LogiCORE IP Video Direct Memory Access v1.1

DS730 September 21, 2010 **Product Specification**

Introduction

The Xilinx Video Direct Memory Access (Video DMA) LogiCORE™ IP allows video cores to access external memory via the Video Frame Buffer Controller (VFBC) port on the Multi-Port Memory Controller (MPMC). The Video DMA is highly programmable through registers coupled with a wide range of interrupts, allowing for easy control of the various features of the core. The integration with the MicroBlaze™ Soft Processor for in-system control of the block in real-time allows designers an easy path to integrate DMA functionality for video data accesses.

Features

- Programmable register control
- Selectable processor interface
  - EDK pCore
  - General Purpose Processor
- Selectable Master/Slave Gen-Lock Mode
- Selectable data interface
  - VDMA FIFO interface
  - Xilinx Streaming Video Interface (XSVI)
- Configurable Read, Write, or Read/Write DMA mode
- Programmable data width -8, -16, -32 or -64
- Seamless integration with Video Frame Buffer Controller
- PLB46 support for interrupts and status register access
- Support for up to 16 buffer addresses
- Support for non-aligned transfers

LogiCORE IP Facts Table					
Core Specifics					
Supported Device Family(1)	Spartan®-3A DSP, Spartan-6, Virtex®-5, Virtex-6				
Supported User Interfaces	General Processor Interface, EDK PLB 4.6				
	Resources(2)				**Frequency**
Configuration	**LUTs**	**FFs**	**DSP Slices**	**Block RAMs(3)**	**Max. Freq.(4)**
Write_Only, pCore IF, 5 Frame Stores	1048	1617	0	0	225
Read_Only, pCore IF, 3 Frame Stores, Non-Aligned Transfers	1177	1588	0	0	225
Read/Write, pCore IF, 3 Frame Stores	1240	1678	0	0	225
Provided with Core					
Documentation	Product Specification				
Design Files	Netlist, EDK pCore				
Example Design	Not Provided				
Test Bench	Not Provided				
Constraints File	Not Provided				
Simulation Model	Not Provided				
Tested Design Tools					
Design Entry Tools	ISE® 12.3 XPS 12.3				
Simulation	ModelSim v6.5c ISE Simulator 12.3				
Synthesis Tools	ISE XST 12.3				
Support					
Provided by Xilinx, Inc.					

1. For a complete listing of supported devices, see the release notes for this core.
2. Resources listed here are for Virtex-6® devices. For more complete device performance numbers, see "Core Resource Utilization," page 42.
3. Based on 36K block RAMs.
4. Performance numbers listed are for Virtex-6 FPGAs. For more complete performance data, see "Performance," page 44.

Discontinued IP

© Copyright 2009 - 2010. Xilinx, Inc. XILINX, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owner.

Applications

- Video Surveillance
- Industrial Imaging
- Video Conferencing
- Machine Vision

Overview

The majority of video systems being designed utilize external buffers for temporary storage of video frames. The requirements of these systems provide a challenge to developers to easily control this external buffer and the transfer of the data for processing. Additionally complicating the control is processing cores implemented within the design clocked at different sampling rates, and external memory interfaces imposing restrictions on the format or location of memory transfers. As a result, synchronizing video as it passes through a multi-rate system can be very challenging and error prone, making the job of designing a video system very difficult.

The Video DMA was designed to help address these issues. It was designed to directly interface to the Video Frame Buffer Controller (VFBC) integrated into the Multi-Port Memory Controller (MPMC). It automatically generates the CMD signals for the VFBC and simplifies the process of setting up and controlling frame buffers in external memory. Additionally, it can compensate for data transfers that do not meet the VFBC format requirements. The Video DMA also has a system synchronization mechanism called Gen-Lock that eases the burden of synchronizing data as it moves from one processing domain to another based on block-to-block shared signaling.

The Video DMA is very flexible and can be used in a number of modes and configurations. A comprehensive set of registers and interrupts makes the Video DMA highly programmable and easy to control in real-time with a processor such as MicroBlaze.

Discontinued IP

CORE Generator Graphical User Interface (GUI)

The Xilinx Video Direct Memory Access LogiCORE IP is easily configured to meet the developer's specific needs through the CORE Generator™ graphical user interface (GUI). This section provides a quick reference to parameters that can be configured at generation time. Figure 1 shows the first page of the GUI.

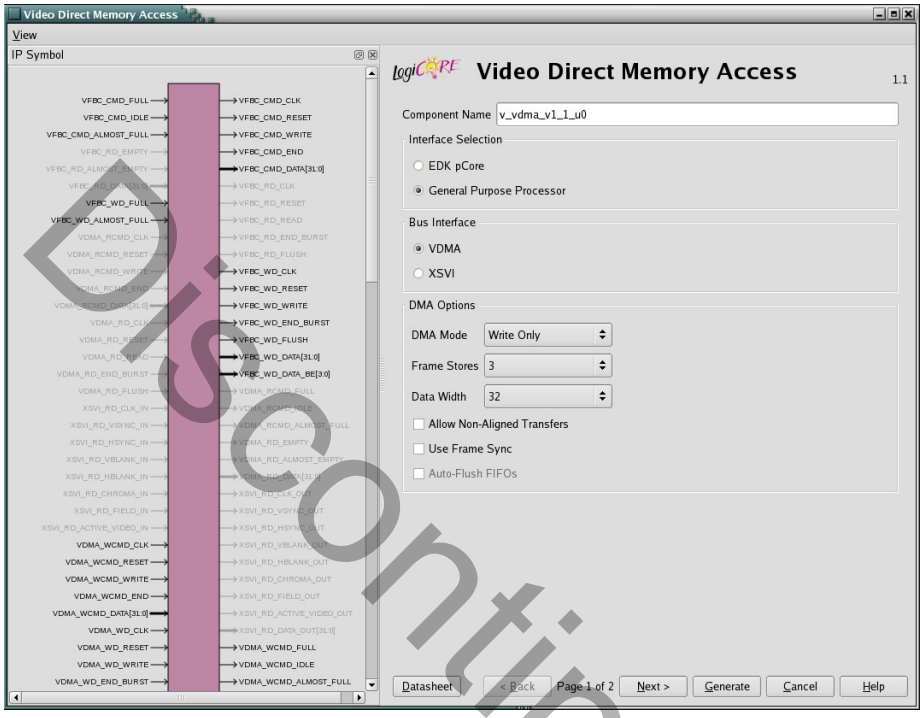

Figure 1: **Video DMA Main Screen**

The main screen displays a representation of the IP symbol on the left side, and the parameter assignments on the right side, which are described as follows:

- **Component Name:** The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9 and "_".

  Note: The name "v_vdma_v1_1" is not allowed.

- **Interface Selection:** The Video DMA is generated with one of two processor interfaces
  - **EDK pCore Interface:** CORE Generator will generate the Video DMA as a pCore which can be easily imported into an EDK project as a hardware peripheral. The core registers can then be programmed in real-time via the processor. See the "EDK pCore Interface" section. When the EDK pCore is selected, the rest of the options are disabled and set to the default value. All modifications to the Video DMA pCore are made with the EDK GUI.
  - **General Purpose Processor Interface:** CORE Generator will generate a set of ports that can be used to program the Video DMA. See the "General Processor Interface" section. When the General Purpose Processor interface is selected, the rest of the configuration options become active and can be used to generate a customized Video DMA core.

Discontinued IP

- **Bus Interface:** The Video DMA is generated with one of two data interfaces
  - **VDMA:** CORE Generator will generate the Video DMA with a VDMA FIFO data interface.
  - **XSVI**: CORE Generator will generate the Video DMA with an XSVI streaming data interface.
- **DMA Options**
  - **DMA Mode:** The Video DMA can be configured for three different modes of operation. The allowable selections are:
    - **Write_Only Mode:** The Video DMA will perform only write operations.
    - **Read_Only Mode:** The Video DMA will perform only read operations.
    - **Read/Write Mode:** The Video DMA will perform both read and write operations.
  - **Frame Stores:** The Frame Stores parameter allows the Video DMA to be configured with the specified number of Read or Write Address Registers. The permitted values are 1 – 16. Typically this is the number of frame buffers to be created in external memory.
  - **Data Width:** The Data Width parameter specifies the width of the data buses of the Video DMA data read and write ports. The permitted values are 8, 16, 32 and 64.
  - **Allow Non-Aligned Transfers:** When selected, this parameter specifies that additional logic will be included in the Video DMA to perform horizontal cropping/padding of VFBC read or writes. Horizontal cropping/padding is necessary if any transfers will not be properly aligned with the 128-byte boundaries required by the VFBC. This includes memory addressing that does not align to the 128-byte boundaries or to horizontal data lengths that are not 128-byte multiples.
  - **Use Frame Sync:** When selected, this parameter specifies that the Video DMA will synchronize all frame operations with the falling edge of the `fsync` signal. The `fsync` signal is commonly driven by the Video Timing Controller LogiCORE IP or the `vsync` signal of a streaming video bus.
  - **Auto-Flush FIFOs:** When selected, this parameter specifies that all VFBC FIFOs should be Flushed and Reset before each transfer. This option is available only if "Use Frame Sync" is selected.

Discontinued IP

Page 2 of the Video DMA GUI (Figure 2) allows the specification of the core optional Gen-Lock capabilities.

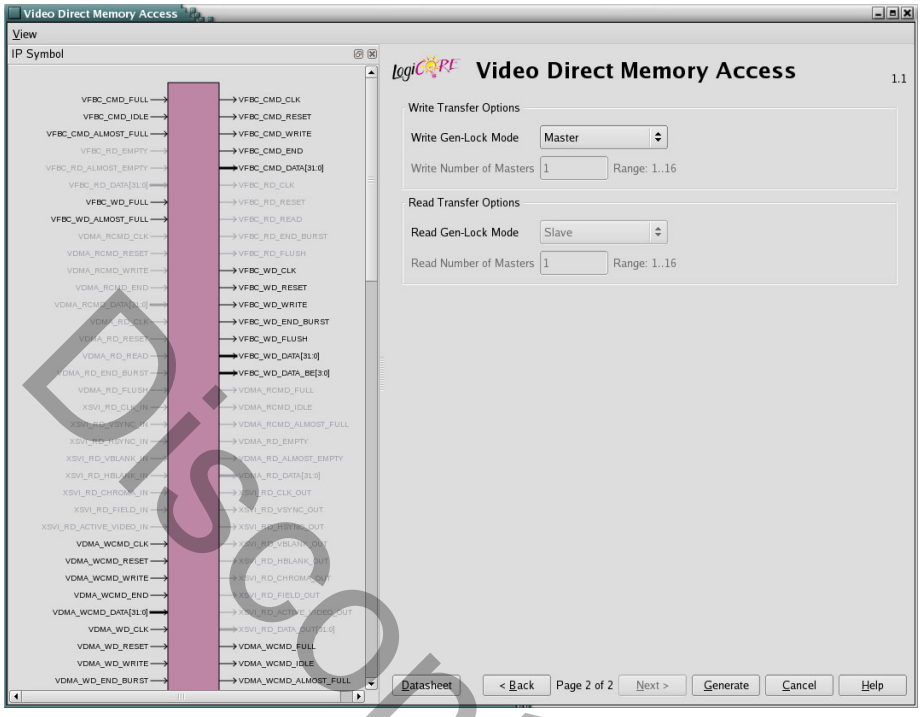

Figure 2: **Video DMA, Gen-Lock Options Screen**

- **Write Transfer Options:** These parameters configure the Video DMA Gen-Lock when in operated in write mode.
  - **Write Gen-Lock Mode:** Specifies the operating mode of the Write Gen-Lock. The allowed selections are:
    - **Master:** Master mode specifies that the Video DMA will operate as the Gen-Lock Master. Gen-Lock Masters do not drop or repeat frames. See the "Gen-Lock Operation" section for more details.
    - **Slave:** Slave mode specifies that the Video DMA will operate as a Gen-Lock Slave. Gen-Lock Slaves automatically drop and repeat frames based on the master and slave frame rates. See the "Gen-Lock Operation" section for more details.
  - **Write Number of Masters:** Specifies the number of Masters to which the Slave can synchronize. The Video DMA uses a register to dynamically specify which master is in control at any given time. The Write Number of Masters parameter is available only if the Write Gen-Lock Mode is set to Slave.
- **Read Transfer Options:** These parameters configure the Video DMA Gen-Lock when in operated in read mode.
  - **Read Gen-Lock Mode:** Specifies the operating mode of the Read Gen-Lock. The allowed selections are:
    - **Master:** Master mode specifies that the Video DMA will operate as the Gen-Lock Master. Gen-Lock Masters do not drop or repeat frames. See the "Gen-Lock Operation" section for more details.
    - **Slave:** Slave mode specifies that the Video DMA will operate as a Gen-Lock Slave. Gen-Lock Slaves automatically drop and repeat frames based on the master and slave frame rates. See the "Gen-Lock Operation" section for more details.
  - **Read Number of Masters:** Specifies the number of Masters to which the Slave can synchronize. The Video DMA uses a register to dynamically specify which master is in control at any given time. The Read Number of Masters parameter is available only if the Read Gen-Lock Mode is set to Slave.

