique is discussed in section 5.2. The single allowable frequency for *clk* is 40 MHz.

3.3.3 Memory Requirements

The following is a list of all RAM/ROM requirements for the core.

3.3.3.1 RT Memory Requirements

The memory below is required for any cores implementing RT functionality. The RAM BUFF and READ RAM blocks may be implemented in HDL compiled memory blocks, as they are relatively small.

Table 14: Memory Requirements for RT mode

BLOCK NAME	Configuration	Port connections
RAM BUFF	16x58 Synchronous Active High WE	**ADDR[3:0]** = addr_out [3:0] **DATA_IN[57:0]** = dout_queue [57:0] **DATA_OUT[57:0]** = dout_ram [57:0] **WE** = we
READ RAM	16x32 Synchronous Active High WE	**ADDR[3:0]** = read_ram_addr [3:0] **DATA_IN[31:0]** = memory_data [31:0] (monitors memory input to Flexcore IP) **DATA_OUT[31:0]** = data_out_rr [31:0] **WE** = read_ram_we
ROM	Read-Only No Pipelining 1024x32 (32-Bit IP) -or- 512x32(16-Bit IP)	**ADDR[9:0]** = rom_addr [9:0] **DATA[31:0]** = rom_data [31:0]

3.3.3.2 BC Memory Requirements

The BC operation requires no ancillary memory blocks.

3.3.3.3 Multi-RT Memory Requirements

When Multi-RT mode is implemented, the following memory is required in addition to the RT base requirements in 3.3.3.1. As these memory blocks are rather small, the option to implement them as HDL compiled memory exists.

Table 15: Memory Requirements for Multi-RT mode

BLOCK NAME	Configuration	Port connections
READ RAM RT RT	16x32 Synchronous Active High WE	**ADDR[3:0]** = read_ram_addr [3:0] *(monitors memory input to Flexcore IP)* **DATA_IN[31:0]** = memory_data [31:0] **DATA_OUT[31:0]** = data_out_rr_rt_rt [31:0] **WE** = read_ram_we_rt_rt
REG RAM	32x18 Synchronous Active High WE	**ADDR[4:0]** = reg_ram_addr [4:0] **DATA_IN[17:0]** = reg_ram_data [17:0] **DATA_OUT[17:0]** = reg_ram_din [17:0] **WE** = reg_ram_we
RT PROTOCOL RAM	32x43 Synchronous Active High WE	**ADDR[4:0]** = reg_ram_addr [4:0] **DATA_IN[42:0]** = rtproto_ram_data [42:0] **DATA_OUT[42:0]** = rtproto_ram_data_in [42:0] **WE** = rtproto_ram_we

3.3.4 Flex-Core ROM Data Interface

All versions of ACE Flex-Core require ROM for the Protocol State Machine. 16-bit versions of the core make use of a 512x32 ROM, while 32-bit core architectures require a 1024x32 ROM.

3.3.5 Shared RAM Interface

The main purpose of RAM in a 1553 interface is to provide storage space for received message data, and message and control data for messages to be transmitted. This storage space must be accessible by 1553 protocol logic enable real time message processing. In addition, it must be accessible by the host processor to enable transmit data and control information to be written, and received data and message status information to be read.

Traditionally, single port SRAMs will be used here due to the higher cost of dual port SRAMs, however, many modern FPGAs provide internal block RAM, which may be configured as a dual port RAM. The Dual-Port RAM will greatly improve performance, allowing the Host unarbitrated access to the message data.

This interface allows the connection of up to 2MB of memory. The bus Memory size = ((*ram_size[6:0]*+1)*4K) x 'DataWidth' (where DataWidth is 16/18-bits for the 16-bit core or 32/36-bits for the 32-bit core).

Therefore if *ram_size[6:0]* = 0000011b then the RAM is (3+1) x 4K = 16K.

Table 16 denotes example *ram_size*[6:0] configurations.

Table 16: RAM Size Configuration

ram_size[6:0]	RAM SIZE
0000000	4K
0000001	8K
0000011	16K
0000111	32K
0001111	64K
0011111	128K
0111111	256K
1111111	512K

3.3.5.1 16-bit IP Block Architecture

Each data word in RAM is 16 bits wide. An additional two bits are required in order to calculate RAM parity. Although two bits are used in parity calculation, Parity errors are indicated on the full16-bit word only. RAM parity can be added to the core by including the RAM PARITY BLOCK during design synthesis. When parity generation and checking logic is present in the ACE Flex-Core an interrupt can be issued to indicate a parity error.

3.3.5.2 32-bit IP Block Architecture

Each data word in RAM is 32 bits wide. An additional four bits are required in order to calculate RAM parity. Although four bits are used in parity calculation, Parity errors are indicated on the full 32-bit word only. RAM parity can be added to the core by including the RAM PARITY BLOCK during design synthesis. When parity generation and checking logic is present in the ACE Flex-Core an interrupt can be issued to indicate a parity error.

3.4 1553 Transceiver Interface

The 1553 transceiver interface signals include the serial channel A (primary bus) receive and transmit signal pairs *arxp* and *arxn,*

atxp_enc and *atxn_enc* and the serial channel B (secondary bus) signal pairs *brxp* and *brxn, btxp_enc* and *btxn_enc*. The channel A and B transmit inhibits, *atxinh* and *btxinh* feed to the transceiver, and the external transceiver inhibit inputs. *ext_tx_inh_a* and *ext_tx_inh_b* are used to disable the transmitter outputs from the ACE Flex-Core. The interface to a MIL-STD-1553 transceiver and a typical MIL-STD-1553 bus is illustrated in Figure 4.

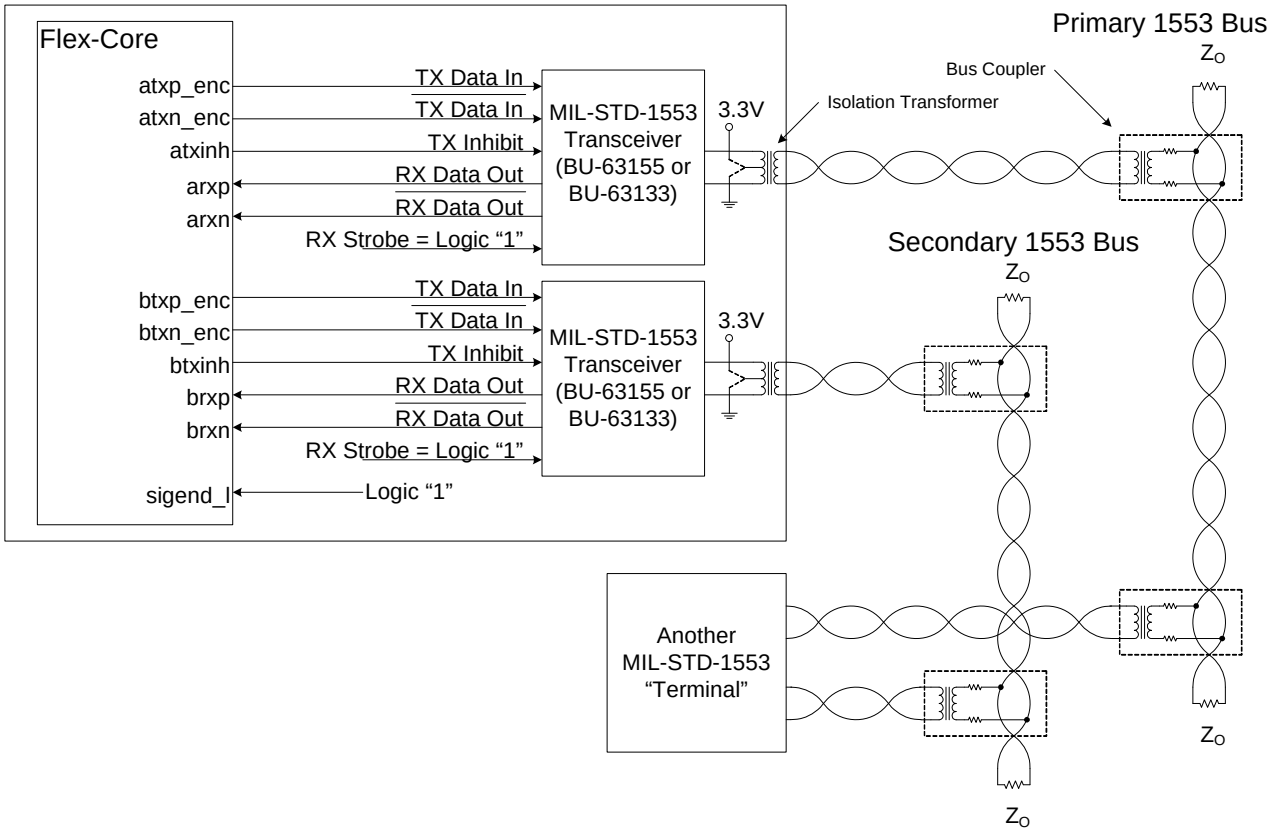


Figure 4: MIL-STD-1553 Transceiver Connections

The input signal *sigend_l* determines if the input from the 1553 receiver will be "single-ended". The single-ended option is generally not used with MIL-STD-1553 electrical buses. The most common use of the single-ended option is with a fiber optic transceiver to implement MIL-STD-1773 (optical version of MIL-STD-1553). In conventional MIL-STD-1553 applications the decoder within the Flex-Core will provide better noise immunity when the single-ended option is not enabled.

The output of a typical MIL-STD-1553 receiver provides two logic bits (RX and RX-not) to report three logic states of the bus to the decoder (active high, active low, and inactive). The three logic states are determined within the receiver through the use of two comparators; one with a positive threshold and one with a negative threshold. The

two comparators (i.e. three logic states) provide hysteresis to the decoder within the Flex-Core which provides for superior zero crossing detection.

3.5 Host Processor Interface and Control Logic

The Flex-Core provides a host interface bus to allow the host processor to read and write to Flex-Core's internal registers and the shared RAM.

The Flex-Core's shared RAM may be implemented using either a single port RAM (as illustrated in Figure 5) or using a dual port RAM (as illustrated in Figure 6).

If a single port RAM is used then the processor will access the RAM using the same interface as registers.The host processor must first arbitrate for access to the shared RAM bus (using the **target_rqst** and **target_grnt** signals) before reading or writing memory.

When a dual port RAM is used to implement the Flex-Core's shared RAM the host processor will access the shared RAM directly via the other port on the dual port RAM. It is still possible to utilize the host interface (and arbitration logic) to access the shared RAM, however, it will be much less efficient.

Regardless of the type of SRAM that is used to implement the Flex-Core's shared RAM, the host will still need to access the Flex-Core's internal registers through the host interface. The Host processor does not have to arbitrate to access the Flex-Core's internal registers.

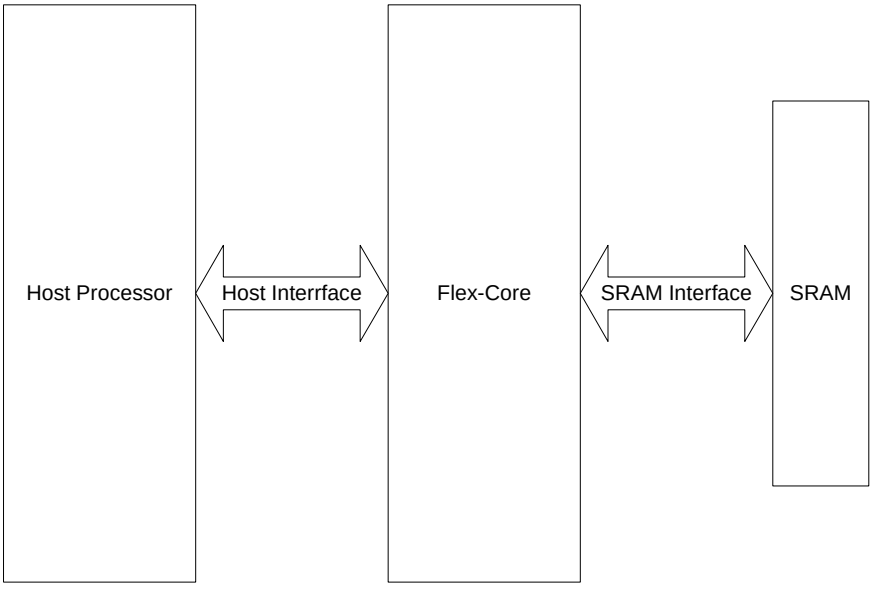


Figure 5: Flex-Core Host Interface with Single Port RAM

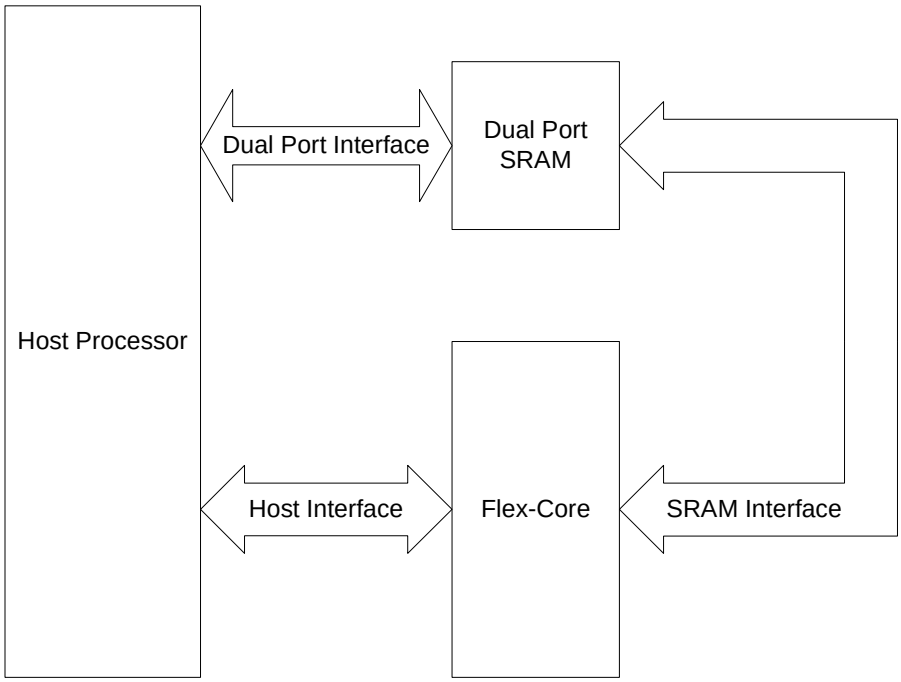


Figure 6: Flex-Core Host Interface with Dual Port RAM

3.6 Host Interface Timing

3.6.1 Host Memory Read and Write

The sections below provide illustrations of the timing between the host and the Flex-Core for memory read and write operations. In certain cases Setup and Hold timing markers have been included to indicate which clock edge the various input signals will be sampled on. The actual values for setup and hold time will be implementation dependent and should be addressed during static timing analysis.