Discontinued IP

EDK pCore Graphical User Interface (GUI)

When the Xilinx Video Direct Memory Access LogiCORE IP is generated from CORE Generator as an EDK pCore, it is generated with each option set to the default value. All customizations of a Video DMA pcore are done with the EDK pCore graphical user interface (GUI). Figure 3 and Figure 4 illustrate the EDK pCore GUI for the Video DMA. All of the options in EDK pCore GUI for the Video DMA correspond to the same options in the CORE Generator GUI for the Video DMA. See the "CORE Generator Graphical User Interface (GUI)" section for option details.

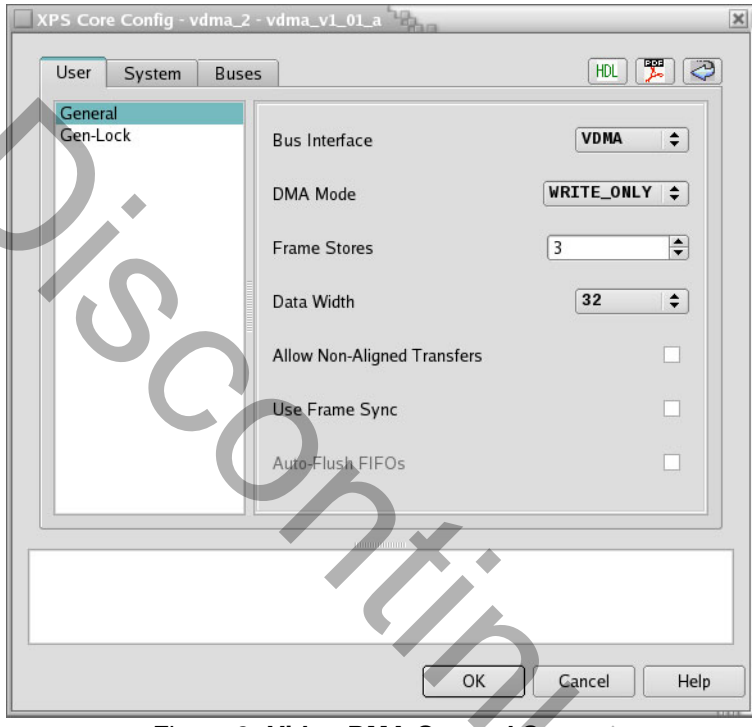

Figure 3: **Video DMA General Screen**

Discontinued IP

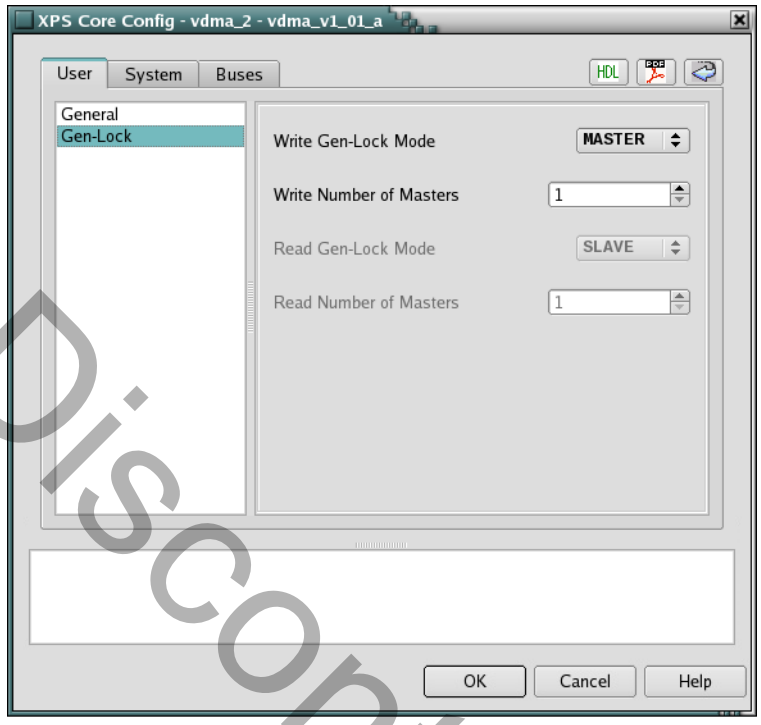

Figure 4: **Video DMA Gen-Lock Screen**

Discontinued IP

Video DMA Core Interfaces

There are many video systems developed that use an integrated MicroBlaze processor soft core to dynamically control the parameters within the system. This is especially important when several independent image processing cores are integrated into a single FPGA. The Video DMA core can be configured with one of two interfaces: an EDK pCore Interface or a General Purpose Processor Interface.

EDK pCore Interface

The pCore interface creates a core that can be easily added to an EDK Project as a hardware peripheral. This section describes the Register Set, the pCore Driver Files, and the I/O signals associated with the Video DMA pCore.

Once generated by CORE Generator software, the new VDMA pCore is located in the CORE Generator project directory at <Component_Name>/pcores/vdma_v1_01_a. The pCore should be copied to the user's <EDK_Project>/pcores directory or to a user pCores repository. The VDMA pCore driver software is located in the CORE Generator project directory at <Component_Name>/drivers/vdma_v1_01_a. The driver software should be copied to the user's <EDK_Project>/drivers directory or to a user pCores repository.

pCore Register Set

The pCore interface provides a memory mapped interface for the programmable registers within the core, which are defined in Table 1, all registers default to 0x00000000 on Power-on/Reset.

Table 1: **Video DMA pCore Memory Mapped Register Set**

Address (hex)	Register Name	Access Type	Description	
BASEADDR + 0x0000	VDMA Control	R/W	General Control Register	
			31	Reserved
			30	SW Write DMA Reset Clear Write DMA Command and Flush Write Data.
			29	SW Read DMA Reset Clear Read DMA Command and Flush Read Data.
			28	SW Write FIFO Flush
			27	SW Read FIFO Flush
			26	Reserved
			25	Read HW Lockout 1=Disable VDMA Read Command Hardware from writing to Command Interface. All commands from the VDMA Read Command Interface will be ignored.
			24	Write HW Lockout 1=Disable VDMA Write Command Hardware from writing to Command Interface. All commands from the VDMA Write Command Interface will be ignored.
			20:23	Read Frame Store Pointer When Circular Buffer Enable = 0, the Frame Store Start Address reference number stored here will force the VDMA to place transactions to/from this Start Address. Ignored with Circular Buffer Enable = 1.
			16:19	Write Frame Store Pointer When Circular Buffer Enable = 0, the Frame Store Start Address reference number stored here will force the VDMA to place transactions to/from this Start Address. Ignored with Circular Buffer Enable = 1.

Discontinued IP

Table 1: **Video DMA pCore Memory Mapped Register Set *(Cont'd)***

Address (hex)	Register Name	Access Type	Description	
			12:15	Read Pointer Number The source pointer to use for Read Gen-Lock frame pointer comparisons. Used only when the "Read Number of Masters" parameter is > 1.
			8:12	Write Pointer Number The source pointer to use for Write Gen-Lock frame pointer comparisons. Used only when the "Write Number of Masters" parameter is > 1.
			7	Frame Count Enable 1 = Based on value in Frame Count register, only enable transfers for this number of frames. The VDMA will halt when the frame count is reached.
			5:6	Reserved
			4	Horizontal Cropping Enable 1 = Allow Non-Aligned Memory transfers as well as horizontal lengths that are not 128-byte multiples.
			3	Sync_Enable When Circular Buffer Enable = 1, and when 1, compare current frame store address pointer to incoming frame store address pointer. When 0, the slave VDMA will not be synchronized to the master VDMA.
			2	Circular Buffer Enable 0 = Use Start Address specified in Read/Write Frame Store Pointer. 1 = Rotate Start Addresses.
			1	VDMA Read Enable Enable/Start Internal VDMA Read Transaction(s). This enables generating internal VDMA read command words. Read Command words can still be written into the VDMA if this bit is zero.
			0	VDMA Write Enable Enable/Start Internal VDMA Write Transaction(s). This enables generating internal VDMA write command words. Write Command words can still be written into the VDMA if this bit is zero.

Discontinued IP

Table 1: **Video DMA pCore Memory Mapped Register Set *(Cont'd)***

Address (hex)	Register Name	Access Type	Description	
BASEADDR + 0x000c	Control Counters	R/W	Frame and Delay Counters	
			24:31	Read Frame Count Send interrupt after this number of frames has been read.
			16:23	Read Delay Timer Count Send interrupt after this number of delay counter ticks for the read DMA. The delay counter period is controlled by a clock divider.
			8:15	Write Frame Count Send interrupt after this number of frames has been written.
			0:7	Write Delay Timer Count Send interrupt after this number of delay counter ticks for the write DMA. The delay counter period is controlled by a clock divider.
BASEADDR + 0x0010	Status Counters	R	Frame and Delay Counters Status	
			24:31	Read Frame Counter Value Indicates the number of frames left to be read of the number specified Read Frame Count.
			16:23	Read Delay Count Value Indicates the number of counter ticks left of the number specified in Read Delay Timer Count.
			8:15	Write Frame Counter Value Indicates the number of frames left to be written of the number specified in Write Frame Count.
			0:7	Write Delay Count Value Indicates the number of counter ticks left of the number specified in Write Delay Timer Count.

Discontinued IP

Table 1: **Video DMA pCore Memory Mapped Register Set *(Cont'd)***

Address (hex)	Register Name	Access Type	Description	
BASEADDR + 0x0014	Status Frame Store	R	General Status Register for Frame Store Information	
			31	VDMA Busy DMA transaction in progress.
			30	Cmd FIFO Full
			29	Cmd FIFO Almost Full
			28	Write FIFO Full
			27	Write FIFO Almost Full
			26	Read FIFO Empty
			25	Read FIFO Almost Empty
			24	Cmd FIFO Write Enable
			23	Write FIFO Write Enable
			22	Read FIFO Read Enable
			20:21	Reserved
			16:19	Max Valid Frame Store Number of last Frame Store in VDMA. The VDMA can be configured for 1 – 16 frame stores. This register will report 0 – 15.
			8:15	Reserved
			4:7	Current Read Frame Store Pointer to Current Frame Store Start Address Number. Reports the frame store the current transaction is operating upon.
			0:3	Current Write Frame Store Pointer to Current Frame Store Start Address Number. Reports the frame store the current transaction is operating upon.

Discontinued IP

Table 1: **Video DMA pCore Memory Mapped Register Set *(Cont'd)***

Address (hex)	Register Name	Access Type	Description	
BASEADDR + 0x0020	Control Write Frame Size	R/W	Horizontal Size and Vertical Size of Each Write FIFO Frame	
			28:31	Reserved
			16:27	VDMA Write Vsize The write vertical size in lines starting from line 0.
			12:15	Reserved
			0:11	VDMA Write HSize The horizontal size in number of data writes.
BASEADDR + 0x0024	Control Write Stride	R/W	Stride (Line Increment) of each Write FIFO Frame	
			20:31	Reserved
			16:19	Write Frame Delay The number of frame stores the write VDMA should be behind the locked Read DMA. Used only if the "Write Gen-Lock Mode" parameter is set to "SLAVE" and the circular buffer is enabled.
			12:15	Reserved
			0:11	VDMA Write Stride The stride in number of data elements of DATA WIDTH size.
BASEADDR + 0x0028	Control Read Frame Size	R/W	Horizontal Size and Vertical Size of each Read FIFO Frame	
			28:31	Reserved
			16:27	VDMA Read Vsize The read vertical size in lines starting from line 0.
			12:15	Reserved
			0:11	VDMA Read HSize The horizontal size in number of data reads.
BASEADDR + 0x002c	Control Read Stride	R/W	Stride (Line Increment) of each Read FIFO Frame	
			20:31	Reserved
			16:19	Read Frame Delay The number of frame stores the read VDMA should be behind the locked Write DMA. Used only if the "Read Gen-Lock Mode" parameter is set to "SLAVE" and the circular buffer is enabled.
			12:15	Reserved
			0:11	VDMA Read Stride. The stride in number of data elements of DATA WIDTH size.

Discontinued IP

Table 1: **Video DMA pCore Memory Mapped Register Set *(Cont'd)***

Address (hex)	Register Name	Access Type	Description
BASEADDR + 0x0030	Control Write Start Address 0	R/W	Start Address of Write Frame Store 0
BASEADDR + 0x0034	Control Write Start Address 1	R/W	Start Address of Write Frame Store 1
BASEADDR + 0x0038	Control Write Start Address 2	R/W	Start Address of Write Frame Store 2
BASEADDR + 0x003c	Control Write Start Address 3	R/W	Start Address of Write Frame Store 3
BASEADDR + 0x0040	Control Write Start Address 4	R/W	Start Address of Write Frame Store 4
BASEADDR + 0x0044	Control Write Start Address 5	R/W	Start Address of Write Frame Store 5
BASEADDR + 0x0048	Control Write Start Address 6	R/W	Start Address of Write Frame Store 6
BASEADDR + 0x004c	Control Write Start Address 7	R/W	Start Address of Write Frame Store 7
BASEADDR + 0x0050	Control Write Start Address 8	R/W	Start Address of Write Frame Store 8
BASEADDR + 0x0054	Control Write Start Address 9	R/W	Start Address of Write Frame Store 9
BASEADDR + 0x0058	Control Write Start Address 10	R/W	Start Address of Write Frame Store 10
BASEADDR + 0x005c	Control Write Start Address 11	R/W	Start Address of Write Frame Store 11
BASEADDR + 0x0060	Control Write Start Address 12	R/W	Start Address of Write Frame Store 12
BASEADDR + 0x0064	Control Write Start Address 13	R/W	Start Address of Write Frame Store 13
BASEADDR + 0x0068	Control Write Start Address 14	R/W	Start Address of Write Frame Store 14
BASEADDR + 0x006c	Control Write Start Address 15	R/W	Start Address of Write Frame Store 15
BASEADDR + 0x0070	Control Read Start Address 0	R/W	Start Address of Read Frame Store 0
BASEADDR + 0x0074	Control Read Start Address 1	R/W	Start Address of Read Frame Store 1
BASEADDR + 0x0078	Control Read Start Address 2	R/W	Start Address of Read Frame Store 2
BASEADDR + 0x007c	Control Read Start Address 3	R/W	Start Address of Read Frame Store 3
BASEADDR + 0x0080	Control Read Start Address 4	R/W	Start Address of Read Frame Store 4
BASEADDR + 0x0084	Control Read Start Address 5	R/W	Start Address of Read Frame Store 5
BASEADDR + 0x0088	Control Read Start Address 6	R/W	Start Address of Read Frame Store 6

Discontinued IP

Table 1: **Video DMA pCore Memory Mapped Register Set *(Cont'd)***

Address (hex)	Register Name	Access Type	Description	
BASEADDR + 0x008c	Control Read Start Address 7	R/W	Start Address of Read Frame Store 7	
BASEADDR + 0x0090	Control Read Start Address 8	R/W	Start Address of Read Frame Store 8	
BASEADDR + 0x0094	Control Read Start Address 9	R/W	Start Address of Read Frame Store 9	
BASEADDR + 0x0098	Control Read Start Address 10	R/W	Start Address of Read Frame Store 10	
BASEADDR + 0x009c	Control Read Start Address 11	R/W	Start Address of Read Frame Store 11	
BASEADDR + 0x00a0	Control Read Start Address 12	R/W	Start Address of Read Frame Store 12	
BASEADDR + 0x00a4	Control Read Start Address 13	R/W	Start Address of Read Frame Store 13	
BASEADDR + 0x00a8	Control Read Start Address 14	R/W	Start Address of Read Frame Store 14	
BASEADDR + 0x00ac	Control Read Start Address 15	R/W	Start Address of Read Frame Store 15	
BASEADDR + 0x0F0	Version Register	R	Reports Version of the Video DMA core	
			28:31	Major Version Number. Set to 0x2.
			20:27	Minor Version Number. Set to 0x01.
			16:19	Revision Number. Set to 0xA.
			0:15	Reserved
BASEADDR + 0x021c	Global Interrupt Enable	R/W	Global Interrupt Enable	
			31:	Writing a 1 to this bit will enable all interrupts. Set to 0 (all interrupts disabled) by default.
			0:30	Reserved
BASEADDR + 0x0220	Interrupt Status/Clear	R/W	Interrupt Status when read, Interrupt Clear when written	
			19:31	Reserved
			18	Write FIFO Error
			17	Write Cmd FIFO Error
			16	Write Frame Count Interrupt Indicates that the Frame count has reached the frame count threshold. Write 1 to clear.
			15	Write Delay Count Interrupt Indicates that the delay timeout event has occurred. Write 1 to clear.

Discontinued IP

Table 1: **Video DMA pCore Memory Mapped Register Set *(Cont'd)***

Address (hex)	Register Name	Access Type	Description	
			14	Write Frame Repeat This bit indicates that the Frame Store Pointer or Start Address was repeated based on Gen-Lock synchronization. Write 1 to clear.
			13	Write Frame Skip This bit indicates that the Frame Store Pointer or Start Address was advanced by more than one Frame Store based on Gen-Lock synchronization. Write 1 to clear.
			12	Write DMA Done This bit indicates that the current Frame has completed. Write 1 to clear.
			11	Reserved
			10	Read FIFO Error
			9	Read Cmd FIFO Error
			8	Read Frame Count Interrupt Indicates that the Frame count has reached the frame count threshold. Write 1 to clear.
			7	Read Delay Count Interrupt Indicates that the delay timeout event has occurred. Write 1 to clear.
			6	Read Frame Repeat This bit indicates that the Frame Store Pointer or Start Address was repeated based on Gen-Lock synchronization. Write 1 to clear.
			5	Read Frame Skip This bit indicates that the Frame Store Pointer or Start Address was advanced by more than one Frame Store based on Gen-Lock synchronization. Write 1 to clear.
			4	Read DMA Done This bit indicates that the current Frame has completed. Write 1 to clear.
			2:3	Reserved
			1	Address Error Interrupt Current Buffer Address is Not a valid address. Valid Addresses are between C_PI_BASEADDR and C_PI_HIGHADDR. Write 1 to clear.

Discontinued IP

Table 1: **Video DMA pCore Memory Mapped Register Set *(Cont'd)***

Address (hex)	Register Name	Access Type	Description	
			0	Write Busy Error Interrupt This bit indicates that the Frame Size, Frame Stride or Frame Start Address registers were written while the DMA was busy. Write 1 to clear.
BASEADDR + 0x0228	IER – Interrupt Enable	R/W	Interrupt Enable Mask For each bit: 0 = Mask Interrupt 1 = Enable Interrupt	
			17:31	Reserved
			16	Write Frame Count Interrupt En
			15	Write Delay Count Interrupt En
			14	Write Frame Repeat En
			13	Write Frame Skip En
			12	Write DMA Done En
			9:11	Reserved
			8	Read Frame Count Interrupt En
			7	Read Delay Count Interrupt En
			6	Read Frame Repeat En
			5	Read Frame Skip En
			4	Read DMA Done En
			2:3	Reserved
			1	Address Error En
			0	Write Busy Error En

Discontinued IP

pCore Driver Files

The Video DMA pCore includes a software driver written in the C programming language that the user can use to control the Video DMA. A high-level API is provided to hide the details of the Xilinx Video DMA, and application developers are encouraged to use it to access the device features. A low-level API is also provided in case developers prefer to access the devices directly through the system registers described in the previous section.

Table 2 lists the files that are included with the Video DMA pCore driver.

Table 2: **Software Driver Files Provided with the Video DMA pCore**

File Name	Description
xvdma.h	Contains all prototypes of high-level API to access all of the features of the Xilinx Video DMA devices.
xvdma.c	Contains the implementation of high-level API to access all of the features of the Xilinx Video DMA devices except interrupts.
xvdma_intr.c	Contains the implementation of high-level API to access the interrupt feature of the Xilinx Video DMA devices.
xvdma_sinit.c	Contains static initialization methods for the Xilinx Video DMA device driver.
xvdma_g.c	Contains a template for a configuration table of Xilinx Video DMA devices. This file is used by the high-level API and is automatically generated to match the Video DMA device configuration by Xilinx EDK/SDK tools when the software project is built.
xvdma_hw.h	Contains low-level API (that is, identifiers and register-level driver API) that can be used to access the Xilinx Video DMA devices.
xvdma_i.h	Contains internal functions of the Xilinx Video DMA device driver. The application should never need to invoke any function/macro in this file.
example.c	An example that demonstrates how to control the Xilinx Video DMA devices using the high-level API.

Discontinued IP

pCore I/O Signals

The I/O signals for the Video DMA pCore are shown in Table 3. The signals can be broken into three groups: Streaming Video, pCore Gen-Lock and PLB v4.6 signals. The Streaming Video Signals are specified in Table 4. The pCore Gen-Lock signals are specified in Table 5. The PLB v4.6 signals are specified in Table 6.

The selected modes of the Video VDMA pCore determine the signals that are available to the user. When the Bus_Interface is set to VDMA, the VDMA bus interface is available and the XSVI bus interface is not. Conversely, when the XSVI is selected the XSVI bus interface is available and the VDMA bus interface is not. When the DMA_Mode is set to Read/Write, both read and write related signals are available. When Read_Only mode is selected, only read related signals are available. When Write_Only mode is selected, only write related signals are available.

Table 3: **Video DMA pCore I/O Diagram**

VFBC Command Interface	
VFBC_cmd_full VFBC_cmd_almost_full VFBC_cmd_idle	VFBC_cmd_clk VFBC_cmd_reset **VFBC_cmd_data** VFBC_cmd_write VFBC_cmd_end
VFBC Read Interface	
VFBC_rd_empty VFBC_rd_almost_empty **VFBC_rd_data**	VFBC_rd_clk VFBC_rd_reset VFBC_rd_read VFBC_rd_end_burst VFBC_rd_flush
VFBC Write Interface	
VFBC_wd_full VFBC_wd_almost_full	VFBC_wd_clk VFBC_wd_reset VFBC_wd_write VFBC_wd_end_burst VFBC_wd_flush **VFBC_wd_data** **VFBC_wd_be**
VDMA Read Command Interface	
VDMA_rcmd_full VDMA_rcmd_almost_full VDMA_rcmd_idle	VDMA_rcmd_clk VDMA_rcmd_reset **VDMA_rcmd_data** VDMA_rcmd_write VDMA_rcmd_end
VDMA Read Interface	
VFBC_rd_empty VFBC_rd_almost_empty **VFBC_rd_data**	VFBC_rd_clk VFBC_rd_reset VFBC_rd_read VFBC_rd_end_burst VFBC_rd_flush

Discontinued IP

Table 3: **Video DMA pCore I/O Diagram *(Cont'd)***

VDMA Write Command Interface	
VDMA_wcmd_full VDMA_wcmd_almost_full VDMA_wcmd_idle	VDMA_wcmd_clk VDMA_wcmd_reset **VDMA_wcmd_data** VDMA_wcmd_write VDMA_wcmd_end
VDMA Write Interface	
VDMA_wd_full VDMA_wd_almost_full	VDMA_wd_clk VDMA_wd_reset VDMA_wd_write VDMA_wd_end_burst VDMA_wd_flush **VDMA_wd_data** **VDMA_wd_be**
XSVI Read Interface	
XSVI_rd_clk_in XSVI_rd_vsync_in XSVI_rd_hsync_in XSVI_rd_vblank_in XSVI_rd_hblank_in XSVI_rd_chroma_in XSVI_rd_field_in XSVI_rd_active_video_in	XSVI_rd_clk_out XSVI_rd_vsync_out XSVI_rd_hsync_out XSVI_rd_vblank_out XSVI_rd_hblank_out XSVI_rd_chroma_out XSVI_rd_field_out XSVI_rd_active_video_out **XSVI_rd_data_out**
XSVI Write Interface	
XSVI_wd_clk_in XSVI_wd_vsync_in XSVI_wd_active_video_in **XSVI_wd_data_in**	
Gen-Lock and Frame Synchronization	
s_rd_frame_ptr_in1 **s_rd_frame_ptr_in2** **s_rd_frame_ptr_in3** **s_rd_frame_ptr_in4** **s_rd_frame_ptr_in5** **s_rd_frame_ptr_in6** **s_rd_frame_ptr_in7** **s_rd_frame_ptr_in8** **s_wd_frame_ptr_in1** **s_wd_frame_ptr_in2** **s_wd_frame_ptr_in3** **s_wd_frame_ptr_in4** **s_wd_frame_ptr_in5** **s_wd_frame_ptr_in6** **s_wd_frame_ptr_in7** **s_wd_frame_ptr_in8** fsync rd_fsync wd fsync	**m_rd_frame_ptr_out** **m_wd_frame_ptr_out**

Discontinued IP

Table 3: **Video DMA pCore I/O Diagram *(Cont'd)***

PLB Interface	
SPLB_Clk	SI_addrAck
SPLB_Rst	SI_SSize
PLB_ABus	SI_wait
PLB_UABus	SI_rearbitrate
PLB_PAValid	SI_wrDAck
PLB_SAValid	SI_wrComp
PLB_rdPrim	SI_wrBTerm
PLB_wrPrim	**SI_rdDBus**
PLB_MasterID	**SI_rdWdAddr**
PLB_abort	SI_rdDAck
PLB_buslock	SI_rdComp
PLB_RNW	SI_rdBterm
PLB_BE	**SI_MBusy**
PLB_Msize	**SI_MWrErr**
PLB_size	**SI_MRdErr**
PLB_type	**SI_MIRQ**
PLB_lockErr	IP2INTC_Irpt
PLB_wrDBus	
PLB_wrBurst	
PLB_rdBurst	
PLB_wrPendReq	
PLB_rdPendReq	
PLB_wrPendPri	
PLB_rdPendPri	
PLB_reqPri	
PLB_TAttribute	

Table 4: **Streaming Video Signals**

Name	Direction	Description
fsync	In	Frame Synchronization Input (Read_Only or Write_Only modes)
rd_fsync	In	Read Frame Synchronization Input (Read/Write mode)
wd_fsync	In	Write Frame Synchronization Input (Read/Write mode)
vfbc_cmd_clk	Out	VFBC Command Clock
vfbc_cmd_reset	Out	VFBC Command Reset
vfbc_cmd_data [31:0]	Out	VFBC Command Data
vfbc_cmd_write	Out	VFBC Command Write Enable
vfbc_cmd_end	Out	VFBC Command End
vfbc_cmd_full	In	VFBC Command Full
vfbc_cmd_almost_full	In	VFBC Command Almost Full
vfbc_cmd_idle	In	VFBC Command Idle
vfbc_wd_clk	Out	VFBC Write Data Clock
vfbc_wd_reset	Out	VFBC Write Data Reset