The Flex-Core memory transfer interface is implemented using a pipelined architecture. All inputs to the Flex-Core (including the **target_request** input) are assumed to be synchronous to the rising edge of **clk**.

The host must first arbitrate for access to the internal shared RAM by asserting a logic “1” to the **target_request** input signal. The timing diagrams show the **target_request** signal being asserted following the rising edge on **clk**.

In response to an assertion of the **target_request** input the Flex-Core will assert **target_grnt** output to indicate when the host may access the shared RAM. The Flex-Core will hold the **target_grnt** output signal deasserted until the RAM can be accessed by the host. An uncontested access (i.e. an access in which there is no internal

transfer in progress or no higher priority request pending) will result in a minimum of two clock cycles to arbitrate access to the RAM.

The host processor must not hold the shared RAM bus (by keeping **target_request** asserted) for longer than 1 usec (as illustrated in Figure 7). Holding the shared RAM bus may deny the 1553 protocol/memory management unit access to required data structures that may result in 1553 protocol violations. To avoid this, the host should be forced to release **target_request** after a predetermined timeout threshold is reached.

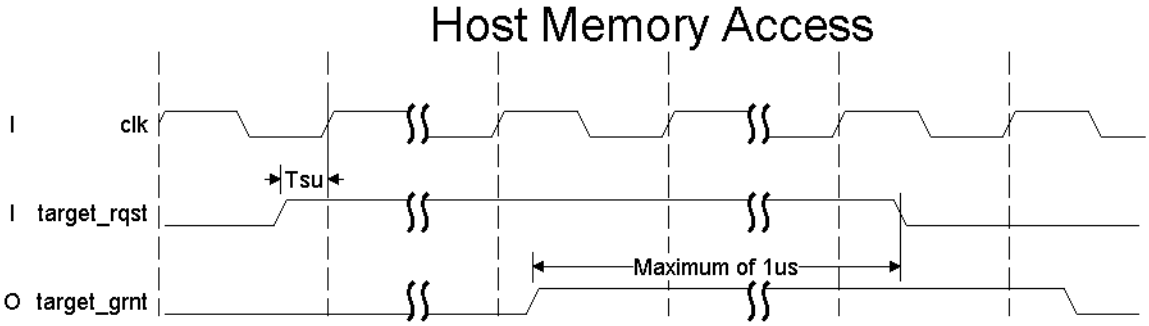


Figure 7: Maximum Host Access

The clock to the Shared RAM is assumed to be synchronous with the **clk** clock input to the Flex-Core in all below diagrams. However, the pipelined architecture of the RAM interface from the Flex-Core is designed such that a phase-delayed version of the **clk** input may be used to clock the SRAM, but this will complicate the static timing analysis process.

The **mem_reg_l** input signal to the Flex-Core is used to multiplex the memory interface between the Flex-Core's internal registers and the internal shared RAM. The **mem_reg_l** input must be setup (set to a "1") along with the desired memory address prior to the beginning of any read/write cycle to initiate a memory access.

The **read_en** and **write_en** inputs should not be asserted until after the **target_grnt** output has been driven to logic "1" by the Flex-Core. The system integrator must ensure that this condition is not violated and should take care here, since different implementations will result in various outcomes, such as additional bus latency and/or more difficult Static Timing Analysis.

Recommendations:

1. The user may implement a state machine that waits for a rising clock edge in which **target_grnt** is logic "1"

2. The **read_en** and **write_en** input signal may be gated with the **target_grnt** input signal through combinatorial logic.

Method 1 will add a clock cycle to the access time because the host must sample the **target_grnt** output before asserting the **read_en** or **write_en** signal. Method 2 will not require an additional clock cycle but does assume that **mem_reg_l** is valid with **read_en** or **write_en**. Additionally, method 2 is typically more difficult analyze using Static Timing analysis.

3.6.1.1 Host Memory Read

A single host memory read is illustrated in Figure 8. The memory read cycle will begin on the clock cycle in which both the **target_grnt** output from Flex-Core and the **read_en** input to Flex-Core are asserted. The Flex-Core will latch the **addr_host** and **mem_reg_l** input signals on the rising edge of the **clk** input.

The resulting read data will be valid on the third clock cycle after the **addr_host** and **mem_reg** signals are latched with both **target_grnt** and **read_en** asserted.

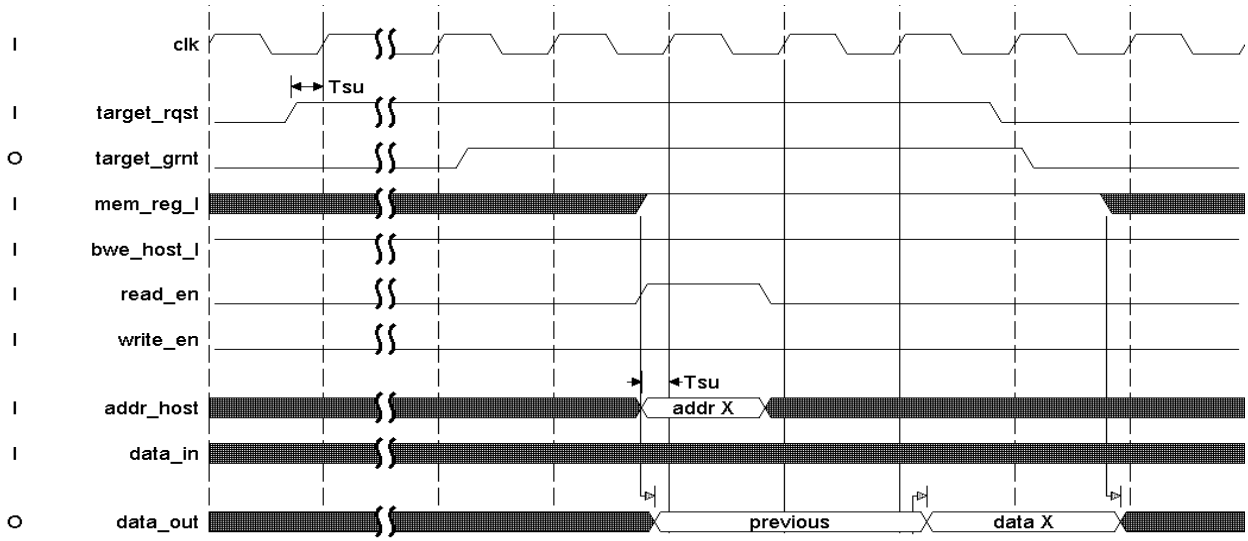


Figure 8: Host Single Word Memory Read

3.6.1.2 Host Memory Write

Figure 9 provides an illustration of the timing between the host and the Flex-Core for a single memory write. The memory write cycle will begin on the first clock cycle in which **target_grnt** and **write_en** are asserted. The Flex-Core will latch the **addr_host**, **mem_reg_l**, **data_in**, and **bwe_host_l** input signals on the rising edge of the clock. If a single Memory write is to be performed, the **write_en** should only be held active for one clock cycle. Each additional clock cycle where **write_en** is asserted with perform another write cycle to the memory.

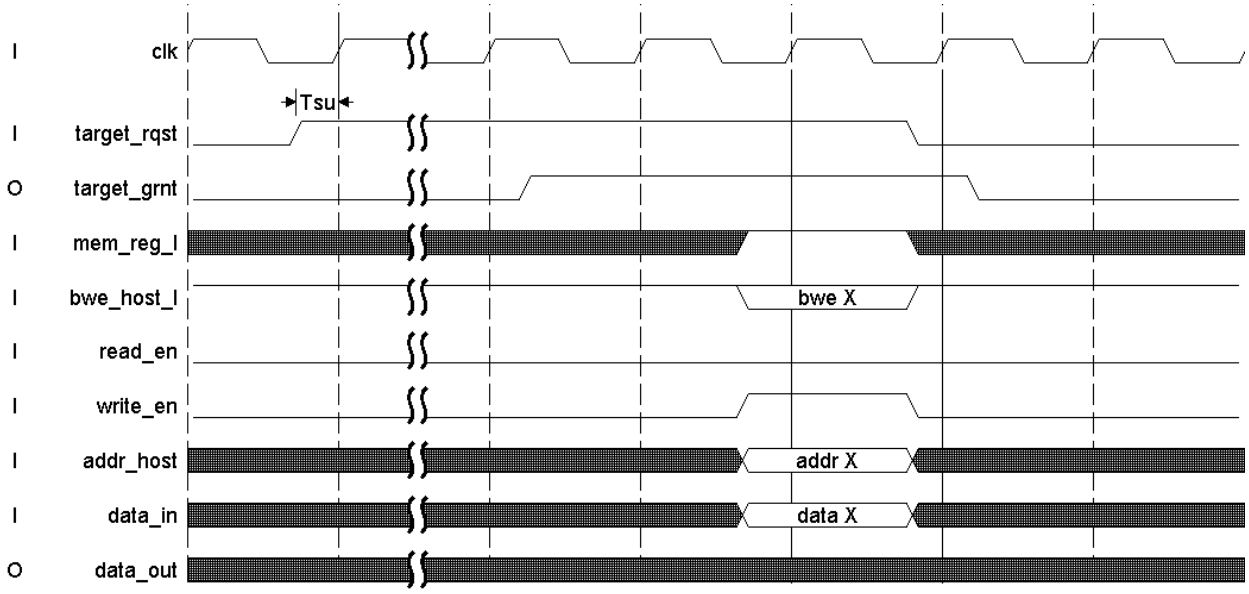


Figure 9: Host Single Word Memory Write Table

3.6.1.3 Host Memory Read Multiple

A multiple host memory read is illustrated in Figure 10. The memory read cycle will begin on the first clock cycle in which both the **target_grnt** output from Flex-Core and the **read_en** input to Flex-Core are asserted. The Flex-Core will latch the **addr_host** and **mem_reg_l** input signals on the rising edge of the **clk** input. Each additional clock cycle with **read_en** and **target_grnt** asserted with latch the next address to read from the **addr_host** signal.

The resulting first read data will be valid on the second clock cycle after the read cycle began with both **target_grnt** and **read_en** asserted. Each additional clock cycle will output the next address data.

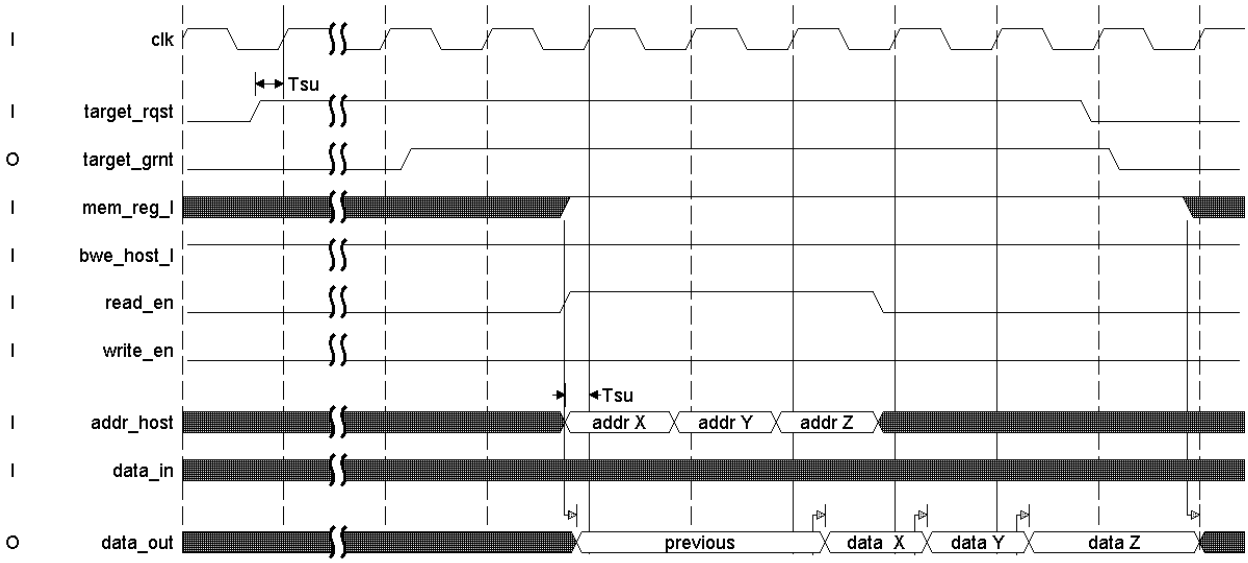


Figure 10: Host Memory Read Multiple

3.6.1.4 Host Memory Write Multiple

Figure 11 provides an illustration of the timing between the host and the Flex-Core for a multiple memory write. The memory write cycle will begin on the first clock cycle in which **target_grnt** and **write_en** are asserted. The Flex-Core will latch the **addr_host**, **mem_reg_l**, **data_in**, and **bwe_host_l** input signals on the rising edge of the clock. Each additional clock cycle with **write_en** and **target_grnt** will latch a new value of **addr_host**, **mem_reg_l**, **data_in**, and **bwe_host_l** to perform another write cycle.