Discontinued IP

Table 4: **Streaming Video Signals *(Cont'd)***

Name	Direction	Description
vfbc_wd_write	Out	VFBC Write Data Write Enable
vfbc_wd_end_burst	Out	VFBC Write Data End Burst
vfbc_wd_flush	Out	VFBC Write Data Flush
vfbc_wd_data [DATA_WIDTH-1:0]	Out	VFBC Write Data
vfbc_wd_data_be [(DATA_WIDTH/8)-1:0]	Out	VFBC Write Data Byte Enable
vfbc_wd_full	In	VFBC Write Data Full
vfbc_wd_almost_full	In	VFBC Write Data Almost Full
vfbc_rd_clk	Out	VFBC Read Data Clock
vfbc_rd_reset	Out	VFBC Read Data Reset
vfbc_rd_read	Out	VFBC Read Data Read Enable
vfbc_rd_end_burst	Out	VFBC Read Data End Burst
vfbc_rd_flush	Out	VFBC Read Data Flush
vfbc_rd_data [DATA_WIDTH-1:0]	In	VFBC Read Data
vfbc_rd_empty	In	VFBC Read Data Empty
vfbc_rd_almost_empty	In	VFBC Read Data Almost Empty
vdma_wcmd_clk	In	VDMA Write Command Clock
vdma_wcmd_reset	In	VDMA Write Command Reset
vdma_wcmd_data [31:0]	In	VDMA Write Command Data
vdma_wcmd_write	In	VDMA Write Command Write Enable
vdma_wcmd_end	In	VDMA Write Command End
vdma_wcmd_full	Out	VDMA Write Command Full
vdma_wcmd_almost_full	Out	VDMA Write Command Almost Full
vdma_wcmd_idle	Out	VDMA Write Command Idle
vdma_wd_clk	In	VDMA Write Data Clock
vdma_wd_reset	In	VDMA Write Data Reset
vdma_wd_write	In	VDMA Write Data Write Enable
vdma_wd_end_burst	In	VDMA Write Data End Burst
vdma_wd_flush	In	VDMA Write Data Flush
vdma_wd_data [DATA_WIDTH-1:0]	In	VDMA Write Data
vdma_wd_data_be [(DATA_WIDTH/8)-1:0]	In	VDMA Write Data Byte Enable
vdma_wd_full	Out	VDMA Write Data Full
vdma_wd_almost_full	Out	VDMA Write Data Almost Full
vdma_rcmd_clk	In	VDMA Read Command Clock
vdma_rcmd_reset	In	VDMA Read Command Reset
vdma_rcmd_data [31:0]	In	VDMA Read Command Data
vdma_rcmd_write	In	VDMA Read Command Write Enable
vdma_rcmd_end	In	VDMA Read Command End
vdma_rcmd_full	Out	VDMA Read Command Full

Discontinued IP

Table 4: **Streaming Video Signals *(Cont'd)***

Name	Direction	Description
vdma_rcmd_almost_full	Out	VDMA Read Command Almost Full
vdma_rcmd_idle	Out	VDMA Read Command Idle
vdma_rd_clk	In	VDMA Read Data Clock
vdma_rd_reset	In	VDMA Read Data Reset
vdma_rd_read	In	VDMA Read Data Read Enable
vdma_rd_end_burst	In	VDMA Read Data End Burst
vdma_rd_flush	In	VDMA Read Data Flush
vdma_rd_data [DATA_WIDTH-1:0]	Out	VDMA Read Data
vdma_rd_empty	Out	VDMA Read Data Empty
vdma_rd_almost_empty	Out	VDMA Read Data Almost Empty
xsvi_rd_clk_in	In	XSVI Read Data Clock Input
xsvi_rd_vsync_in	In	XSVI Read Vertical Sync Input
xsvi_rd_hsync_in	In	XSVI Read Horizontal Sync Input
xsvi_rd_vblank_in	In	XSVI Read Vertical Blank Input
xsvi_rd_hblank_in	In	XSVI Read Horizontal Blank Input
xsvi_rd_chroma_in	In	XSVI Read Chroma Input
xsvi_rd_field_in	In	XSVI Read Field Input
xsvi_rd_active_video_in	In	XSVI Read Active Video Input
xsvi_rd_clk_out	Out	XSVI Read Data Clock Output
xsvi_rd_vsync_out	Out	XSVI Read Vertical Sync Output
xsvi_rd_hsync_out	Out	XSVI Read Horizontal Sync Output
xsvi_rd_vblank_out	Out	XSVI Read Vertical Blank Output
xsvi_rd_hblank_out	Out	XSVI Read Horizontal Blank Output
xsvi_rd_chroma_out	Out	XSVI Read Chroma Output
xsvi_rd_field_out	Out	XSVI Read Field Output
xsvi_rd_active_video_out	Out	XSVI Read Active Video Output
xsvi_rd_data_out [DATA_WIDTH-1:0]	Out	XSVI Read Data Output
xsvi_wd_clk_in	In	XSVI Write Data Clock Input
xsvi_wd_vsync_in	In	XSVI Write Vertical Sync Input
xsvi_wd_active_video_in	In	XSVI Write Active Video Input
xsvi_wd_data_in [DATA_WIDTH-1:0]	In	XSVI Write Data Input

Discontinued IP

Table 5: **pCore Gen-Lock Signals**

Name	Direction	Description
s_wd_frame_ptr_in1[4:0]	In	Gen-Lock Slave Write Frame Pointer Input #1
s_wd_frame_ptr_in2[4:0]	In	Gen-Lock Slave Write Frame Pointer Input #2
s_wd_frame_ptr_in3[4:0]	In	Gen-Lock Slave Write Frame Pointer Input #3
s_wd_frame_ptr_in4[4:0]	In	Gen-Lock Slave Write Frame Pointer Input #4
s_wd_frame_ptr_in5[4:0]	In	Gen-Lock Slave Write Frame Pointer Input #5
s_wd_frame_ptr_in6[4:0]	In	Gen-Lock Slave Write Frame Pointer Input #6
s_wd_frame_ptr_in7[4:0]	In	Gen-Lock Slave Write Frame Pointer Input #7
s_wd_frame_ptr_in8[4:0]	In	Gen-Lock Slave Write Frame Pointer Input #8
m_wd_frame_ptr_out[4:0]	Out	Gen-Lock Master Write Frame Pointer Output
s_rd_frame_ptr_in1[4:0]	In	Gen-Lock Slave Read Frame Pointer Input #1
s_rd_frame_ptr_in2[4:0]	In	Gen-Lock Slave Read Frame Pointer Input #2
s_rd_frame_ptr_in3[4:0]	In	Gen-Lock Slave Read Frame Pointer Input #3
s_rd_frame_ptr_in4[4:0]	In	Gen-Lock Slave Read Frame Pointer Input #4
s_rd_frame_ptr_in5[4:0]	In	Gen-Lock Slave Read Frame Pointer Input #5
s_rd_frame_ptr_in6[4:0]	In	Gen-Lock Slave Read Frame Pointer Input #6
s_rd_frame_ptr_in7[4:0]	In	Gen-Lock Slave Read Frame Pointer Input #7
s_rd_frame_ptr_in8[4:0]	In	Gen-Lock Slave Read Frame Pointer Input #8
m_rd_frame_ptr_out[4:0]	Out	Gen-Lock Master Read Frame Pointer Output

Table 6: **Processor Local Bus (PLB) v4.6 Signals**

Name	Direction	Description
SPLB_Clk	In	Slave PLB Clock
SPLB_Rst	In	Slave PLB Reset
PLB_ABus [0:C_SPLB_AWIDTH-1]	In	PLB Address Bus
PLB_PAValid	In	PLB Primary Address Valid indicator
PLB_masterID[0:C_SPLB_MID_WIDTH-1]	In	PLB Current Master identifier
PLB_abort	In	PLB Abort Bus Request indicator
PLB_RNW	In	PLB Read Not Write
PLB_BE [0:(C_SPLB_DWIDTH/8)-1]	In	PLB Byte Enables
PLB_MSize [0:1]	In	PLB Master Data Bus Size
PLB_size [0:3]	In	PLB Transfer Size
PLB_type [0:2]	In	PLB Transfer Type
PLB_wrDBus [0:C_SPLB_DWIDTH-1]	In	PLB Write Data Bus
PLB_wrBurst	In	PLB Burst Write Transfer indicator
PLB_rdBurst	In	PLB Burst Read Transfer indicator
PLB_SAValid	In	PLB Secondary Address Valid

Discontinued IP

Table 6: **Processor Local Bus (PLB) v4.6 Signals *(Cont'd)***

Name	Direction	Description
PLB_UABus[0:31]	In	PLB Upper Address Bus
PLB_BusLock	In	PLB Bus Lock
PLB_LockErr	In	PLB Lock Error
PLB_TAttribute[0:15]	In	PLB Attribute
PLB_RdPrim	In	PLB Read Primary
PLB_WrPrim	In	PLB Write Primary
PLB_RDPendPri[0:1]	In	PLB Read Pending on Primary
PLB_WrPendPri[0:1]	In	PLB Write Pending on Primary
PLB_RdPendReq	In	PLB Read Pending Request
PLB_WrPendReq	In	PLB Write Pending Request
Sl_addAck	Out	Slave Address Acknowledge
Sl_SSize[0:1]	Out	Slave Data Bus Size
Sl_wait	Out	Slave Wait Indicator
Sl_rearbitrate	Out	Slave Rearbitrate Bus indicator
Sl_wrDAck	Out	Slave Write Data Acknowledge
Sl_wrComp	Out	Slave Write Transfer Complete indicator
Sl_wrBTerm	Out	Slave Terminate Write Burst Transfer
Sl_rdDBus[0:C_SPLB_DWIDTH-1]	Out	Slave Read Data Bus
Sl_rdWdAddr[0:3]	Out	Slave Read Word Address
Sl_rdDAck	Out	Slave Read Data Acknowledge
Sl_rdComp	Out	Slave Read Transfer Complete indicator
Sl_rdBTerm	Out	Slave Terminate Read Burst Transfer
Sl_MBusy[0:C_SPLB_NUM_MASTERS-1]	Out	Slave Busy indicator
Sl_MrdErr[0:C_SPLB_NUM_MASTERS-1]	Out	Slave Read Error indicator
Sl_MwrErr[0:C_SPLB_NUM_MASTERS-1]	Out	Slave Write Error indicator
Sl_MIRQ[0:C_SPLB_NUM_MASTERS-1]	Out	Slave Interrupt
IP2INTC_Irpt	Out	Interrupt Signal

Discontinued IP

General Processor Interface

The other interface option is the General Purpose Processor (GPP) interface. The GPP Interface is shown in Table 7 and consists of the Streaming Video Signals listed in Table 4, the GPP Gen-Lock Signals listed in Table 8 and the Control, Interrupt, and Status signals detailed in Table 9. The signals in Table 9 correspond to the registers in Table 1, which has more in-depth descriptions of each signal.

The selected modes of the Video DMA core determine the signals that are available to the user. When the Bus_Interface is set to VDMA, the VDMA bus interface is available and the XSVI bus interface is not. Conversely, when the XSVI is selected the XSVI bus interface is available and the VDMA bus interface is not. When the DMA_Mode is set to Read/Write, both read and write related signals are available. When Read_Only mode is selected, only read related signals are available. When Write_Only mode is selected, only write related signals are available.

The directly exposed control, interrupt and status signals allow the user to wrap these signals with a user-defined bus interface targeting any arbitrary processor. New values written to the control signals take effect immediately. The recommendation when using this functionality is to disable the `ctrl_rd_dma_en` and `ctrl_wd_dma_en` signals before updating the control signals.

Table 7: **Video DMA General Purpose Processor I/O Diagram**

VFBC Command Interface	
VFBC_cmd_full VFBC_cmd_idle VFBC_cmd_almost_full	VFBC_cmd_clk VFBC_cmd_reset **VFBC_cmd_data** VFBC_cmd_write VFBC_cmd_end
VFBC Read Interface	
VFBC_rd_empty VFBC_rd_almost_empty **VFBC_rd_data**	VFBC_rd_clk VFBC_rd_reset VFBC_rd_read VFBC_rd_end_burst VFBC_rd_flush
VFBC Write Interface	
VFBC_wd_full VFBC_wd_almost_full	VFBC_wd_clk VFBC_wd_reset VFBC_wd_write VFBC_wd_end_burst VFBC_wd_flush **VFBC_wd_data** **VFBC_wd_be**
VDMA Read Command Interface	
VDMA_rcmd_full VDMA_rcmd_idle VDMA_rcmd_almost_full	VDMA_rcmd_clk VDMA_rcmd_reset **VDMA_rcmd_data** VDMA_rcmd_write VDMA_rcmd_end

Discontinued IP

Table 7: **Video DMA General Purpose Processor I/O Diagram *(Cont'd)***

VDMA Read Interface	
VFBC_rd_empty VFBC_rd_almost_empty **VFBC_rd_data**	VFBC_rd_clk VFBC_rd_reset VFBC_rd_read VFBC_rd_end_burst VFBC_rd_flush
VDMA Write Command Interface	
VDMA_wcmd_full VDMA_wcmd_idle VDMA_wcmd_almost_full	VDMA_wcmd_clk VDMA_wcmd_reset **VDMA_wcmd_data** VDMA_wcmd_write VDMA_wcmd_end
VDMA Write Interface	
VDMA_wd_full VDMA_wd_almost_full	VDMA_wd_clk VDMA_wd_reset VDMA_wd_write VDMA_wd_end_burst VDMA_wd_flush **VDMA_wd_data** **VDMA_wd_be**
XSVI Read Interface	
XSVI_rd_clk_in XSVI_rd_vsync_in XSVI_rd_hsync_in XSVI_rd_vblank_in XSVI_rd_hblank_in XSVI_rd_chroma_in XSVI_rd_field_in XSVI_rd_active_video_in	XSVI_rd_clk_out XSVI_rd_vsync_out XSVI_rd_hsync_out XSVI_rd_vblank_out XSVI_rd_hblank_out XSVI_rd_chroma_out XSVI_rd_field_out XSVI_rd_active_video_out **XSVI_rd_data_out**
XSVI Write Interface	
XSVI_wd_clk_in XSVI_wd_vsync_in XSVI_wd_active_video_in **XSVI_wd_data_in**	
Gen-Lock and Frame Synchronization	
rd_frame_ptr_in **wd_frame_ptr_in** fsync rd_fsync wd_fsync	**rd_frame_ptr_out** **wd_frame_ptr_out**

Discontinued IP

Table 7: **Video DMA General Purpose Processor I/O Diagram *(Cont'd)***

Control, Status and Interrupts	
sclr	intr_wd_frame_count
ctrl_wd_reset	intr_wd_delay_count
ctrl_rd_reset	intr_wd_frame_repeat
ctrl_wd_flush	intr_wd_frame_skip
ctrl_rd_flush	intr_wd_done
ctrl_rd_fstore_ptr	intr_wd_done intr_rd_frame_count
ctrl_wd_fstore_ptr	intr_rd_delay_count
ctrl_fcount_en	intr_rd_frame_repeat
ctrl_hw_lockout	intr_rd_frame_skip
ctrl_hcrop_en	intr_rd_done
ctrl_sync_en	intr_err_address
ctrl_circbuff_en	intr_err_write_busy
ctrl_rd_dma_en	intr_err_wd_fifo
ctrl_wd_dma_en	intr_err_write_busy
ctrl_wd_frame_delay	intr_err_rd_fifo
ctrl_rd_frame_delay	intr_err_rcmd_fifo
ctrl_wd_src_ptr_num	stat_busy
ctrl_rd_src_ptr_num	stat_cmd_almost_full
ctrl_rd_frame_preload	stat_cmd_full
ctrl_wd_frame_preload	stat_wd_full
ctrl_rd_delay_preload	stat_wd_almost_full
ctrl_wd_delay_preload	stat_rd_empty
ctrl_wd_vsize	stat_rd_almost_empty
ctrl_wd_hsize	stat_cmd_we
ctrl_wd_stride	stat_wd_we
ctrl_rd_vsize	stat_rd_re**stat_last_fstore**
ctrl_rd_hsize	**stat_curr_rd_fstore**
ctrl_rd_stride	**stat_curr_wd_fstore**
ctrl_wd_start_addr	**stat_rd_frame_count**
ctrl_rd_start_addr	**stat_wd_frame_count**
	stat_rd_delay_count
	stat_wd_delay_count
	stat_version

Table 8: **GPP Gen-Lock Signals**

Name	Direction	Description
wd_frame_ptr_in [("Write Number of Masters"*5) -1:0]	In	Slave Gen-Lock Write Frame Pointer Input
wd_frame_ptr_out[4:0]	Out	Master Gen-Lock Write Frame Pointer Output
rd_frame_ptr_in [("Read Number of Masters"*5) -1:0]	In	Slave Gen-Lock Read Frame Pointer Input
rd_frame_ptr_out[4:0]	Out	Master Gen-Lock Read Frame Pointer Output

Discontinued IP

Table 9: **Control, Interrupt and Status Signals**

Name	Direction	Description
ctrl_wd_reset	In	SW Write DMA Reset
ctrl_rd_reset	In	SW Read DMA Reset
ctrl_wd_flush	In	SW Write FIFO Flush
ctrl_rd_flush	In	SW Read FIFO Flush
ctrl_wd_fstore_ptr [3:0]	In	Write Frame Store Pointer
ctrl_rd_fstore_ptr [3:0]	In	Read Frame Store Pointer
ctrl_fcount_en	In	Frame Count Enable
ctrl_hw_lockout	In	HW Lockout
ctrl_fullduplex_en	In	Reserved
ctrl_hcrop_en	In	Horizontal Cropping Enable
ctrl_sync_en	In	Sync_Enable
ctrl_circbuff_en	In	Circular Buffer Enable
ctrl_rd_dma_en	In	VDMA Read Enable
ctrl_wd_dma_en	In	VDMA Write Enable
ctrl_wd_frame_delay [3:0]	In	Write Frame Delay
ctrl_rd_frame_delay [3:0]	In	Read Frame Delay
ctrl_wd_src_ptr_num [3:0]	In	Write Pointer Number
ctrl_rd_src_ptr_num [3:0]	In	Read Pointer Number
ctrl_wd_frame_preload [7:0]	In	Write Frame Count
ctrl_rd_frame_preload [7:0]	In	Read Frame Count
ctrl_wd_delay_preload [7:0]	In	Write Delay Timer Count
ctrl_rd_delay_preload [7:0]	In	Read Delay Timer Count
ctrl_wd_v_size [11:0]	In	VDMA Write Vsize
ctrl_wd_hsize [11:0]	In	VDMA Write Hsize
ctrl_wd_stride [11:0]	In	VDMA Write Stride
ctrl_rd_v_size [11:0]	In	VDMA Read Vsize
ctrl_rd_hsize [11:0]	In	VDMA Read Hsize
ctrl_rd_stride [11:0]	In	VDMA Read Stride
ctrl_wd_start_addr [32*"Frame Stores" -1 : 0]	In	Write Start Addresses Bus width based on the "Frame Stores" parameter. Valid range = 1 – 16.
ctrl_rd_start_addr [32*"Frame Stores" -1 : 0]	In	Read Start Addresses Bus width based on the "Frame Stores" parameter. Valid range = 1 – 16.
intr_wd_frame_count	Out	Write Frame Count Interrupt
intr_wd_delay_count	Out	Write Delay Count Interrupt
intr_wd_frame_repeat	Out	Write Frame Repeat Interrupt
intr_wd_frame_skip	Out	Write Frame Skip Interrupt

Discontinued IP

Table 9: **Control, Interrupt and Status Signals *(Cont'd)***

Name	Direction	Description
intr_wd_done	Out	Write DMA Done Interrupt
intr_rd_frame_count	Out	Read Frame Count Interrupt
intr_rd_delay_count	Out	Read Delay Count Interrupt
intr_rd_frame_repeat	Out	Read Frame Repeat Interrupt
intr_rd_frame_skip	Out	Read Frame Skip Interrupt
intr_rd_done	Out	Read DMA Done Interrupt
intr_err_address	Out	Address Error Interrupt
intr_err_write_busy	Out	Write Busy Error Interrupt
intr_err_wd_fifo	Out	Write FIFO Error Interrupt
intr_err_wcmd_fifo	Out	Write Cmd FIFO Error Interrupt
intr_err_rd_fifo	Out	Read FIFO Error Interrupt
intr_err_rcmd_fifo	Out	Read Cmd FIFO Error Interrupt
stat_busy	Out	VDMA Busy
stat_cmd_almost_full	Out	Cmd FIFO Almost Full
stat_cmd_full	Out	Cmd FIFO Full
stat_wd_full	Out	Write FIFO Full
stat_wd_almost_full	Out	Write FIFO Almost Full
stat_rd_empty	Out	Read FIFO Empty
stat_rd_almost_emtpy	Out	Read FIFO Almost Empty
stat_cmd_we	Out	Cmd FIFO Write Enable
stat_wd_we	Out	Write FIFO Write Enable
stat_rd_re	Out	Read FIFO Read Enable
stat_last_fstore [3:0]	Out	Max Valid Frame Store
stat_curr_wd_fstore [3:0]	Out	Current Write Frame Store
stat_curr_rd_fstore [3:0]	Out	Current Read Frame Store
stat_wd_frame_count [7:0]	Out	Write Frame Counter Value
stat_rd_frame_count [7:0]	Out	Read Frame Counter Value
stat_wd_delay_count [7:0]	Out	Write Delay Count Value
stat_rd_delay_count [7:0]	Out	Read Delay Count Value
stat_version [31:0]	Out	VDMA Version Number

Discontinued IP

Note: The Control, Interrupt, and Status Signals in Table 9 are the same as the bits in the pCore Memory Mapped Register Set. See Table 1 for more information on these signals.

Video DMA Control and Timing

The Video DMA can be operated in a number of different modes. The modes are the same for "Read Only", "Write Only", and "Read/Write" operations.

Command Format

All VDMA operations involve writing a command to the VFBC cmd port. The VFBC commands consist of 4-word packets. Table 10 shows the command packet data structure.

Table 10: **Command Packet Data Structure**

Command Packet							
Command Word 0		**Command Word 1**		**Command Word 2**		**Command Word 3**	
31:15 Reserved	14:0 X_Size(1)	31 Write_ NotRead	30:0 Start Address(1)	31:24 Reserved	23:0 Y_Size	31:24 Reserved	23:0 Stride(1)

1. The X_Size, Start Address and Stride must be aligned to a 32-word boundary. These values must be a multiple of 128 bytes and require that bits [6:0] be 0.

- Command Word 0 – Includes the X Size of the transfer, which is the number of consecutive linear bytes of the transaction per line.
- Command Word 1– Includes the direction of the transfer and the start address. Bit 31, or Write_NotRead, denotes a write transaction if high and a read transaction if low. Bits 30:0 are the physical memory byte start address, which is the start address of the transfer.
- Command Word 2 –Includes the Y Size of the transfer, which is the number of lines of the transfer minus one.
- Command Word 3 – Includes the Stride of the transfer, which is the number of bytes to skip between the start of each line of the transfer. This is the line length (in bytes) of the 2D storage in external memory.

Basic Read Operation

When configured in "Read Only" mode, the Video DMA can perform read operations in a number of different ways, all of which follow the same basic format. The basic read operation of the Video DMA is as follows:

1. The Video DMA sends a "read command" to the VFBC. This command tells the VFBC how much data to read and from where to read the data.
2. VFBC signals that it has data ready to be transferred.
3. The Video DMA inputs the data on the `vfbc_rd` port from the VFBC.
4. The Video DMA then outputs the data on the `vdma_rd` port.

The Video DMA uses FIFO flow control signals on the `vfbc_rd` and `vdma_rd` ports to control the flow of data. More detailed descriptions of the different read operations can be found in the sections that follow.

Discontinued IP

Basic Write Operation

In "Write Only" mode, the Video DMA can perform write operations in a number of different ways, all of which follow the same basic process. The basic write operation of the Video DMA is as follows:

1. The Video DMA sends a "write command" to the VFBC.
2. After the command has been written, the Video DMA can begin transferring the data to the VFBC.
3. The Video DMA inputs data on the `vdma_wd` port,
4. The Video DMA then outputs the data to the VFBC on the `vfbc_wd` port.

The Video DMA uses FIFO flow control signals on the `vdma_wd` and `vfbc_wd` ports to control the flow of data. More detailed descriptions of the different write operations can be found in the sections that follow.

Discontinued IP

Basic Read/Write Operation

In "Read/Write" mode, the Video DMA can simultaneously perform read and write operations in a number of different ways, all of which follow the basic read and basic write operations described in the two previous sections. The read side and write side share one VFBC CMD bus. As a result, there is an added restriction that only one side can write a command to the VFBC at a time.

Command Input Mode

When configured in "Write Only" mode, the `vdma_wcmd` port can be used to drive the operation of the Video DMA. A write command that is written to the `vdma_wcmd` port is then written by the Video DMA to the VFBC by way of the `vfbc_cmd` port. Once the command has been written, the full transfer can begin. See Figure 5 for a view of multiple write transactions. See Figure 6 for a view of the write command that is written to the VFBC. Once the write transaction begins, the vdma_wd_full pin should be used to control the writing of data into the Video DMA.

Note: The `vfbc_cmd_idle` signal should be active before a command is written to the `vdma_wcmd` port.