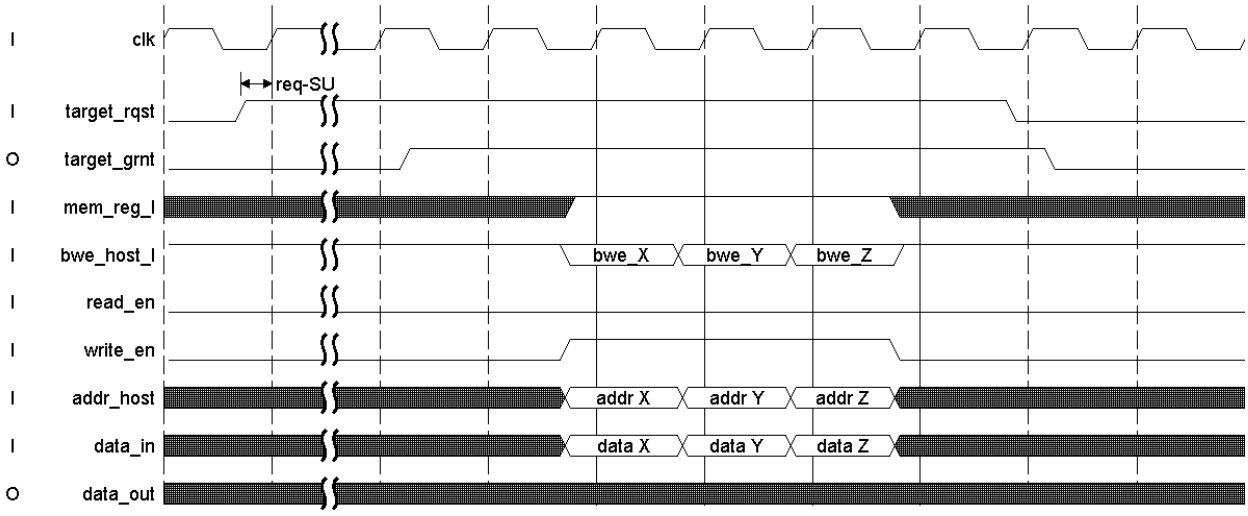


Figure 11: Host Memory Write Multiple

3.6.2 Host Register Read and Write

The below sections provide illustrations of the timing between the host and the Flex-Core for register reads and writes. The actual values for

setup and hold time will be implementation dependent and should be addressed during static timing analysis.

Unlike memory transfers, the host does not have to arbitrate for access to the Flex-Core's internal registers (i.e. the host is not required to assert the **target_rqst** input and wait for the **target_grnt** output). Registers are accessible at any time.

3.6.2.1 Host Register Read

A single host register read is illustrated in Figure 12. The register read cycle will begin on the clock cycle in which the **read_en** input is asserted to the Flex-Core. The Flex-Core will latch the **addr_host** and **mem_reg_l** input signals on the rising edge of the **clk** input.

The resulting read data will be valid on the next clock cycle after the read cycle began with **read_en** asserted.

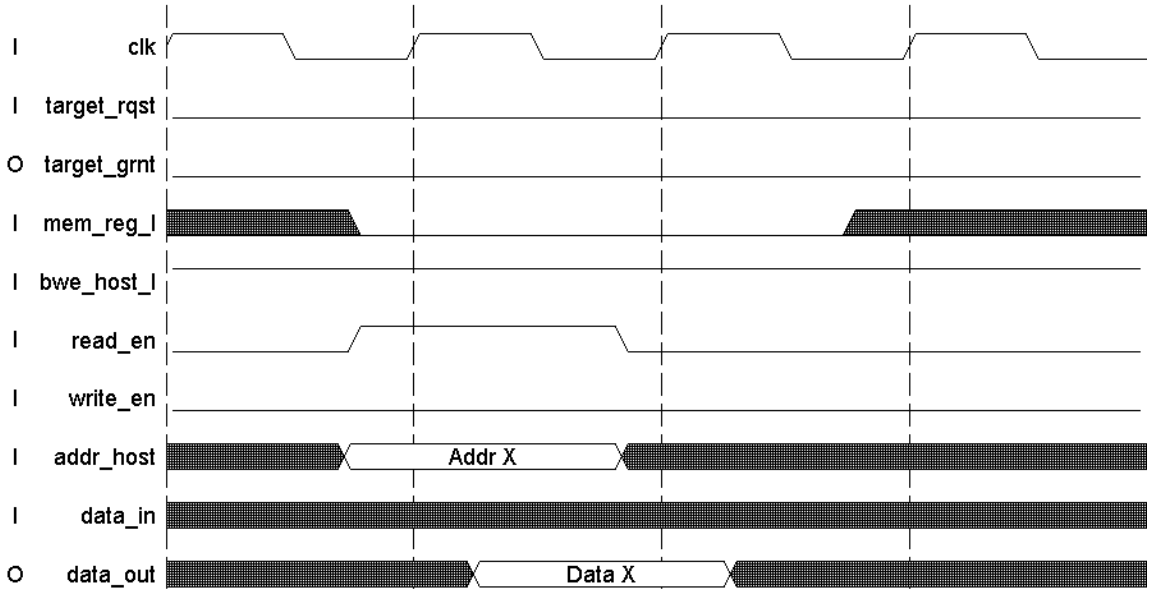


Figure 12: Host Register Read

3.6.2.2 Host Register Write

Figure 13 provides an illustration of the timing between the host and the Flex-Core for a single register write. The register write cycle will begin once **write_en** is asserted. The Flex-Core will latch the **addr_host**, **mem_reg_l**, **data_in**, and **bwe_host_l** input signals on the rising edge of the clock. If a single register write is to be performed, the **write_en** should only be held active for one clock cycle. Each additional clock cycle where **write_en** is asserted will result in an additional write cycle.

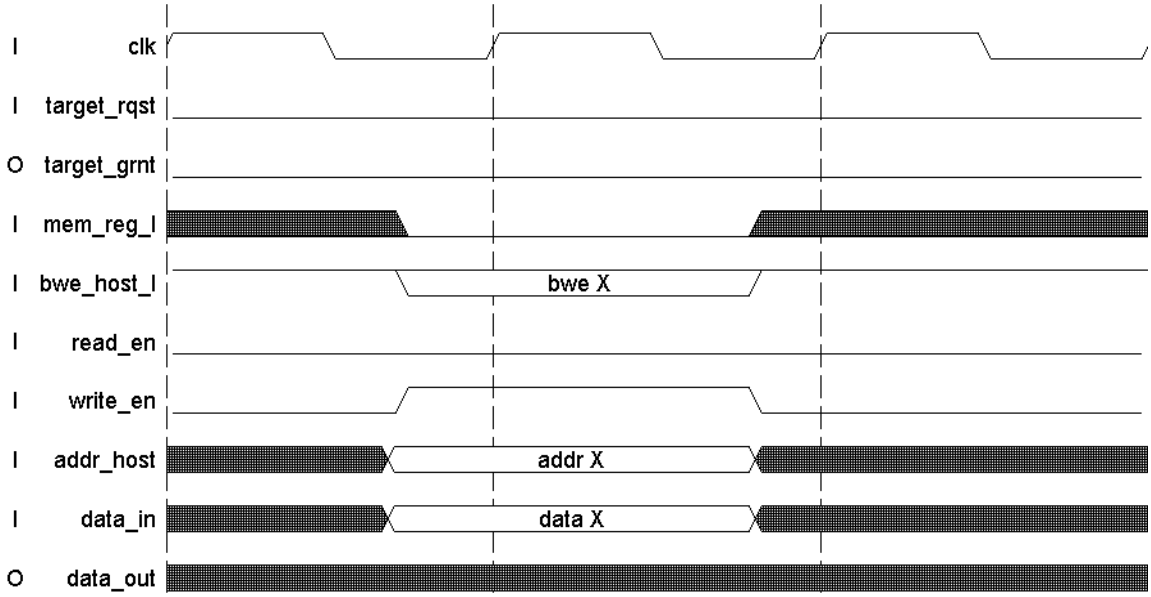


Figure 13: Host Register Write

3.7 Shard RAM Interface Timing

The shared RAM is used by the Flex-Core's memory management unit (MMU) to buffer transmit and receive data from the MIL-STD-1553 bus. In addition the shared RAM will contain the various data structures used by the Flex-Core's MMU (such as command stacks and look-up tables).

3.7.1 Internal Flex-Core Read and Write

The following is a description of the signal flow involved during a flex-core to shared memory write operation. Flex-core performs both 16-bit and 32-bit memory write operations. The various signals involved in this are listed in section 3.2.5.

The Flex-Core block asserts the **data_en**, **adsc_l,** the desired memory address (**addr_mem**), the data to be written (**data_out_mem[35:0]**), and the desired byte enables (**bwe_l [3:0]**) for each write cycle. The **oe_l** signal remains deasserted during the entire write operation.

Data on **data_out_mem[35:0]** signal is arranged as follows:

Data_out_mem[7:0]	-First byte
Data_out_mem[15:8]	-Second byte
Data_out_mem[23:16]	-Third byte
Data_out_mem[31:24]	-Fourth byte
Data_out_mem[32]	-Parity bit for first byte
Data_out_mem[33]	-Parity bit for second byte
Data_out_mem[34]	-Parity bit for third byte
Data_out_mem[35]	-Parity bit for fourth byte

The **bwe_mem_l[3:0]** bit signals are used to signify which of the bytes are valid. They are active low signals. The four bits correspond to each of the four bytes in the data to be written, as follows

bwe_mem_l[0]	-First byte
bwe_mem_l[1]	-Second byte
bwe_mem_l[2]	-Third byte
bwe_mem_l[3]	-Fourth byte

memory_data[35:0] signal remains invalid throughout the operation.

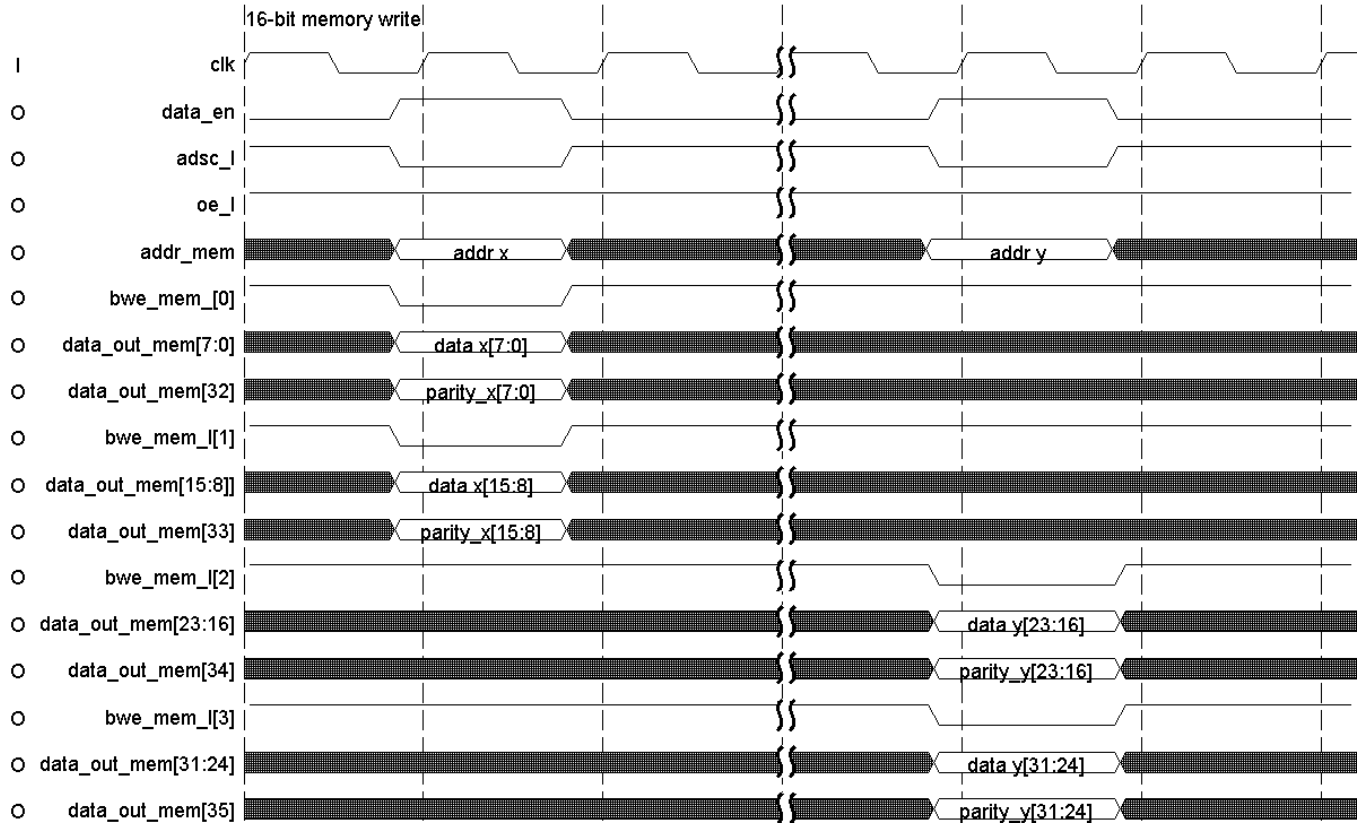


Figure 14: Shared Memory Write Operation

3.7.2 Shared RAM Single Word Write

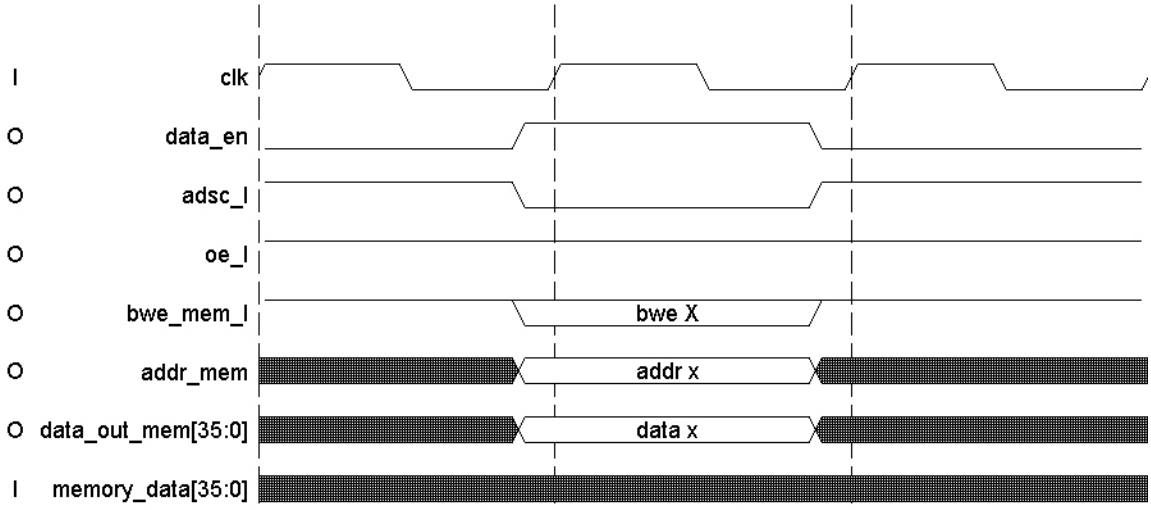


Figure 15: Shared RAM Write Operation

3.7.3 Shared RAM Single Word Read

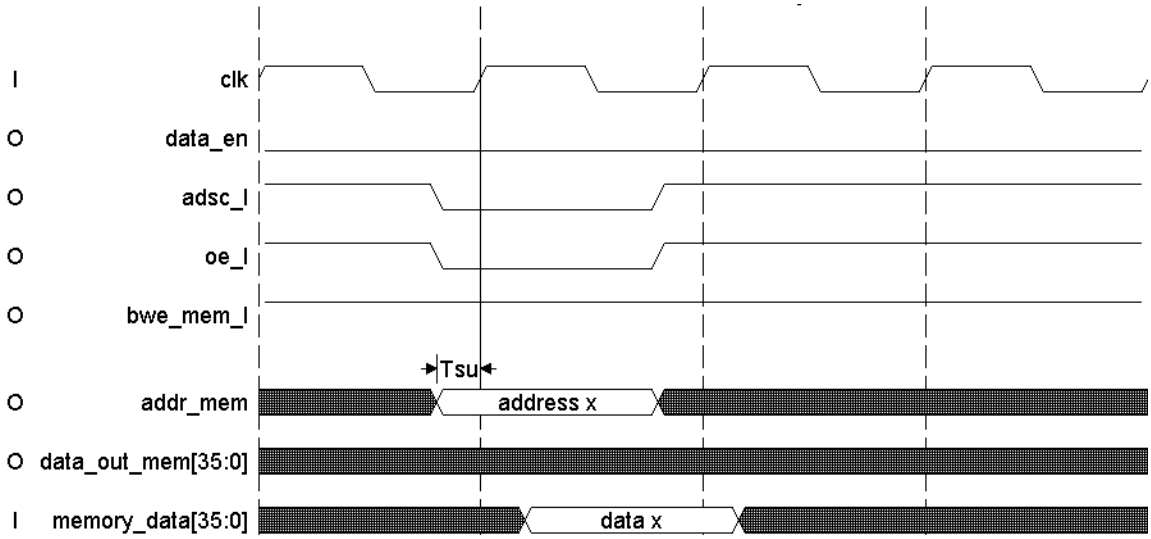


Figure 16: Shared RAM Read Operation

3.7.4 Shared RAM Multi-Word Write

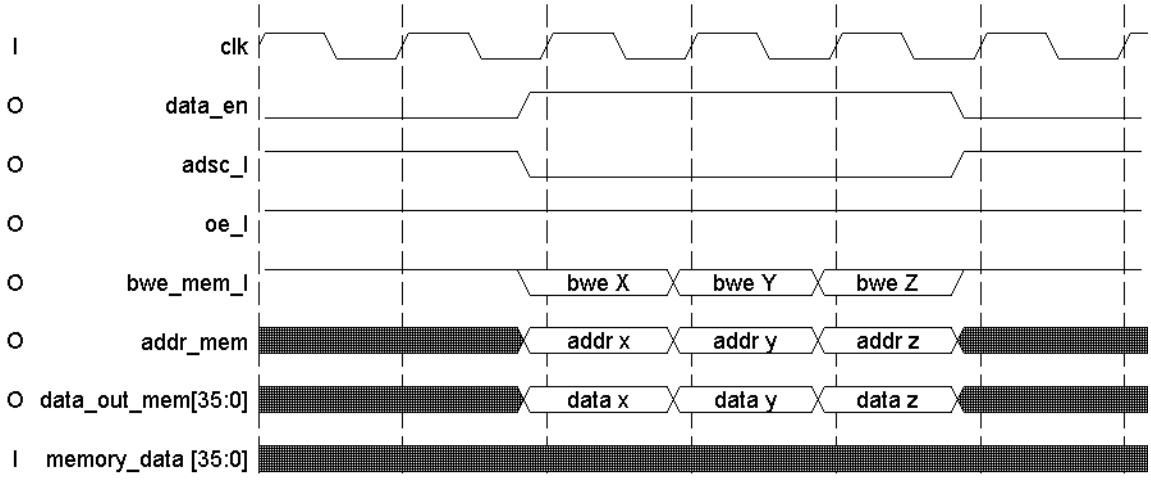


Figure 17: Shared RAM Multi-Word Write Operation

3.7.5 Shared RAM Multi-Word Read

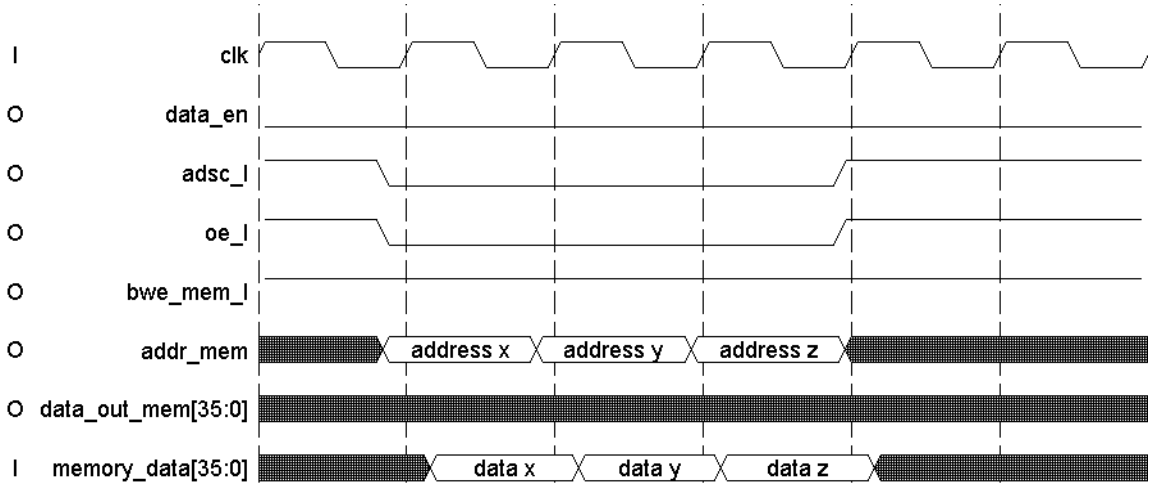


Figure 18: Shared RAM Multi-Word Read Operation

3.8 Read RAM Interface Timing

3.8.1 Functional Description of Read RAM

The following figure gives a detailed illustration of the timing of various signals of the read_ram block. The read_ram block has inputs **clkout_ram**(clock input), **read_ram_add[3:0]** (4 bit address), **read_ram_we**(write enable), **memory_data[31:0]**(32 bit data input) and output **dout[31:0]**(32 bit data output).

Write cycles occur upon sampling of **read_ram_we** asserted. On each rising clock edge with **read_ram_we** asserted, the data on **memory_data** is written to address **read_ram_addr**.

The **dout** bus is always active and will output the data associated with the address sampled on **read_ram_addr** upon the rising edge of **clkout_ram**.

3.8.2 Read RAM Timing

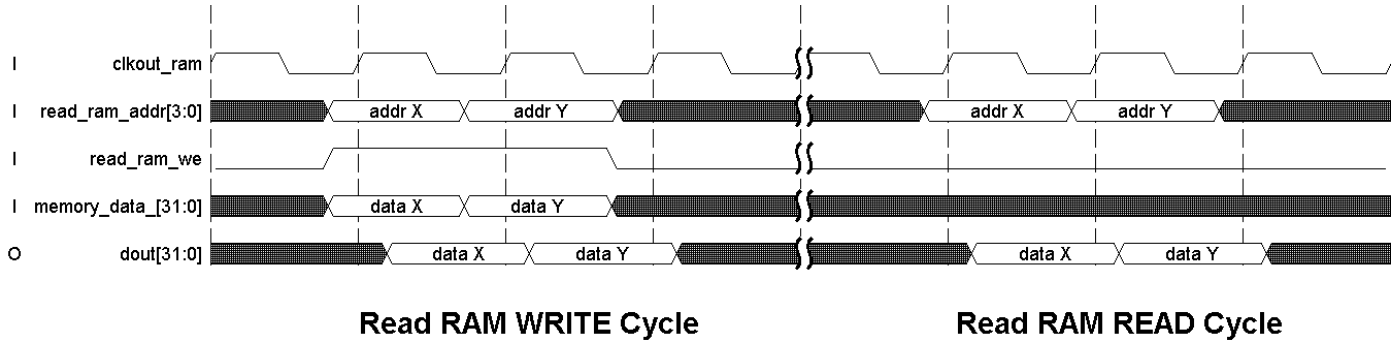


Figure 19: Read Ram Timing

3.9 RAM Buff Interface Timing

3.9.1 Functional Description of RAM Buff

3.9.2 RAM Buff Timing

3.10 Flex ROM Interface Timing

3.10.1 Functional Description of Flex ROM

3.10.2 Flex ROM Timing

4 SIMULATION

4.1 The ACE Flex-Core Test Bench

The ACE Flex-Core Test Bench supports the VHDL 1993 language syntax.

The ACE Flex-Core Test Bench block is connected to the ACE Flex-Core I/O so that it can simulate inputs and verify outputs.

One of the main functions of Test Bench is to read, interpret and execute lines of vectors found in test register files. Each test register file can be designed to test a specific ACE Flex-Core function. This makes it easy to modify testbench routines, without a large investment in VHDL simulation environment design.

The number of tests run on the Design Under Test (DUT) is selectable using a selection file */sim/vectors/vectors_select.vec*.

Upon completion of the testbench run, a *vectors_test.rpt* file is created that contains a log of the tester run with all vectors run, tests passed, comments dictated in the vector files, and any failures.

4.1.1 Test Bench Hierarchy

The test bench hierarchy uses the same major entity names regardless of which IP block and/or whether the IP source or netlist is being implemented.