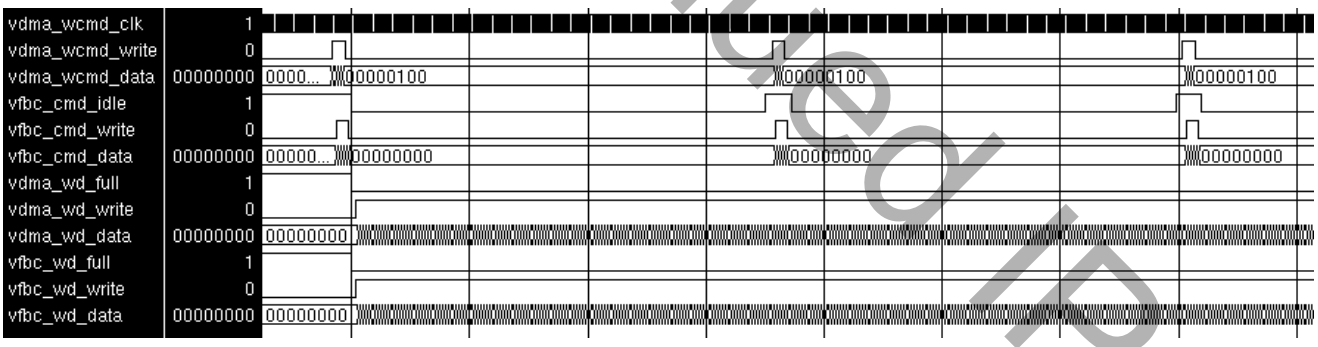

Figure 5: **Write Command Interface, Multiple Transactions**

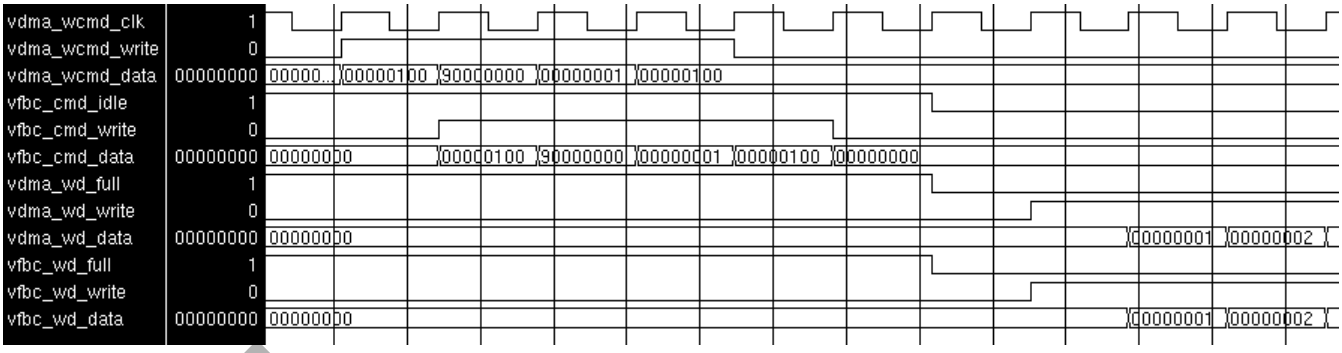

Figure 6: **Write Command Interface, Write Command**

When configured in "Read Only" mode, the Video DMA operates very similarly to the way it does in the write operation shown in Figure 5. There are two exceptions:

1. Read commands are written in on the `vdma_rcmd` port instead of the `vdma_wcmd` port.
2. The Video DMA uses the `vfbc_rd` and `vdma_rd` ports to transfer the data instead of the `vdma_wd` and `vfbc_wd` ports.

See Figure 7 for a view of the read command that is written to the VFBC. Once the read transaction begins, the `vdma_rd_empty` pin should be used to control the reading of data from the Video DMA.

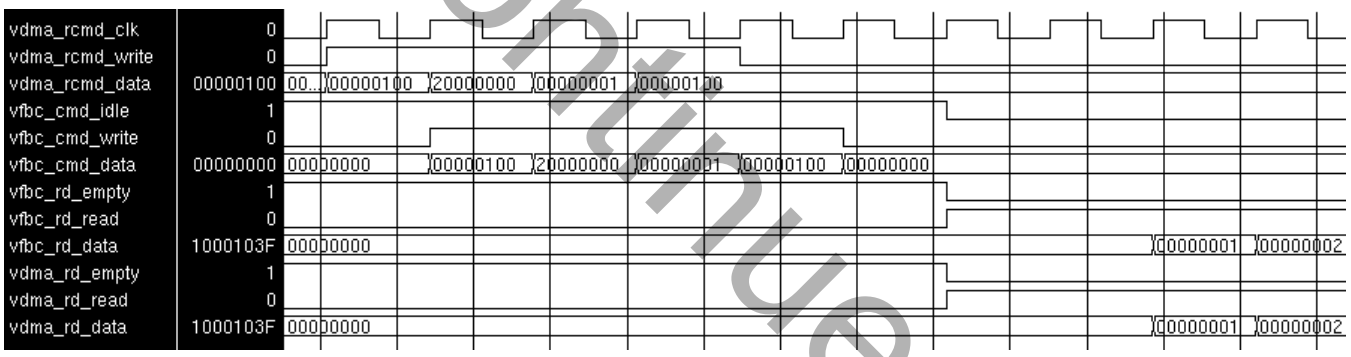

Figure 7: **Read Command Interface, Read Command**

The format of the commands input on the `vdma_wcmd` or `vdma_rcmd` ports matches the format of the VFBC commands (see the "Command Format" section for more details), with two exceptions:

1. For Command Word 1, the `Write_NotRead` (Bit 31) is not used.
2. When "Allow Non-Aligned Transfers" is selected, the `X_Size` and `Start Address` do not have to conform to the 128-byte boundary requirement.

Discontinued IP

When configured in "Read/Write" mode, the Video DMA operates very similarly to the way it does for both the write operation shown in Figure 6 and the read operation shown in Figure 7. The only restriction is that only one command can be written to the VFBC at a time. The `vdma_rcmd_full`, `vdma_rcmd_almost_full`, `vdma_wcmd_full`, and `vdma_wcmd_almost_full` signals can be used to determine when it is safe to write a command to the VFBC. Figure 8 demonstrates simultaneous read and write operations.

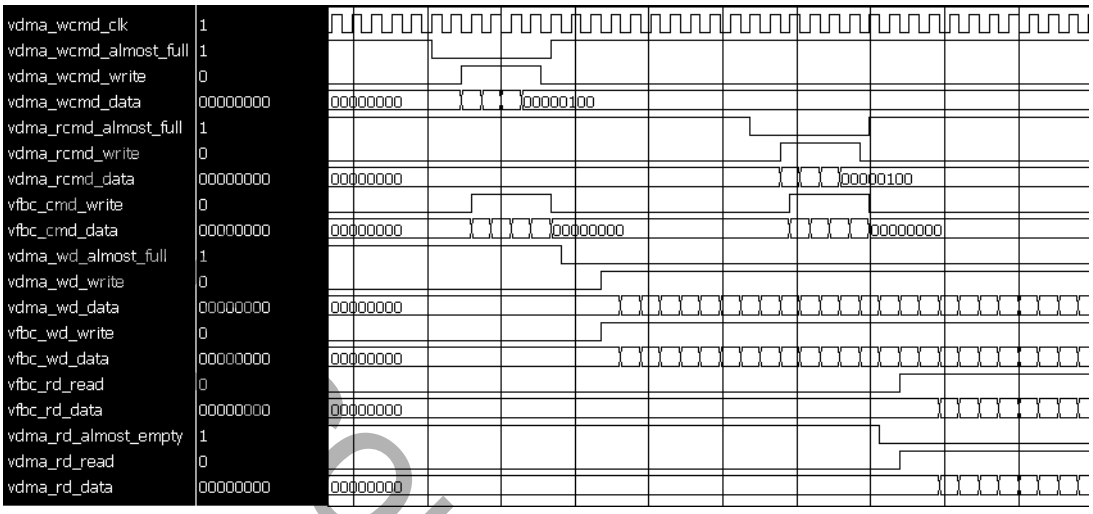

Figure 8: **Command Interface, Read and Write Commands**

Register Command Mode

The Video DMA can be configured to have between 1 and 16 read or write address registers. The number of address registers is specified in the CORE Generator GUI by the "Frame Stores" parameter. When operated in "Register Command Mode," the Video DMA uses the addresses stored in these registers when writing commands to the VFBC. Typically, each register corresponds to a frame buffer in the temporary frame storage. The address and frame store size/stride registers are not double-buffered within the VDMA. Register values are only used at the start of each frame, but care must still be taken to ensure that current values are not overwritten.

There are a number of controls that affect the operation of the core when run in Register Command Mode. These controls are discussed in the sections that follow.

Free-Running

In Free-Running Register Command Mode, the Video DMA automatically initiates transfers based upon the control settings. The core watches the `vfbc_cmd_idle` flag and waits until the VFBC is idle before writing a command to the port. The core can operate in two modes determined by "Frame Count Enable:"

1. Frame Count Enable = 0: The Video DMA initiates frame transfers indefinitely.
2. Frame Count Enable = 1: The Video DMA initiates only a specified number of frame transfers. The frame limit is set by the "Read Frame Count" for read operations or "Write Frame Count" for write operations.

Continued IP

When writing a VFBC command, the Video DMA uses the VDMA Hsize to specify the horizontal length, the VDMA Vsize to specify the vertical length, and the VDMA Stride to specify the stride between each line. There are read and write versions of each of these registers that are used for read or write commands respectively. The Start Address registers are used for the memory address in the VFBC command. The Video DMA sources the Start Address registers in one of two ways based on the Circular Buffer Enable:

1. Circular Buffer Enable = 0: The address register index stored in "Frame Store Pointer" is used to specify that Start Address register that is to be used for all transfers.
2. Circular Buffer Enable = 1: The core rotates though all of the valid Start Address registers. The valid Start Address registers are determined by the "Frame Stores" parameter which specifies the number of Frame Stores (also known as Start Address registers) to use. After reset, Start Address 0 is always the first register used in the rotation. After each transfer, the core rotates to the next Start Address register. The core circles back to the Start Address 0 once all of the valid Start Address registers have been accessed.

The trigger to begin a new transfer depends upon the "Use Frame Sync" option in the CORE Generator GUI. When "Use Frame Sync" is selected, new transfers wait for the falling edge of the `fsync` signal. See the "Fsync Synchronization" section for more details. When "Use Frame Sync" is not selected, new transfers wait for the `VFBC_Cmd_Idle` signal to become active.

Figure 9 shows an example of a Free-Running Register Write transfer. Once `vfbc_cmd_idle` is active, the Video DMA sends a write command to the VFBC and then begins the data transfer. A Free Running Register Read transfer occurs in the same fashion.

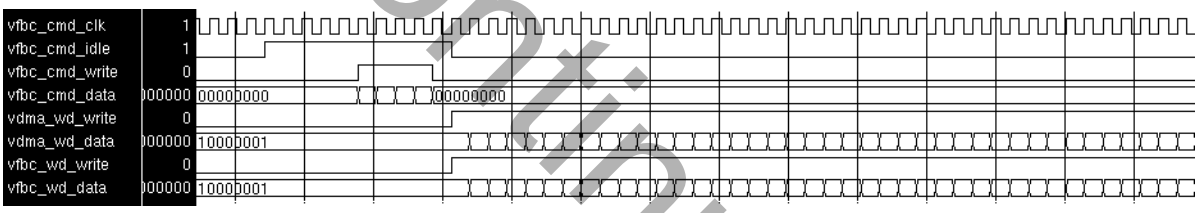

Figure 9: **Free-Running Register Write Command Mode**

Discontinued IP

Slave Gen-Lock

When used in Slave Gen-Lock mode, the Start Address register that the Video DMA uses when writing commands to the VFBC is specified by the Gen-Lock Master. See the "Gen-Lock Operation" section for details about the Gen-Lock mechanism.

Figure 10 shows the Video DMA operating as a Gen-Lock Slave for multiple read transfers. In this example, the `rd_frame_ptr_in` bus is driven by a single master. The core was configured to use 16 frames stores, so the first series of transfers follows the grey code sequence of 0, 1, 3, 2 and 6. Internal to the Video DMA, this is decoded to mean that Read Start Address registers 0, 1, 2, 3 and 4 should be used for each successive command.

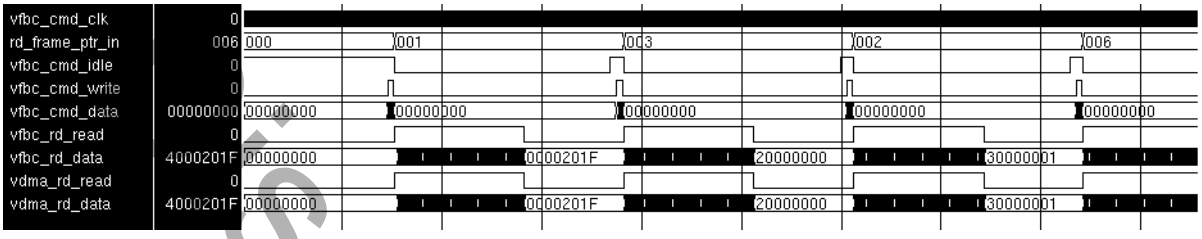

Figure 10: **Gen-Lock Slave Register Read Command Mode**

Gen-Lock Operation

In many video applications, a producer of data will run at a different rate than the consumer of that data. To avoid the potential ill effects that such a rate mismatch can cause, frame buffering is often used. Frame buffering allocates multiple frames worth of memory to be used to hold the data. The data producer writes to one buffer while the consumer reads from another. The two are kept in sync by not allowing both to use the same buffer at the same time. Typically one of the two will be forced to either skip or repeat frames as necessary. This type of synchronization is called Gen-Lock. The Video DMA has been designed to operate as either a Gen-Lock Master or Slave. Both of these modes are discussed in the following sections.