1) **flexcore_top** – This is the top-level DUT

2) **flexcore_top_tb** – This is the top-level testbench entity which connects the DUT and the Tester.

3) **flexcore_tb** – This is the entity which contains the tester

4) **flex_pkg** – This is the VHDL package which contains the tester functions used by the *flexcore_tb*.

5) **vectors_select.vec** – This is the file that selects which test vectors are to be run during the simulator

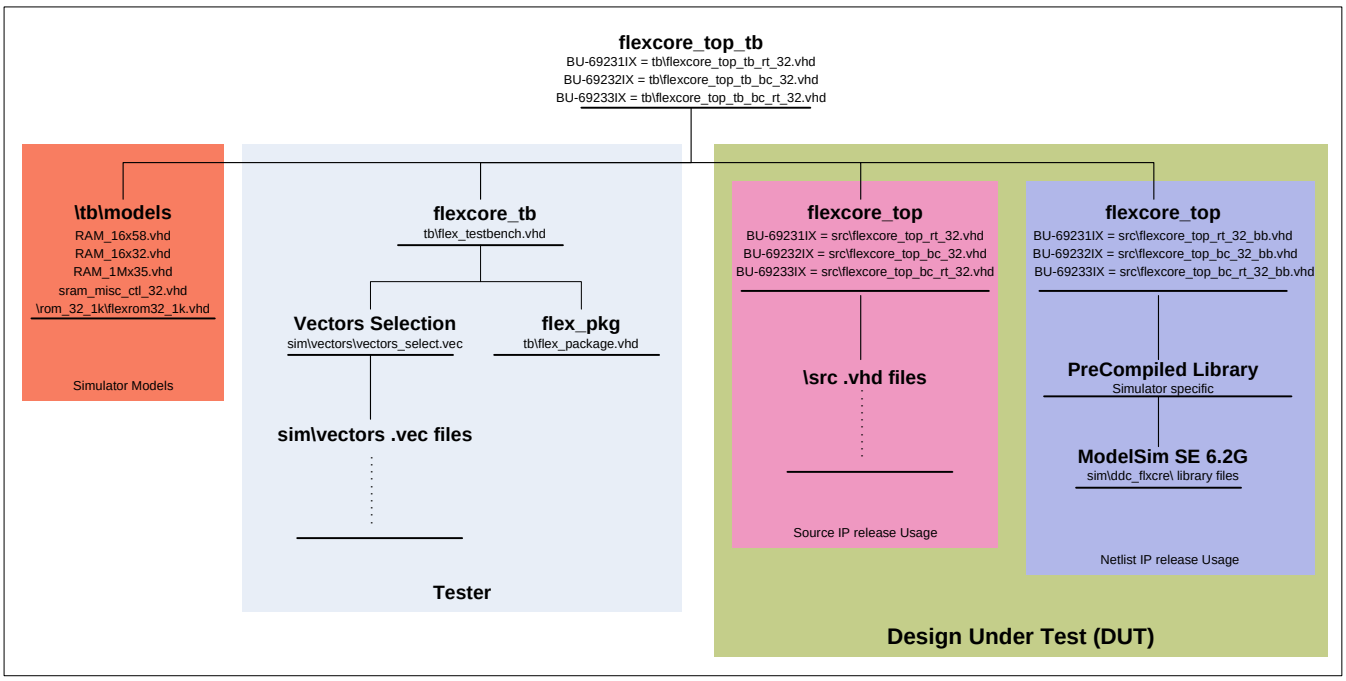


Figure 20: Test Bench Hierarchy

4.2 Vector File Descriptions

There are various vector files distributed in the "\sim\vectors" directory. These files have been designed to provide maximum coverage of the ACE Flex-Core design during simulation. The user can at any time create their own custom vector files by as long as they adhere to the proper architecture of the vector file.

4.2.1 Vector Select file

The TestBench utilizes the \sim\vectors\vectors_select.vec file to allow the user to select which vector files are to be run during the simulation run.

The file starts with a vector select line and ends with and End-Of-File delineator "Z"

When a "*" character is encountered, all information past it is ignored. This should be used to enter any comments in the file.

The vector select line format is as follows:

Usage: *X desired_vector_name Y/N*

Table 17: Vector Select command

Column	Description
1	**X command**
2	**Vector Name** This is the vector file name without the .vec file extension
3	**RUN File** This field selects whether the file is run or not **Y** - Run the vector file during the simulation run **N** - Ignore the vector file

4.2.2 Included Vector Files

Below is a list of available vector files included with the release. Some files require both a stimulus and results file, which checks the output to expected values. Stimulus files that have an associated results file are indicated within Table 18.

Table 18: Vector Files

Stimulus File Name	Results File	Mode Checked	Description
bc_02.vec	bc_response_02.vec	BC	
bc_03.vec	bc_response_03.vec	BC	
bc_rt_mode_code_rst_incmd.vec	N/A	BC/RT combined	
bc_only_incmd.vec	N/A	BC	
rt_only_mode_code_rst_incmd.vec	N/A	RT	
bcexttxinh.vec	N/A	BC	
rtexttxinh.vec	N/A	RT	
interrupt.vec	N/A	All	
memory_02.vec	N/A	All	Test selected memory locations of Shared RAM to proper interface connections
mt_error.vec	N/A	MT	
ram_test.vec	N/A	RT	Test the RT Protocol RAM (Destructive)
register_01.vec	N/A	All	
rt_13.vec	N/A	RT	
rt_address.vec	N/A	RT	
rtboot.vec	N/A	RT	
sync_mode.vec	N/A	RT	

4.3 Vector File Syntax

The following describes the format of the vector files used by Flex-core IP testbench and software drivers to perform simulation tests for verification.

The Byte formatting is as follows for all Hex format values in ASCII used in the following example.

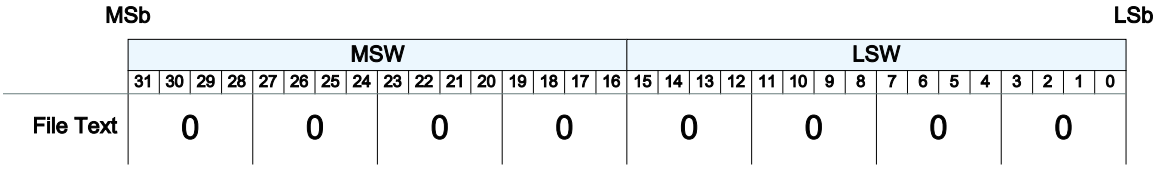


Figure 21: Byte formatting for Hex format in ASCII

NOTE: There are 2 space characters between each column.

4.3.1 Vector File Structure

All vector files start with at least one vector line and end with an End-Of-File delineator.

4.3.2 Vector Command Formats

Each vector type has a specific format. Below is a description of each type and the associated field descriptions.

Table 19: Vector Command Formats

Command Character	Number of Data Columns	Description
*	N/A	**Line Comment** Ignore all characters beyond this character
D	1	**Screen/Log Output** Print text following command to simulator screen output and Log file.
Z	0	**End of File** Denotes the End of the vector file
Y	1	**Delay Vector** Wait at vector for number of clock cycles dictated by data column
P	1	**Data Bus Width Select** Configure the Data bus width as 16-bit or 32-bit to the Device under test

Command Character	Number of Data Columns	Description
U	4	**Host Memory Access** Perform a read/write to the host memory.
V	5	**Device Memory/Register Access** Perform a read/write to the device under test's memory or registers.

4.3.2.1 Screen/Log Output (D Type)

This character delineates a Comment line. The Comment text included in column 2 of this vector will be printed to the simulator output screen and the *vectors_test.rpt* file. The format for this style is as follows:

D (two spaces) Comment Text

Usage: *D This is the comment*

4.3.2.2 End of File (Z Type)

This character delineates the end of a Vector file. It must be used to instruct the vector engine to stop scanning the file.

Usage: *Z*

4.3.2.3 Delay (Y Type)

This character delineates a Delay. It utilizes two columns of data.

Usage: *Y 00000002*

'Y' Type Control Structure

Column Number	1	2	2	2	2	2	2	2	2	(Optional) '*' + Comments
	Y	Delay Decimal Digit 8	Delay Decimal Digit 7	Delay Decimal Digit 6	Delay Decimal Digit 5	Delay Decimal Digit 4	Delay Decimal Digit 3	Delay Decimal Digit 2	Delay Decimal Digit 1	

Figure 22: 'Y' Type Control Structure

Table 20: Delay (Y Type)

Column	Description
1	**Y command**
2	**Delay Value (Decimal Value)** Number of IP core clock cycles to delay the simulation

4.3.2.4 PCI Configuration (P Type)

This character delineates a PCI configuration of 16-bit or 32-bit wide transfers. It utilizes two columns of data.

Usage: *P 0 * 16-bit data bus selected*

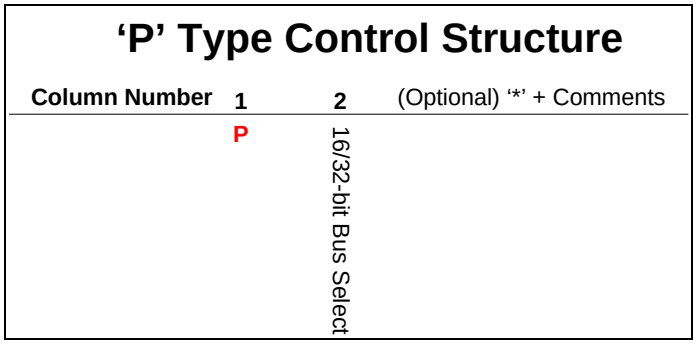


Figure 23: 'P' Type Control Structure

Table 21: PCI Configuration (P Type)

Column	Description
1	**P command**
2	**Bus selection (Binary Value)** **0** – 16-bit data bus. 16-bit data bus gets parsed/concatenated from the 32-bit data vector. **1** – Binary number indicating 32-bit data bus.

4.3.2.5 Host Memory Access (U Type)

This character delineates a Host Memory Read/Write command. It utilizes 5 columns of data.

Usage: *U R 00000014 01234567 * Check that 0x01234567 is read at 0x00000014*

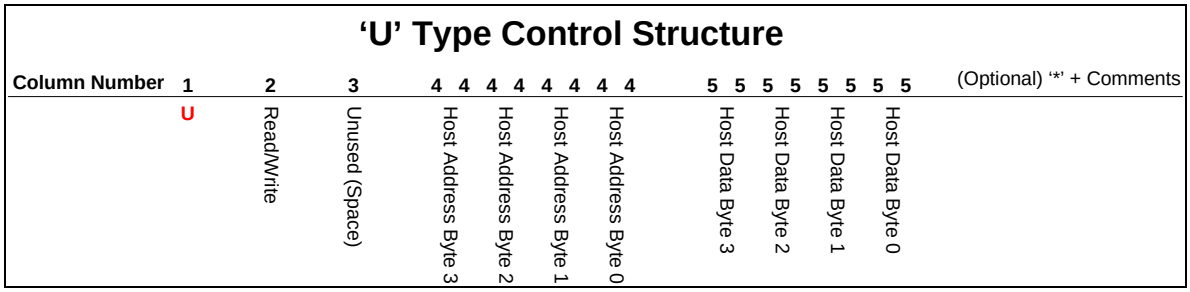


Figure 24: 'U' Type Control Structure

Table 22: Host Memory Access (U Type)

Column	Description
1	**U command**
2	**Read/Write Select** **R** – Host Memory Read Command **W** – Host Memory Write Command
3	**Unused** Leave blank with a space character
4	**Offset Address** (Hex value) using Byte Addressing. This value must address upon a DWORD oriented value (i.e. 0x00000004, 0x00000008, …)
5	**Data DWORD** *Read command* = Expected Data upon memory read *Write command* = Write data to location addressed by column 4

4.3.2.6 Device Memory/Register Access (V Type)

This character delineates a Device Memory/Register Read/Write command. It utilizes 6 columns of data.

Usage: *V W R 0000003 00000001 * Set the SW Rst bit in register 0x3*

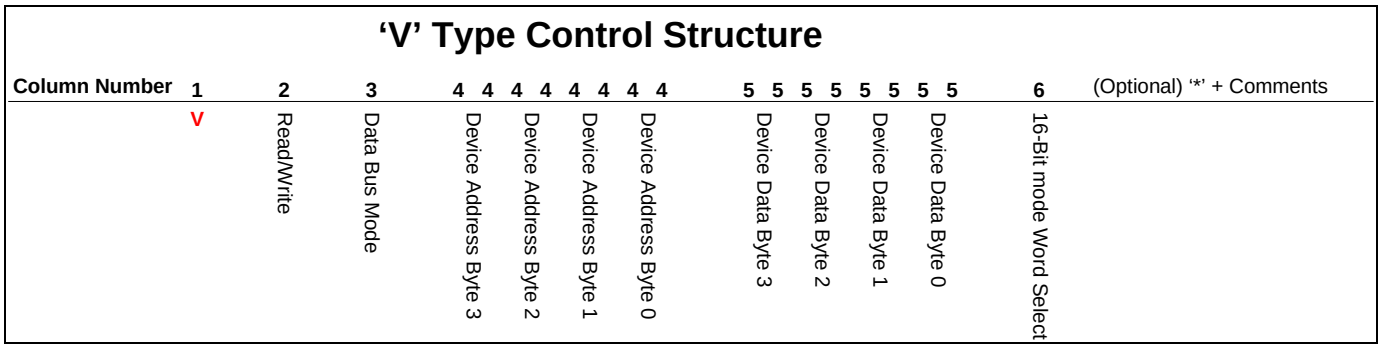


Figure 25: 'V' Type Control Structure

Table 23: Device Memory/Register Access (V Type)

Column	Description
1	**V command**
2	**Read/Write Select** **R** – Device Memory/Register Read Command **W** – Device Memory/Register Write Command
3	**Data Bus Mode** **M** – Flex-Core Shared Memory **R** – Flex-Core Registers **F** – Flash (If Implemented)
4	**Offset Address** Hex value using DWORD Addressing to select device Memory/Register location.
5	**Data DWORD** *Read command* = Expected Data upon memory/register read *Write command* = Write data to location addressed by column 4
5	**16-Bit Word Select** **1** – Use only Least-Significant-Word and ignore Most-Significant-Word in 32-bit vector. **2** – Ignore the Least-Significant-Word and use only Most-Significant-Word in 32-bit vector. ***Note:*** *used only if P command, column 2 indicates 16-bit data, and Data Bus Mode is set to Flex Core Shared Memory. This column is ignored for all other cases.*

4.4 Running Custom Vector Files

To implement the usage of custom vector files the testbench VHDL file must be updated or a file name already implemented may be utilized for easier implementation.

4.5 Simulation Project File

Please reference Figure 20, which graphically depicts the Test bench configuration for both options.

The /vectors directory must exist one level below the simulator project directory.

For proper simulation, the RAM/ROM models must be compiled and are connected to the IP core in the Top-Level testbench file.

4.5.1 Source VHDL IP Block

To run simulations on the VHDL source code, all files existing within the \src directory must be compiled into the simulator.

A sample ModelSim simulation project file is included with the design. *flexcorebcrt.mpf* is located in the *sim* directory. It is required to have Xilinx primitive libraries on the simulator machine to compile the some simulation models.

Script Files for source simulations

The *sim**scripts* directory contains various sample ModelSim simulation script files intended for use when compiling and simulating the IP source. Each script file is designed to simulate one of the available architectures of ACE Flex-Core.

Table 24 lists available script files:

Table 24: Source Simulation script Files

Simulation File	Resulting Action
compile_vhdl_bc_16.do	compiles BC Only 16-bit core
compile_vhdl_bc_rt_16.do	compiles BC/RT/MT 16-bit core
compile_vhdl_rt_16.do	compiles RT/MT 16-bit core
compile_vhdl_bc_32.do	compiles BC Only 32-bit core
compile_vhdl_bc_rt_32.do	compiles BC/RT/MT 32-bit core
compile_vhdl_rt_32.do	compiles RT/MT 32-bit core
compile_vhdl_bc_rt_multi.do	compiles BC/Multi-RT/MT 32-bit core
compile_vhdl_rt_multi.do	compiles Multi-RT /MT 32-bit core
runsim_all.do	run simulation timing analysis

4.5.2 ModelSim compiled simulation library

When the source VHDL code is not available, simulations may be run using the compiled simulation library. Currently only ModelSim 6.x is supported.

The DDC Simulator library is located in the *sim**ddc_flxcre* directory and is named **ddc_flxcre**.

The DDC Simulator library should be added to the ModelSim project by adding the following line to the .mpf file:

ddc_flxcre = ./ddc_flxcre

Upon loading the project, the ddc_flxcre library will be listed in the ModelSim library listing.

Script Files for simulations using Compiled Libraries

The *sim**scripts* directory contains various sample ModelSim simulation script files intended for use when compiling and simulating the Compiled libraries. Each script file is designed to simulate one of the available architectures of ACE Flex-Core.

Table 24 lists available script files:

Table 25: Netlist Simulation script Files

Simulation File	Resulting Action
compile_top_bc_16.do	compiles BC Only 16-bit core
compile_top_bc_rt_16.do	compiles BC/RT/MT 16-bit core
compile_top_rt_16.do	compiles RT/MT 16-bit core
compile_top_bc_32.do	compiles BC Only 32-bit core
compile_top_bc_rt_32.do	compiles BC/RT/MT 32-bit core
compile_top_rt_32.do	compiles RT/MT 32-bit core
compile_top_bc_rt_multi.do	compiles BC/Multi-RT/MT 32-bit core
compile_top_rt_multi.do	compiles Multi-RT /MT 32-bit core
runsim_all.do	run simulation timing analysis

5 DESIGN USAGE

5.1 Core Release Options

DDC offers the Flex-Core IP in either a source code or netlist release.

The Netlist format has shown to reduce implementation times and result in a more reliable build process.

Please contact your local DDC sales representative to discuss the available options further.

5.2 IP Block implementation

The IP block must be connected as per the usage case based on Architecture and options. This may require internal RAM, ROM, IO Buffers, PLL's, etc.

Please refer to section 3 for specific information based on each IP Block IO port group.

5.2.1 Interface requirements to external SRAM

External SRAM typically has only a single bi-Directional Data bus. To interface to this type of memory, it is required that the **memory_data[35:0]** and **data_out_mem[35:0]** be combined into a bidirectional port (called **shared_ram_data[35:0]** for this example). A sample file (***\tb\models\sram_misc_ctl_32.vhd***) that creates this is provided with the release.

The following table denotes the IO direction based on the Shared RAM control signals output from the IP Block.

Table 26: Shared Ram IO Direction

Signal Name	Input/Output
ram_size[6:0]	Input
memory_data[35:0]	Input
data_en	Output
bwe_mem_l[3:0]	Output
oe_l	Output
adsc_l	Output
addr_mem[18:0]	Output
data_out_mem[35:0]	Output

5.3 VHDL Code Synthesis

The VHDL code is a single clock domain synchronous design with no latches utilized.

The signal clk is the only signal in the ACE Flex-Core that is connected to the clock of a D flip-flop. Except for the cases described below, all logic is positive edge clock triggered.

There are two instances where the falling edge of clk is used:

1) The *flex_DECODER* module, where both the rising and falling edges of the clock are used to sample, or oversample, the digital 1553 Manchester input signals. The higher sampling frequency provides improved tolerance for input zero crossing distortion and is therefore a very desirable feature.

2) The *prog_state_machine* module, where both the rising and falling edges of the clock are used to speed up the loading of protocol ROM data on device power-up.

The reset signal is a single Asynchronous reset.

No IO Buffers have been instantiated within the IP core. This alleviates architecture specific entity primitives.

No Delay elements are utilized.

5.3.1 Synthesis Files

The following table dictates the Top-Level VHDL RTL file and entity name for each IP core part number. All files existing within the /src directory are required to be included within the synthesis project for a build.

Sample Synplicity synthesis scripts are included within the /syn directory. The following table lists all files available based upon which core package is obtained.

Table 27: Source Synthesis script Files

Simulation File	Resulting Action
src_compile_bc_16.tcl	compiles BC Only 16-bit core
src_compile_bcrt_16.tcl	compiles BC/RT/MT 16-bit core
src_compile_rt_16.tcl	compiles RT/MT 16-bit core
src_compile_bc_32.tcl	compiles BC Only 32-bit core
src_compile_bcrt_32.tcl	compiles BC/RT/MT 32-bit core
src_compile_rt_32.tcl	compiles RT/MT 32-bit core
src_compile_bcMrt_32.tcl	compiles BC/Multi-RT/MT 32-bit core
src_compile_Mrt_32.tcl	compiles Multi-RT /MT 32-bit core

Table 28: Top-Level VHDL RTL IP Core

	BU-69231IX	BU-69232IX	BU-69233IX
File Name	flexcore_rt_top_32.vhd	flexcore_bc_top_32.vhd	flexcore_bc_rt_top_32.vhd
Entity Name	flexcore_top	flexcore_top	flexcore_top

5.3.2 Synthesis Constraints

It is recommended that the core be synthesized to a higher frequency than expected to run. This results in more reliable synthesis runs and gives more headroom to the Physical implementation tools.

5.3.3 Physical Implementation Constraints

The only Physical Implementation constraint that is required is that clk is set at 40MHz or greater for the tools to check timing and ensure that the design will operate at the required frequency.

The constraints implemented during the Synthesis phase are typically passed to the physical implementation tools. Although the IP is typically synthesized to a higher clock frequency than required thus this may be inefficient. Overriding the clock requirement passed from synthesis may result is better utilization numbers.

5.4 Netlist Implementation

5.4.1 Synthesis Files

No synthesis files are required for the Physical IP core in this case. The only requirement is that the core be instantiated in the application as a black box entity.

Sample files have been included in the *src* directory with the core that instantiate the black box entity and create a top-level entity that complies with the standard ***flexcore_top*** naming. These sample files may be used in the final application to instantiate the IP in the design, as no logic exists within them that are not synthesizable.

Table 29: Sample Synthesis Files

	BU-69231IX	BU-69232IX	BU-69233IX
File Name	flexcore_rt_top_32_bb.vhd	flexcore_bc_top_32_bb.vhd	flexcore_bc_rt_top_32_bb.vhd
Entity Name	flexcore_top	flexcore_top	flexcore_top
Black Box Name	ddc_flxcre_rt_32_top	ddc_flxcre_bc_32_top	ddc_flxcre_bc_rt_32_top

5.4.2 Synthesis Constraints

No specific clock constraints exist in this implementation, as the core has already been synthesized to the required clock frequencies and has passed synthesis timing verification

5.4.3 Physical Implementation Files

The *\netlist* directory contains the physical implementation block that must be included during the Place and route process (i.e. Xilinx ISE translate phase).

For Xilinx implementations using the Xilinx ISE tools, the Macro Search Path must be updated to include the *\netlist* directory. See Figure 26 below.

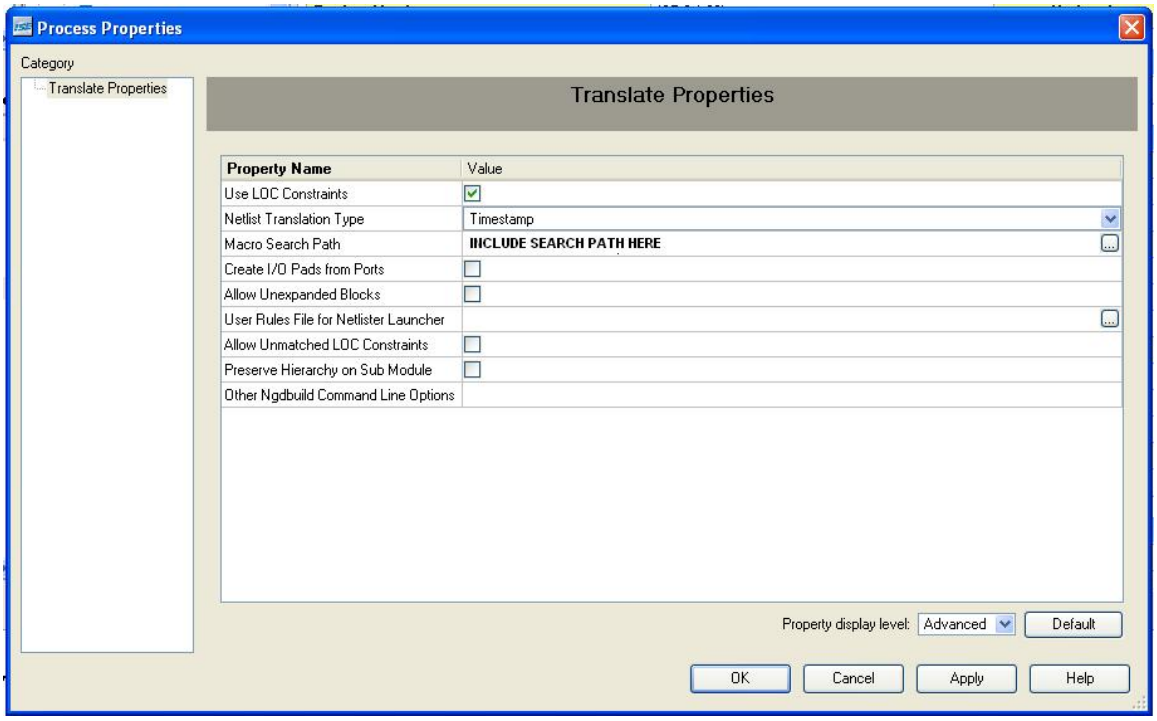


Figure 26: Xilinx Macro Search Path Entry

5.4.4 Physical Implementation Constraints

The only Physical Implementation constraint that is required is that clk is set at 40MHz or greater for the tools to check timing and ensure that the design will operate at the required frequency.

6 APPENDIX C – PCI / DMA INTERFACE

6.1 PCI Bus Interface block with Host Initiated DMA Controller

ACE Flex-Core supports host initialed DMA transfers. This capability was incorporated into a 66 MHz, 32-Bit PCI Target interface for use with the Xilinx PCI Core.

DDC does not provide the Xilinx core, rather we provide a wrapper that utilizes the Xilinx core. The *flex_pci32**src* directory contains the "*pci32_top.vhd*" module. This module will support 8/16/32-bit PCI transfers from the Xilinx PCI interface but ACE Flex-Core Intellectual Property only supports 16/32-bit PCI transfers.