The Gen-Lock mechanism that the Video DMA implements is based around the Start Address registers. The number of Start Address registers can be configured to be between 1 and 16. The “Frame Stores” parameter in the CORE Generator is used to specify this value. The Gen-Lock Master uses the index of the Start Address register to specify which Start Address register the Gen-Lock Slave should use. This Start Address Register index is encoded as a Grey code value. The Grey code that is used depends upon the number of Frame Stores that was specified. Table 11 lists the Grey Codes that are used for each of the 16 possible Frame Store sizes. The Grey code cycles through all of the codes on the first line first and then cycles through all of the codes on the second line before repeating the first line.

discontinued IP

Table 11: **Gen-Lock Grey Codes for different Numbers of Frame Stores**

Number of Frame Stores	Grey Code
1	0 1
2	0 1 3 2
3	1 3 2 6 7 5
4	0 1 3 2 6 7 5 4
5	2 6 7 5 4 12 13 15 14 10
6	3 2 6 7 5 4 12 13 15 14 10 11
7	1 3 2 6 7 5 4 12 13 15 14 10 11 9
8	0 1 3 2 6 7 5 4 12 13 15 14 10 11 9 8
9	4 12 13 15 14 10 11 9 8 24 25 27 26 30 31 29 28 20
10	5 4 12 13 15 14 10 11 9 8 24 25 27 26 30 31 29 28 20 21
11	7 5 4 12 13 15 14 10 11 9 8 24 25 27 26 30 31 29 28 20 21 23
12	6 7 5 4 12 13 15 14 10 11 9 8 24 25 27 26 30 31 29 28 20 21 23 22
13	2 6 7 5 4 12 13 15 14 10 11 9 8 24 25 27 26 30 31 29 28 20 21 23 22 18
14	3 2 6 7 5 4 12 13 15 14 10 11 9 8 24 25 27 26 30 31 29 28 20 21 23 22 18 19
15	1 3 2 6 7 5 4 12 13 15 14 10 11 9 8 24 25 27 26 30 31 29 28 20 21 23 22 18 19 17
16	0 1 3 2 6 7 5 4 12 13 15 14 10 11 9 8 24 25 27 26 30 31 29 28 20 21 23 22 18 19 17 16

Discontinued IP

Master Mode

When the Video DMA is used in Gen-Lock Master mode, the core operates as normal. The Grey coded Start Address Register index that the slave should use is out on the `frame_ptr_out` bus. The bus name depends upon the processor interface and read/write mode being used. For the pCore interface, the `m_wd_frame_ptr_out` or `m_rd_frame_ptr_out` is used. For the GPP interface, the `wd_frame_ptr_out` or `rd_frame_ptr_out` is used.

Slave Mode

When the Video DMA is used as a Gen-Lock Slave, it uses the Grey coded Start Address Register index specified by the Gen-Lock Master to determine which Start Address Register should be used for the next VFBC command. The Video DMA can be configured to handle multiple Gen-Lock Masters, although it can follow only one Master at any one time. The number of Gen-Lock Masters is specified by the "Write Number of Masters" or the "Read Number of Masters" parameters in the CORE Generator GUI depending upon whether the core is used in "Write Only" or "Read Only" mode respectively. The number of masters can be set between 1 and 16. When using the pCore interface, the number of masters should be limited to the range of 1 – 8.

The "Read Pointer Number" and "Write Pointer Number" registers are used to specify which of the masters is in control. The registers use indexes of 0 – 15. When using the pCore interface, the `s_wd_frame_prt_in`(1-8) and `s_rd_frame_ptr_in`(1-8) buses are used to interface with the Gen-Lock masters. When using the GPP interface, the `wd_frame_prt_in` and `rd_frame_ptr_in` buses are used. For the GPP interface, all of the masters are concatenated together into a single bus. The master at the lowest portion of the bus corresponds with index 0, and so on.

Fsync Synchronization

In many video applications, it is advantageous to control when the Video DMA begins each data transfer. This synchronization can be achieved by using a system timing signal that is connected to the `fsync` pin on the core. The `fsync` signal can be derived from various sources depending upon what is available in the system. Most commonly the `fsync` is driven by a Video Timing Controller or by the `vsync` or `vblank` signal associated with a streaming video bus. When connecting the `fsync` signal to the `vblank` signal of a streaming video bus, be sure the `vblank` is active high polarity as the VDMA will internally strobe the rising edge of the `fsync` input signal. It is recommended to use the `vsync` signal if it is available to avoid resetting the internal logic of the VDMA during active video lines.

When "Use Frame Sync" parameter is selected in the CORE Generator GUI, the Video DMA waits until it sees the falling edge of the `fsync` signal before beginning a data transfer. Figure 11 shows an example of a write transfer that has been synchronized to the `fsync` signal. Read transfers follow the same format. The `fsync` signal should pulse only once per frame. Transitions while the Video DMA is processing a transfer could cause unpredictable behavior.

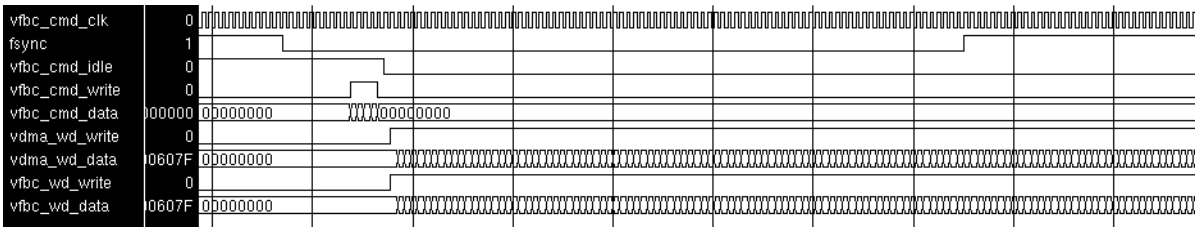

Figure 11: **Write Register Command Mode Synchronized to Fsync**

Discontinued IP

If the "Bus Interface" is set to VDMA and the "DMA Mode" is set to Read_Only or Write_Only, the `fsync` signal is used for Fsync Synchronization. If the "Bus Interface is set to VDMA and the "DMA Mode" is set to Read/Write, the `rd_fsync` and `wd_fsync` signals are used to separately control the read and write sides respectively. If the "Bus Interface" is set to XSVI, the `xsvi_rd_vsync_in` and `xsvi_wd_vsync_in` signals are used to control the read and write sides respectively.

Non-Aligned Address Commands

The VFBC requires that all command addresses conform to a 128-byte address boundary. The Video DMA was designed to allow the reads and writes that do not conform to this 128-byte address boundary. When the "Allow Non-Aligned Transfers" option is selected, the Video DMA is generated with additional resources that handle the work of converting between a non-aligned transfer on the Video DMA side and an aligned transfer on the VFBC side. This is done by rounding the non-aligned memory address down to the nearest 128-byte address boundary before it is written to the VFBC. Since extra data must be padded to the front and end of each transferred line, the "horizontal length" value must also be increased appropriately.

For write commands, the Video DMA pads the front and end of each line of the transfer so that each line conforms to the 128-byte boundaries. Figure 12 illustrates a non-aligned write operation. The command on the `vdma_cmd` port is offset by 0x04 bytes. The Video DMA rounds the address down to the nearest 128-byte boundary and sends that value to the VFBC. Since extra data must be padded to the front and end of each line to conform to the 128-byte boundaries, the "horizontal length" portion of the VFBC command must also be adjusted accordingly. The data bus for this example is 32-bits (4 bytes) wide. As a result, the Video DMA must pad one transfer to the VFBC for the transfer to conform to the VFBC 128-byte address boundaries.

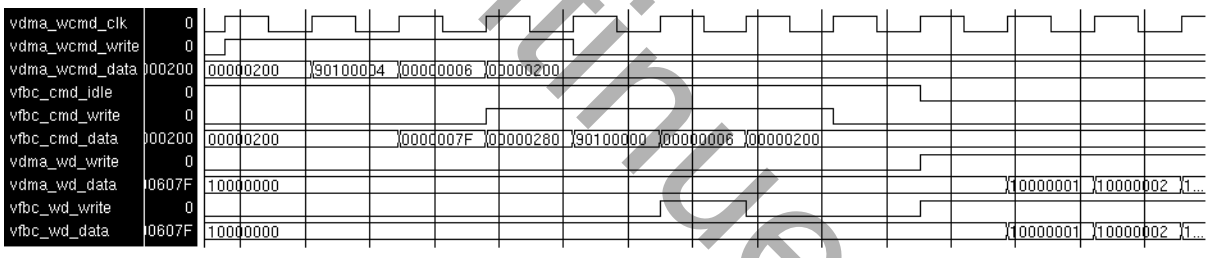

Figure 12: **Non-Aligned Write Operation (Address Non-Alignment = 0x04)**

For read commands, the Video DMA crops the data coming from the VFBC to extract only the portion of the data that the Video DMA is interested in keeping. Figure 13 is an example of a non-aligned read operation. In this example the address in the Video DMA read command is of offset by 0x1C bytes. The data bus for this example is 32-bits (4 bytes) wide. The Video DMA rounds the address written to the VFBC down to the nearest 128-byte boundary. Once the transfer begins, the Video DMA crops the first 7 transfers to account for the address non-alignment of 0x1C (7*4bytes = 28 bytes).

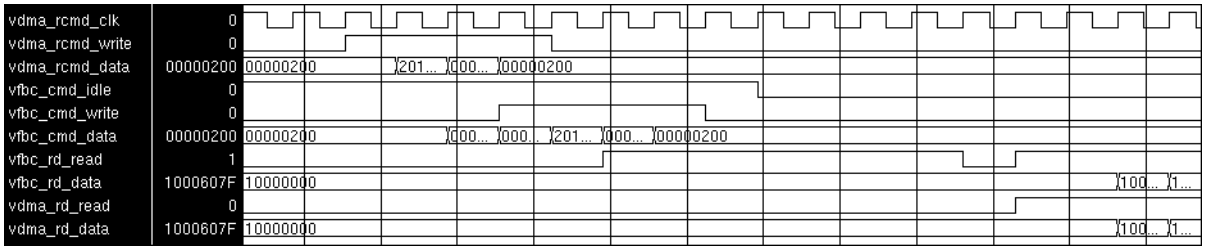

Figure 13: **Non-Aligned Read Operation (Address Non-Alignment = 0x1C)**

Discontinued IP

XSVI Data Interface

When the "Bus Interface" is set to XSVI, the VDMA FIFO data interface is replaced with the XSVI streaming data interface. The XSVI interface makes it easier to connect the Video DMA to IP cores that stream video data. When using the XSVI interface on the Video DMA, the user is responsible for guaranteeing that the logic that is connected to the VDMA's XSVI bus does not overflow or underflow the FIFOs of the VFBC. The `vdma_wd_full`, `vdma_wd_almost_full`, `vdma_rd_empty` and `vdma_almost_empty` signals are provided on the core even when the XSVI bus mode is selected. The VDMA command interfaces are also removed when the XSVI bus is selected. As a result, the Video DMA can only be used in Register Command Mode.

The write side of the XSVI bus consists of 4 input signals: `xsvi_wd_clk_in`, `xsvi_wd_vsync_in`, `xsvi_wd_write_in` and `xsvi_wd_data_in`. The `xsvi_wd_clk_in` is used to drive the write side of the VDMA. The `xsvi_wd_vsync_in` is use as the frame sync if the "Use Frame Sync" option is selected. The `xsvi_wd_active_video_in` is used to specify when valid data should be written. The `xsvi_wd_data_in` is the write side data bus.

The read side of the XSVI bus is more complicated because data is being read from a FIFO onto the streaming video bus. The user is responsible for generating the necessary streaming video control signals to properly frame the data. The VDMA provides inputs for each of the video control signals and outputs the signals on the corresponding video control output signal after a one cycle delay. The `xsvi_rd_clk_in` signal is used to drive the read side of the VDMA. The `xsvi_rd_vsync` is used for the frame sync if the "Use Frame Sync" option is selected. The `xsvi_rd_active_video_in` is used to read data from the VFBC FIFO. The `xsvi_rd_data_out` is the read side data bus.

Figure 14 illustrates read and write operations using the XSVI bus on a Video DMA in Read/Write mode. The Video DMA is operating in Register Command Mode with "Use Frame Sync" enabled. In this case the read and write frame sync signals arrive at the same time. The Video DMA sends a write command to the VFBC and commences with the write operation. Next the Video DMA sends a read command to the VFBC. Once the VFBC reports back that the read FIFO has data available, the Video DMA begins the read transfer. Note that the XSVI read control signals on the output side are delayed by one clock cycle in order to match up with the one cycle delay to read data from the VFBC read FIFO.

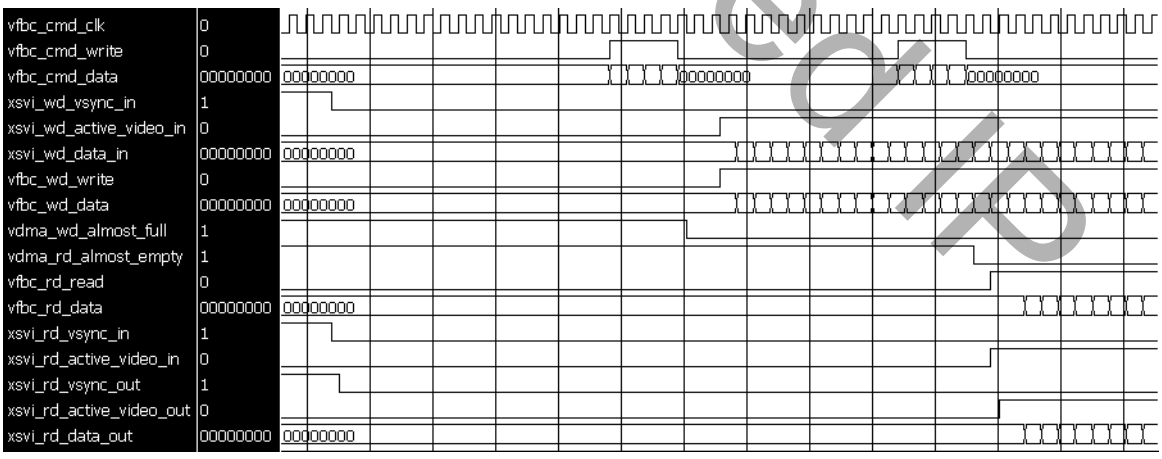

Figure 14: **Video DMA XSVI Data Interface, Read and Write Commands**

Discontinued IP

Use Model

Figure 15 is an example system with two Video DMA (VDMA) controllers connected to Video IP and VFBC PIMs. Each VDMA passes the VFBC signals to the Video IP, allows setting DMA transactions via the PLB bus by the CPU, and can be controlled by a Gen-Lock interface to allow synchronization to the other VDMA module.