The following tables define the signal I/O from *pci32_top*:

Table 30: PCI Bus Signals (to-from PCI Bus)

PCI Bus Signals	Signal Type/ Direction	Description
AD_IO [31:0]	inout	*Address/Data* multiplexed on the same PCI pins.
CBE_IO [3:0]	inout	*Bus Command* and *Byte Enables* multiplexed on the same PCI pins.
PAR_IO	inout	Even *Parity* signal across AD_IO[31:00] and CBE_IO[3:0].
FRAME_IO	inout	*Cycle Frame* is driven by the current master to indicate the beginning and duration of an access.
TRDY_IO	inout	*Target Ready* indicates the target agent's ability to complete the current data phase of the transaction.
IRDY_IO	inout	*Initiator Ready* indicates the initiating agent's ability to complete the current data phase of the transaction.
STOP_ IO	inout	*Stop* indicates the current target is requesting the master to stop the current transaction.
DEVSEL_IO	inout	*Device Select*, when actively driven, indicates the driving device has decoded its address as the target of the current access.
IDSEL_I	in	*Initialization Device Select* is used as a chip select during configuration read and write transactions.
NTA_O	out	*Interrupt A* is used to request an interrupt.
PERR_IO	inout	*Parity Error* is only for the reporting of data parity errors during all PCI transactions

PCI Bus Signals	Signal Type/ Direction	Description
		except a Special Cycle.
SERR_IO	inout	*System Error* is for reporting address parity errors, data parity errors on the Special Cycle command, or any other system error where the result will be catastrophic.
REQ_O	out	*Request* indicates to the arbiter that this agent desires use of the bus.
GNT_I	in	*Grant* indicates to the agent that access to the bus has been granted.
RST_I	in	*Reset* is used to bring PCI-specific registers, sequencers, and signals to a consistent state.
PCLK	in	*PCI Clock* provides timing for all transactions on PCI and is an input to every PCI device.

Table 31: pci32 Reg Block Interface Signals

pci32 Reg Block Interface Signals	Signal Type/ Direction	Description
pci_int_out [2:0]	out	Unused. (signal is used for a DDC board-level product)
init_drr_timer [15:0]	in	Unused. (signal is used for a DDC board-level product)
pci_ctrl [7:0]	in	Unused. (signal is used for a DDC board-level product)
DMA_REQ	in	Ties to ACE Flex-Core Top *dma_start* signal output
DMA_ACK	buffer	Unused. (signal is used for a DDC board-level product)
DMA_CMPLT	buffer	Ties to ACE Flex-Core Top *dma_cmplt* signal input
DMA_REQ_VECTOR [64:0]	in	Ties to ACE Flex-Core Top Configuration Register outputs in the following sequence: *DMA_REQ_VECTOR*[64:50] => *cfg_reg14*[14:0] *DMA_REQ_VECTOR*[49:47] => *cfg_reg13*[2:0] *DMA_REQ_VECTOR*[46:31] => *cfg_reg12*[15:0] *DMA_REQ_VECTOR*[30:16] => *cfg_reg11*[14:0] *DMA_REQ_VECTOR*[15:0] => *cfg_reg10*[15:0]
dma_task_status [3:0]	out	Ties to ACE Flex-Core Top *dma_task_status* signal inputs

Table 32: General Resets and Clock

Signal	Signal Type/ Direction	Description
rst_pin_l	in	Manual reset input
rst_l	out	Manual reset output for synchronization of external devices
mclk	in	40Mhz system clock (same source as ACE Flex-Core clk signal)

Table 33: pci32 DDC Bus Interface Signals (ACE Flex-Core Host Bus)

pci32 DDC Bus Interface Signals	Signal Type/ Direction	Description
ddc_mem_reg_l	out	Ties to ACE Flex-Core Top *mem_reg_l* signal output
ddc_target_rqst	buffer	Ties to ACE Flex-Core Top *target_rqst* signal input
ddc_target_grnt	in	Ties to ACE Flex-Core Top *target_grnt* signal output
ddc_wr	out	Ties to ACE Flex-Core Top *write_en* signal input
ddc_rd	out	Ties to ACE Flex-Core Top *read_en* signal input
ddc_bwe_l [3:0]	out	Ties to ACE Flex-Core Top *bwe_host_l* signal inputs
ddc_addr [18:0]	out	Ties to ACE Flex-Core Top *addr_host* signal inputs
ddc_data_out [31:0]	in	Ties to ACE Flex-Core Top *data_out* signal outputs
ddc_data_in [31:0]	out	Ties to ACE Flex-Core Top *data_int* signal inputs
ddc_int_l	in	Ties to ACE Flex-Core Top *dma_grnt* signal output

Table 34: pci32 DMA Interface Signals (ACE Flex-Core DMA Bus)

pci32 PCI DMA Interface Signals	Signal Type/ Direction	Description
dma_rqst	out	Ties to ACE Flex-Core Top *dma_rqst* signal input
dma_grnt	in	Ties to ACE Flex-Core Top *dma_grnt* signal output
dma_wr	out	Ties to ACE Flex-Core Top *wr_dma* signal input
dma_rd	out	Ties to ACE Flex-Core Top *rd_dma* signal input
dma_addr [18:0]	out	Ties to ACE Flex-Core Top *addr_dma* signal inputs
dma_wdata [31:0]	out	Ties to ACE Flex-Core Top *data_in_dma* signal inputs
dma_bwe_l [3:0]	out	Ties to ACE Flex-Core Top *bwe_dma_l* signal inputs
pci_clk	out	Unused. (signal is used for a DDC board-level product)

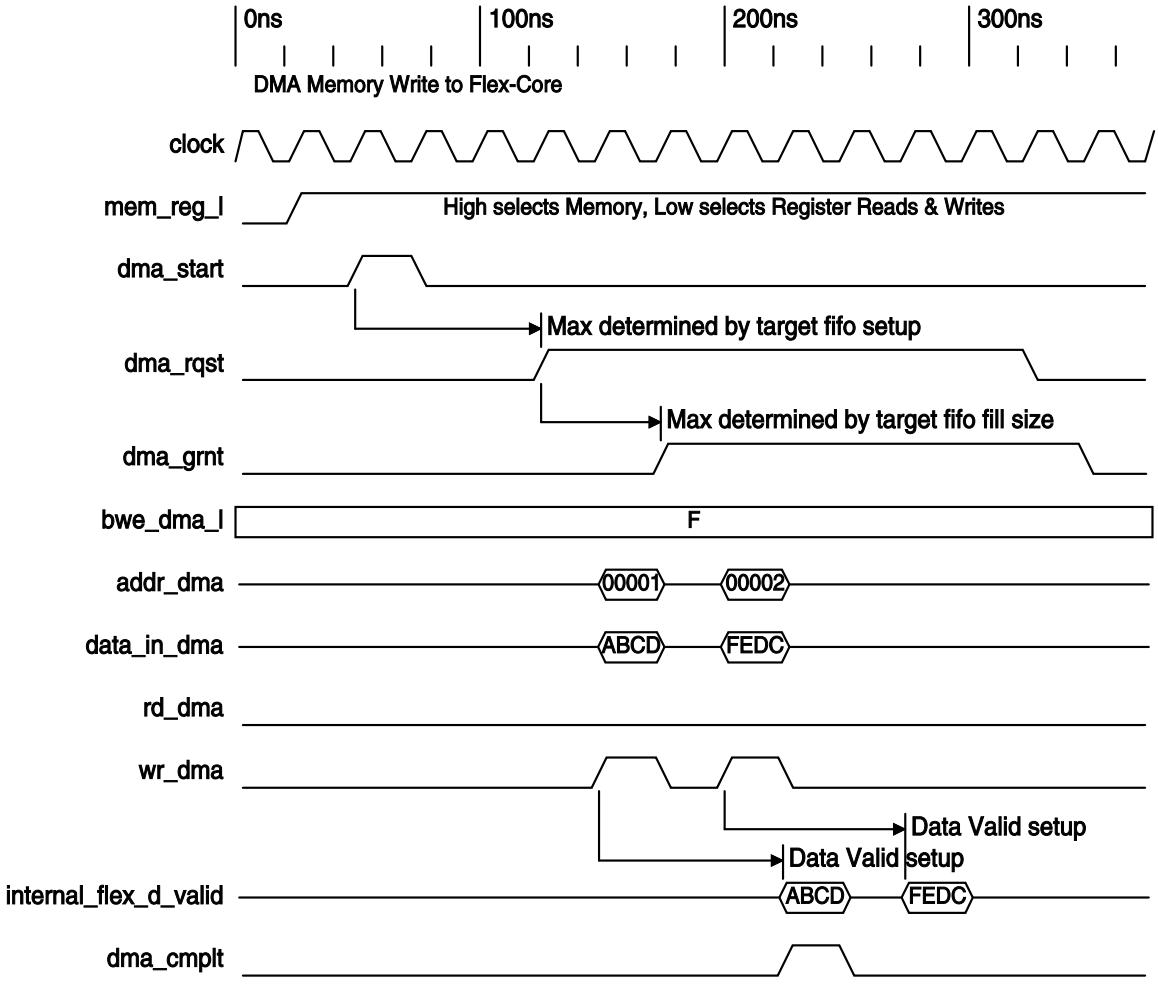


Figure 27: DMA Mem Write to ACE Flex-Core RAM

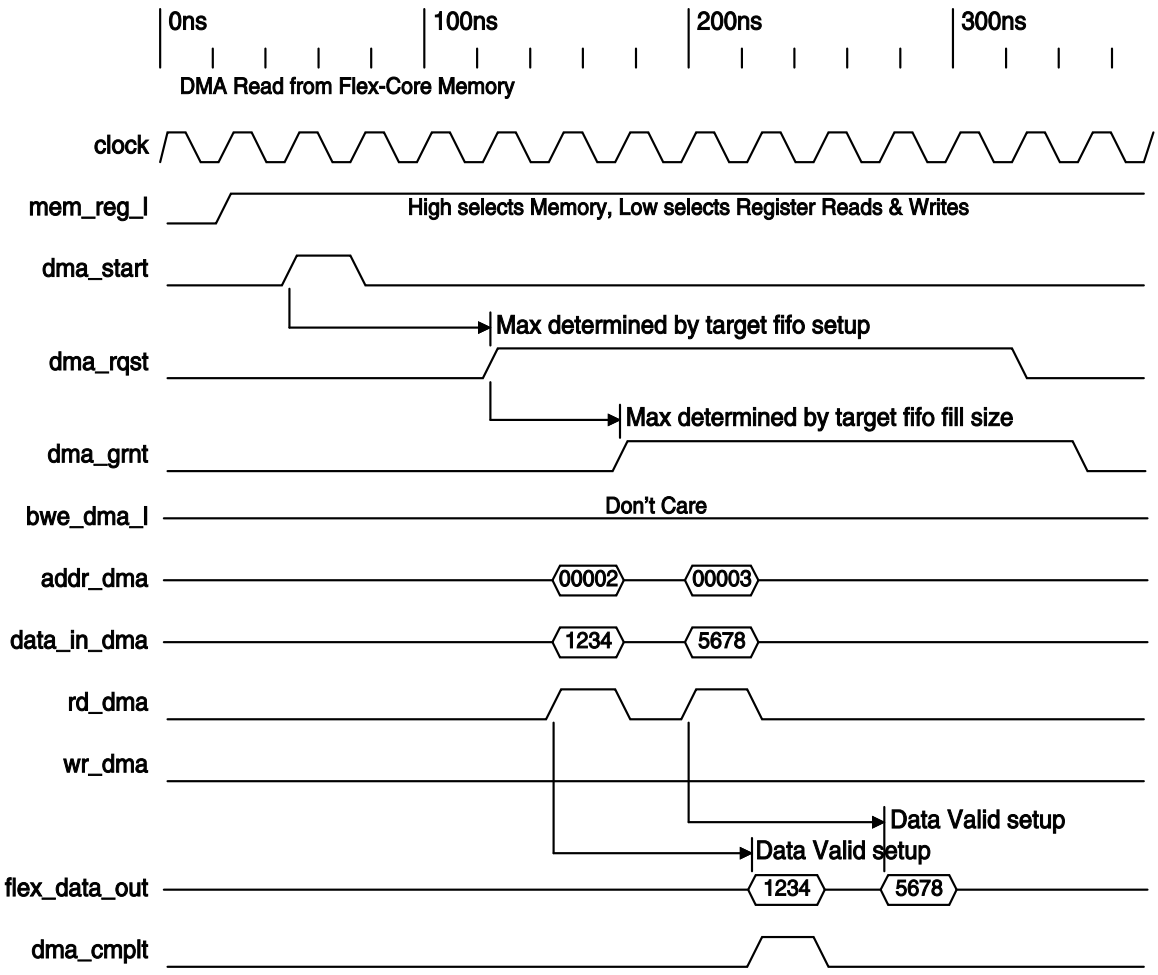


Figure 28: DMA Read from ACE Flex-Core RAM

7 IP MODULE SUPPLEMENTAL DESIGN DATA

7.1 General Description of IP Module

Each instance of the Flex-Core IP requires the use of a BU-69299 “IP Module”. The Flex-Core will not operate for longer than 65 msec without an IP Module. Following a hardware reset, the Flex-Core will establish communication with the IP Module over the single wire “ip_module_serial_data” path. The Flex-Core IP will not function unless it is able to establish communication and verify the existence of a valid IP Module.

7.2 Functional Description

Table 35: Pin Description

Pin Number	Signal Name	Description
1	NC	No Connect. This pin must not be connected to any signal power or ground.
2	V_{SS}	Logic Ground
3	DATA	Input/output signal that must be connected to the “ip_module_serial_data” port on the Flex-Core IP.
4	NC	No Connect. This pin must not be connected to any signal power or ground.
5	V_{DD}	Logic Supply Voltage
6	RESET_L	IP Module reset input. This signal must be connected to either the “rst_l” input or the “ip_module_reset_l” output from the Flex-Core.

7.2.1 Reset and Synchronization

The Flex-Core will establish communication with the IP Module following a hardware reset (i.e. following the assertion of logic 0 to the reset_l input signal). The IP Module must be reset and released out of reset at the same time as the Flex-Core. The rst_l input to the IP Module must be connected to either the ip_module_reset_l output signal from Flex-Core (as illustrated in Figure 29) or the rst_l output signal from Flex-Core (as illustrated in Figure 30).

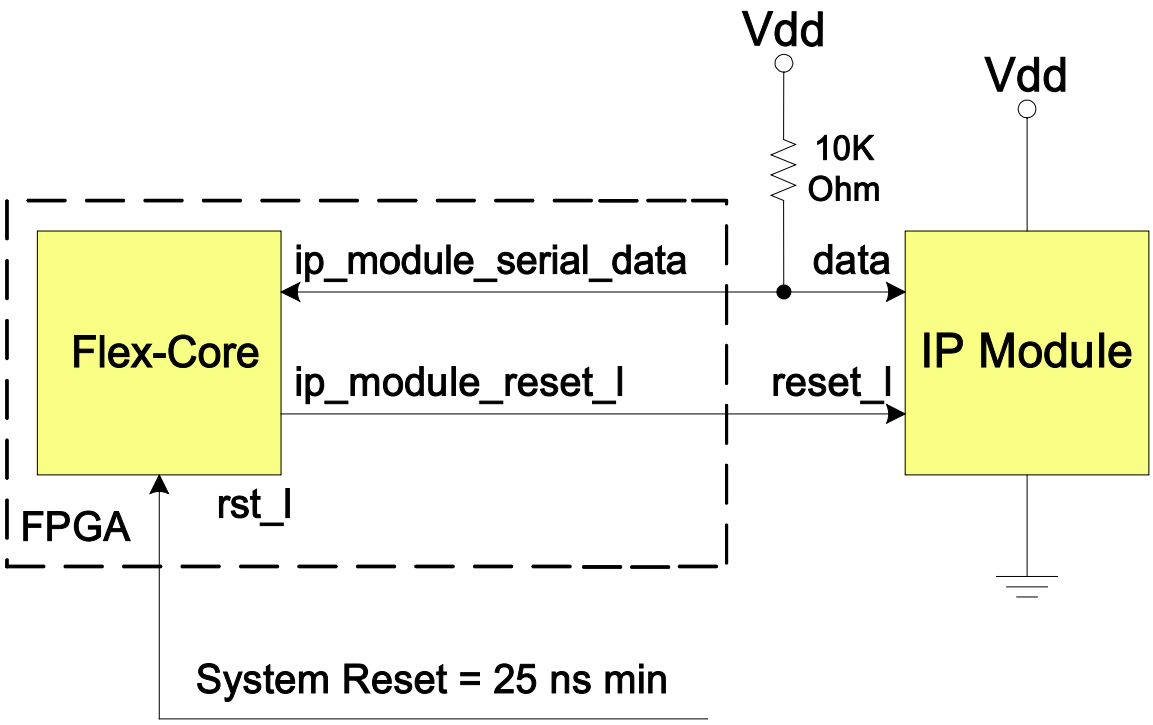


Figure 29: IP Module Reset Using Flex-Core ip_module_reset_l Output

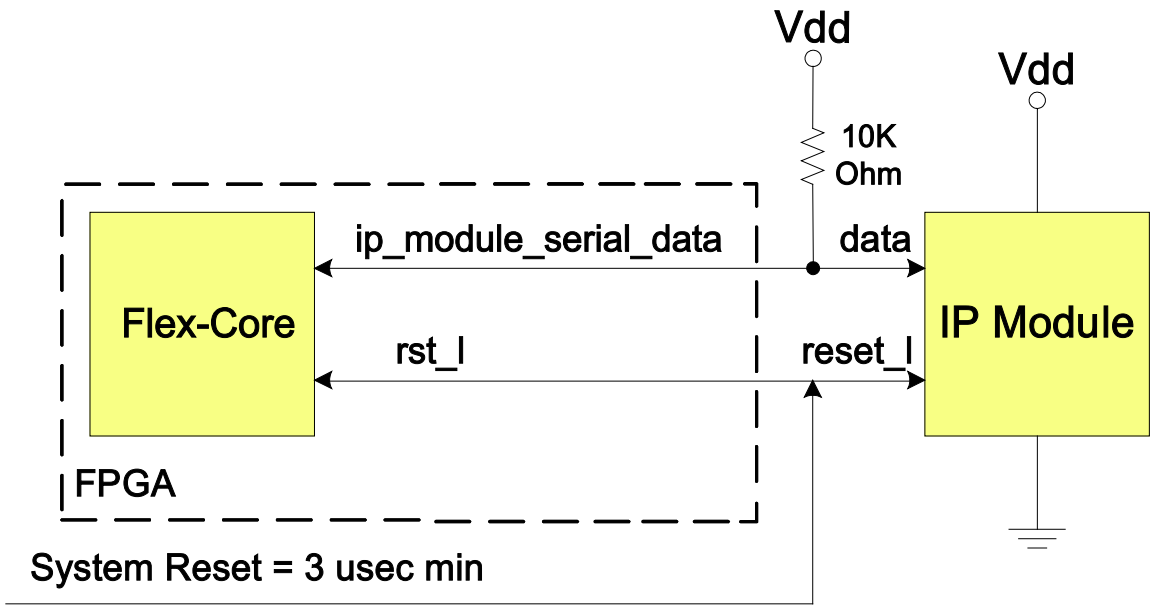


Figure 30: IP Module Reset Using Flex-Core rst_l Input

Flex-Core requires that the rst_l input signal be held low for a minimum of 1 clock cycle (25 ns) to ensure a valid reset. The IP Module requires that the reset_l input signal be held low for a minimum of 3 usec to

ensure a valid reset. If an external hardware reset signal is used to reset both the Flex-Core and the IP Module (i.e. an external signal drives both rst_l on Flex-Core and reset_l on the IP Module as illustrated in Figure 30) then the timing of the reset must be such that it satisfies the 3 usec minimum requirement of the IP Module.

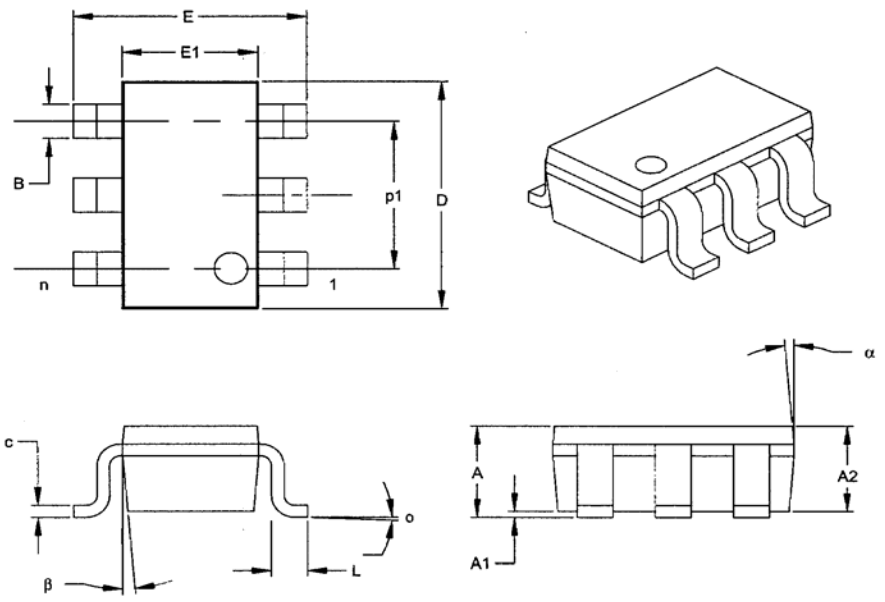
If the ip_module_reset_l output from Flex-Core is used to reset the IP Module (as illustrated inFigure 29) then the Flex-Core will satisfy the 3 usec minimum reset time for the IP Module and the minimum required system reset time is 25 ns (i.e. Flex-Core's reset time).

The Flex-Core's verification process (i.e. verifying the existence of a valid IP Module) takes 65 msec. The Flex-Core will operate correctly for 65 ms following a hardware reset (i.e. following the assertion of a logic 0 followed by logic 1 on the rst_l input). The Flex-Core will become disabled after 65 msec if it does not detect the existence of a valid IP Module. While disabled, the Flex-Core will respond with a fixed data pattern (0x55AAAA55) for all memory or register read accesses. To verify that the Flex-Core has successfully detected the presence of an IP Module, the host may perform a read/write test of either memory or registers after a delay of 65 msec following a hardware reset.

7.3 Mechanical Outline

Figure 31 provides an illustration of the mechanical outline of the BU-69299R. This package is lead free.

6-Lead Plastic Small Outline Transistor (CH or OT) (SOT-23)



Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		6			6	
Pitch	p		.038			0.95	
Outside lead pitch (basic)	p1		.075			1.90	
Overall Height	A	.035	.046	.057	0.90	1.18	1.45
Molded Package Thickness	A2	.035	.043	.051	0.90	1.10	1.30
Standoff	A1	.000	.003	.006	0.00	0.08	0.15
Overall Width	E	.102	.110	.118	2.60	2.80	3.00
Molded Package Width	E1	.059	.064	.069	1.50	1.63	1.75
Overall Length	D	.110	.116	.122	2.80	2.95	3.10
Foot Length	L	.014	.018	.022	0.35	0.45	0.55
Foot Angle	ϕ	0	5	10	0	5	10
Lead Thickness	c	.004	.006	.008	0.09	0.15	0.20
Lead Width	B	.014	.017	.020	0.35	0.43	0.50
Mold Draft Angle Top	α	0	5	10	0	5	10
Mold Draft Angle Bottom	β	0	5	10	0	5	10

*Controlling Parameter

Notes:

Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .005" (0.127mm) per side.

Figure 31: BU-69299R Mechanical Outline Drawing

7.4 Specifications

Table 36: Specifications

Absolute Maximum Ratings				
Parameters	**Min**	**Typ**	**Max**	**Units**
Ambient temperature under bias	-40		+125	°C
Storage temperature	-65		+150	°C
Voltage on V_{DD} with respect to V_{SS}	0		6.5	V
Voltage on RESET_L with respect to V_{SS}	0		13.5	V
Voltage on all other pins with respect to V_{SS}	-0.3		V_{DD}+0.3	V
Total power dissipation(1)			800	mW
Max. current out of V_{SS} pin			80	mA
Max. current into V_{DD} pin			80	mA
Input clamp current, I_{IK} ($V_I < 0$ or $V_I > V_{DD}$)	-20		+20	mA
Output clamp current, I_{OK} ($V_O < 0$ or $V_O > V_{DD}$)	-20		+20	mA
Max. output current sunk by DATA I/O pin			75	mA
Max. output current sourced by DATA I/O pin			75	mA
Power Supply Requirements				
Parameters	**Min**	**Typ**	**Max**	**Units**
V_{DD} Supply Voltage	2.0		5.5	V
I_{DD} Supply Current (V_{DD} = 2.0V)		170		µA
I_{DD} Supply Current (V_{DD} = 5.0V)		250		µA
Logic				
Parameters	**Min**	**Typ**	**Max**	**Units**
V_{IL} ($2.0 \le V_{DD} < 4.5V$)	V_{SS}		$0.15V_{DD}$	V
V_{IL} ($4.5 \le V_{DD} \le 5.5V$)	V_{SS}		0.8	
V_{IH} ($2.0 \le V_{DD} < 4.5V$)	$0.25V_{DD}$ +0.8		V_{DD}	V
V_{IH} ($4.5 \le V_{DD} \le 5.5V$)	2.0		V_{DD}	V
I_{IL} ($V_{SS} \le V_{PIN} \le V_{DD}$, Pin at high-impedance)	-1		+1	µA
I_{IH} ($4.5 \le V_{DD} \le 5.5V$)				
V_{OL} (I_{OL} = 8.5 mA, V_{DD} = 4.5V)			0.6	V
V_{OH} (I_{OH} = -3.0 mA, V_{DD} = 4.5V)	V_{DD}-0.7			V
Capacitive Loading on Output Pins			50	pF

Note: Power dissipation is calculated as follows: $P_{DIS} = V_{DD} \times \{I_{DD} - I_{OH}\} + \{(V_{DD} - V_{OH}) \times I_{OH}\} + (V_{OL} \times I_{OL})$