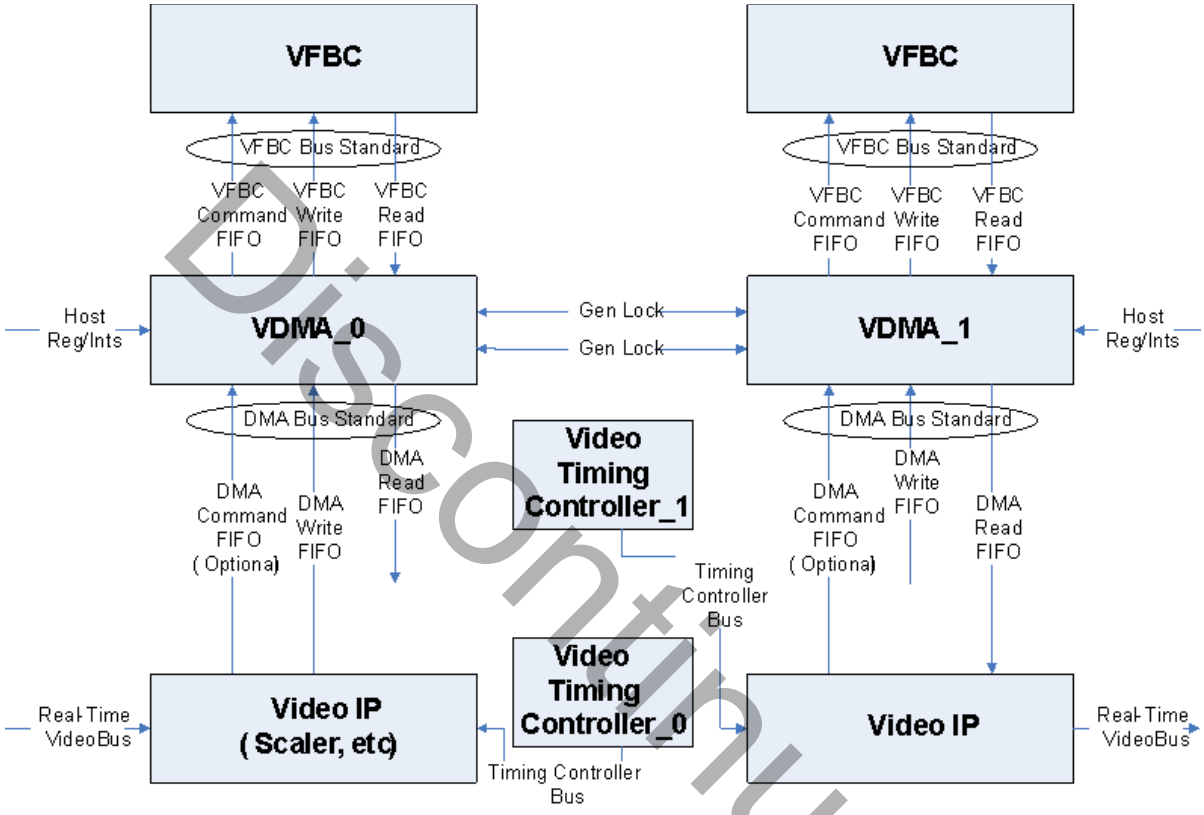

Figure 15: **Video DMA Example Use Model**

Discontinued IP

Figure 16 is a similar system with two Video DMA controllers. In this system, the connections between the VDMAs and the Video IP blocks use the XSVI Bus Standard instead of the DMA Bus Standard. Although the connection between the VDMA and the Video IP is a streaming interface, the connection between the VDMA and the VFBC is a FIFO interface. When using the XSVI Bus Standard, the user is responsible for guaranteeing that the Video IP block does not write when the VFBC Write FIFO is full or read when the VFBC Read FIFO is empty.

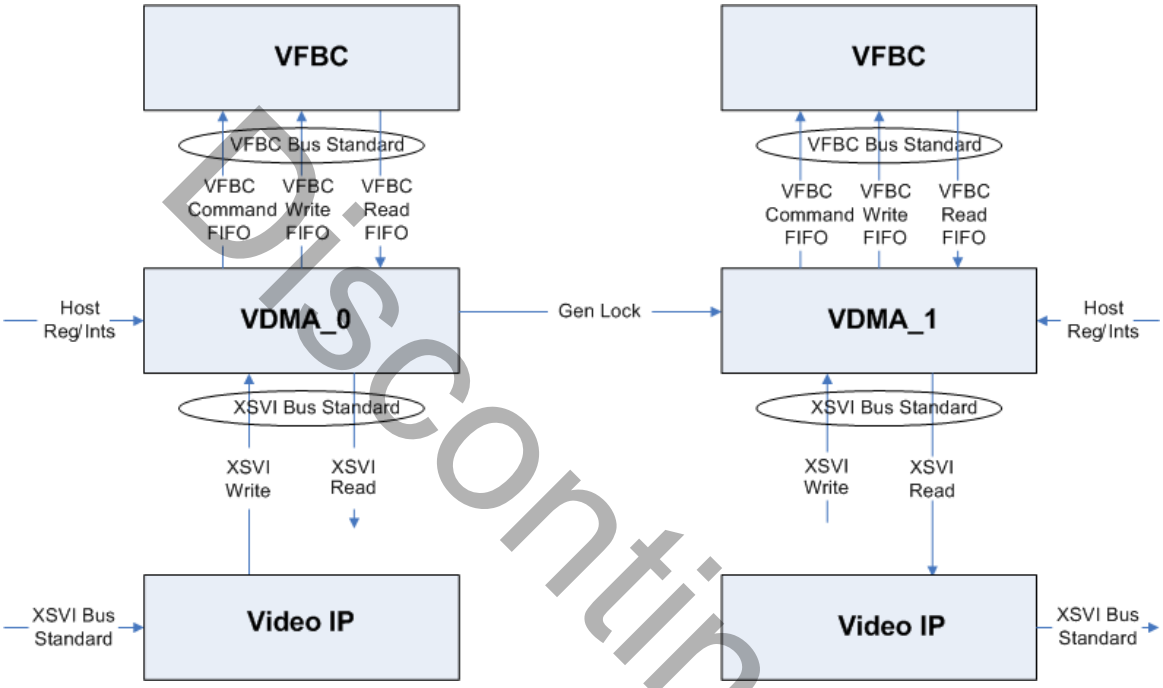

Figure 16: **Video DMA Example Using XSVI Bus Interface**

Discontinued IP

Core Resource Utilization

Resources required for the Video DMA have been estimated for the Spartan®-3A DSP (Table 12), Spartan-6 (Table 13), Virtex®-5 (Table 14) and Virtex-6 (Table 15). These values were generated using the Xilinx CORE Generator tools v12.3. They are derived from post-synthesis reports, and may change during MAP and PAR.

The resource estimates for the Video DMA are not affected by the data width. To calculate the total resources used by a particular implementation of the Video DMA, follow these steps:

- Select the FPGA Family that will be used. (i.e., Spartan-3A DSP, Spartan-6, Virtex-5 or Virtex-6)
- Use the corresponding Resource Estimate table for the following steps.
- Select the Video DMA's mode of operation (i.e., Read_Only, Write_Only or Read/Write)
- Use the resource estimates under the selected mode of operation for the following calculations.
- Use the resource numbers from the "Core (1 Frame Store)" as the base of the resource estimate.
- If using the pCore Interface:
  - Add the "pCore Interface" values to the resource estimate.
  - For each additional Frame Store, add the "Each Additional Frame Store (pCore)" values to the estimate. (For example, if using 3 Frame Stores, add the values two times.)
- If using the General Purpose Processor (GPP) Interface:
  - For each additional Frame Store, add the "Each Additional Frame Store (GPP)" values to the estimate. (For example, if using 8 Frames Stores, add the values seven times.)
- If using Non-Aligned Transfers, add the "Non-Aligned Transfers" values to the estimate.

Table 12: **Spartan-3A DSP Resource Estimates**

Feature	LUTs	FFs
Read_Only or Write_Only Mode		
Core (1 Frame Store)	294	206
Each Additional Frame Store (GPP)	24	4
Each Additional Frame Store (pCore)	32	50
Non-Aligned Transfers	184	147
pCore Interface	374	848
Read/Write		
Core (Read/Write, 1 Frame Store)	492	296
Each Additional Frame Store (GPP)	59	7
Each Additional Frame Store	95	98
Non-Aligned Transfers	303	199
(pCore) pCore Interface	374	848

Discontinued IP

Table 13: **Spartan-6 Resource Estimates**

Feature	LUTs	FFs
Read_Only or Write_Only Mode		
Core (1 Frame Store)	208	201
Each Additional Frame Store (GPP)	13	4
Each Additional Frame Store (pCore)	42	88
Non-Aligned Transfers	215	147
pCore Interface	464	963
Read/Write		
Core (Read/Write, 1 Frame Store)	341	282
Each Additional Frame Store (GPP)	47	5
Each Additional Frame Store	87	140
Non-Aligned Transfers	235	200
(pCore) pCore Interface	464	963

Table 14: **Virtex-5 Resource Estimates**

Feature	LUTs	FFs
Read_Only or Write_Only Mode		
Core (1 Frame Store)	251	206
Each Additional Frame Store (GPP)	14	4
Each Additional Frame Store (pCore)	20	43
Non-Aligned Transfers	140	147
pCore Interface	418	925
Read/Write		
Core (Read/Write, 1 Frame Store)	418	296
Each Additional Frame Store (GPP)	26	4
Each Additional Frame Store	34	82
Non-Aligned Transfers	267	199
(pCore) pCore Interface	418	925

Discontinued IP

Table 15: **Virtex-6 Resource Estimates**

Feature	LUTs	FFs
Read_Only or Write_Only Mode		
Core (1 Frame Store)	213	201
Each Additional Frame Store (GPP)	13	4
Each Additional Frame Store (pCore)	42	88
Non-Aligned Transfers	213	147
pCore Interface	625	976
Read/Write		
Core (Read/Write, 1 Frame Store)	330	282
Each Additional Frame Store (GPP)	29	4
Each Additional Frame Store	95	140
Non-Aligned Transfers	246	200
(pCore) pCore Interface	625	976

Discontinued IP

Performance

The following are typical clock frequencies for the target families. The maximum achievable clock may vary and can depend on the size of the device, various aspects of the system design and other variables.

- Spartan-3A DSP: 150 MHz
- Spartan-6: 150 MHz
- Virtex-5: 225 MHz
- Virtex-6: 225 MHz

The Video DMA does not limit the throughput of the VFBC. When using the "Allow Non-Aligned Transfers" option, delays may be added to the front and end of each transfer. The amount of delay is dependent upon the selected data width. The maximum total delay for each valid data width is:

- 8- Bit: 127 cycles
- 16-Bit: 63 cycles
- 32-Bit: 7 cycles
- 64-Bit: 3 cycles

References

1. Processor Local Bus (PLB) v4.6
2. MPMC Data Sheet (VFBC PIM)
3. Video Timing Controller Data Sheet

Support

Xilinx provides technical support for this LogiCORE product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY*.

License Options

The Xilinx Video Direct Memory Access LogiCORE system provides three licensing options. After installing the required Xilinx ISE® software and IP Service Packs, choose a license option:

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess the core functionality with either the provided example design or alongside your own design and demonstrates the various interfaces on the core in simulation. (Functional simulation is supported by a dynamically-generated HDL structural model.)

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place and route the design, evaluate timing, and perform back-annotated gate-level simulation of the core.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before timing out (ceasing to function), at which time it can be reactivated by reconfiguring the device.

Full

The Full license key is provided when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

- Functional simulation support
- Back-annotated gate-level simulation support
- Full implementation support including place and route and bitstream generation
- Full functionality in the programmed device with no time outs

Obtaining Your License Key

This section contains information about obtaining a simulation, full system hardware, and full license keys.

Simulation License

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

Discontinued IP

Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license:

1. Navigate to the product page for this core.
2. Click Evaluate.
3. Follow the instructions to install the required Xilinx ISE software and IP Service Packs.

Obtaining a Full License

To obtain a Full license key, you must purchase a license for the core. After doing so, click the "Access Core" link on the Xilinx.com IP core product page for further instructions.

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

Ordering Information

The Video Direct Memory Access v1.1 core is provided under the SignOnce IP Site License and can be generated using the Xilinx CORE Generator system v12.3 or higher. The CORE Generator system is shipped with Xilinx ISE Design Suite development software.

Please contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE modules and software. Information about additional Xilinx LogiCORE modules is available on the Xilinx IP Center.

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
09/16/09	1.0	Initial Xilinx release.
04/19/10	1.1	Revised "Command Input Mode" to include "read" command; added Simulator Support section.
09/21/10	2.0	Updated for 12.3 release.

Notice of Disclaimer

Xilinx is providing this product documentation, hereinafter "Information," to you "AS IS" with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice. XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.

Discontinued IP