Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

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.

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.

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.

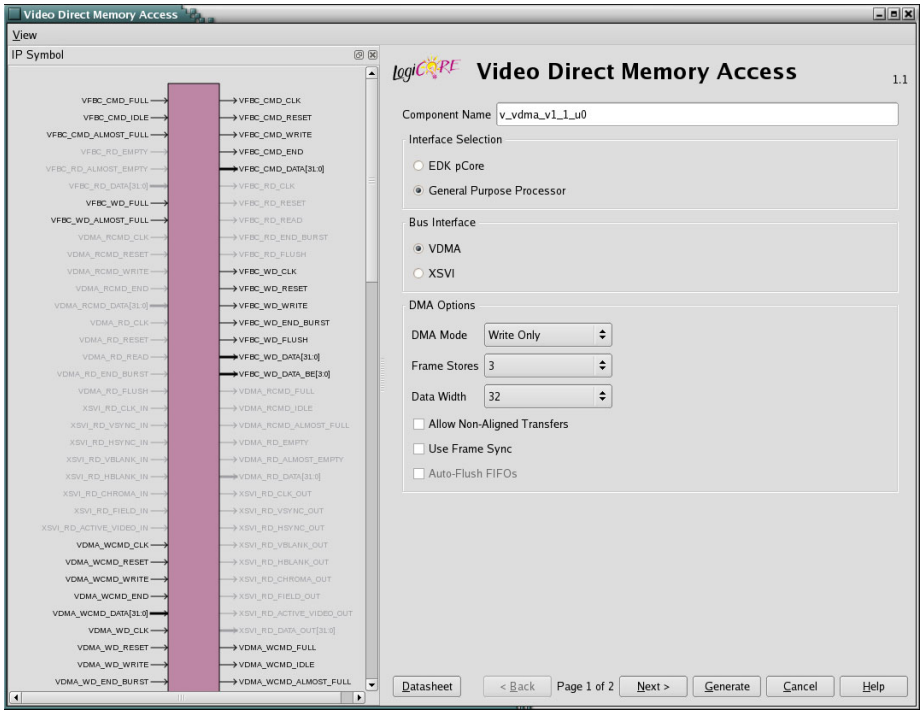

Figure 1: **Video DMA Main Screen**

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.

- **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.

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

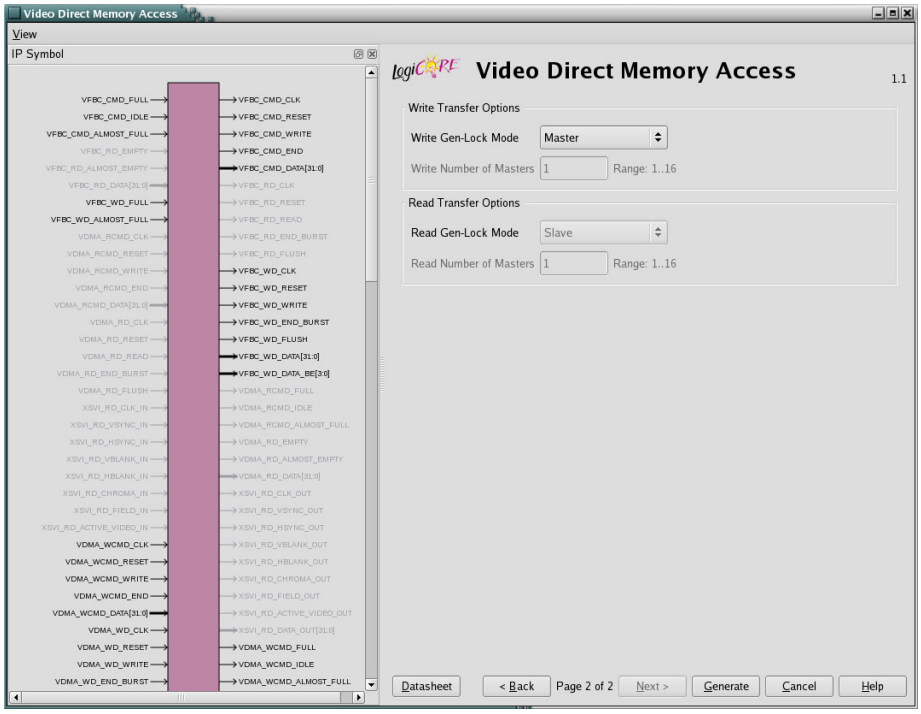

Figure 2: **Video DMA, Gen-Lock Options Screen**

- **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.

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.

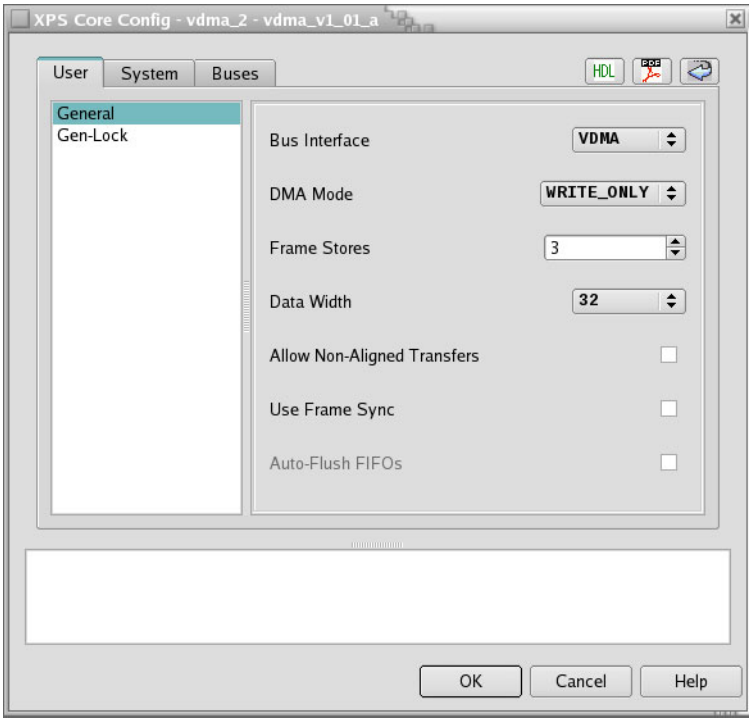

Figure 3: **Video DMA General Screen**

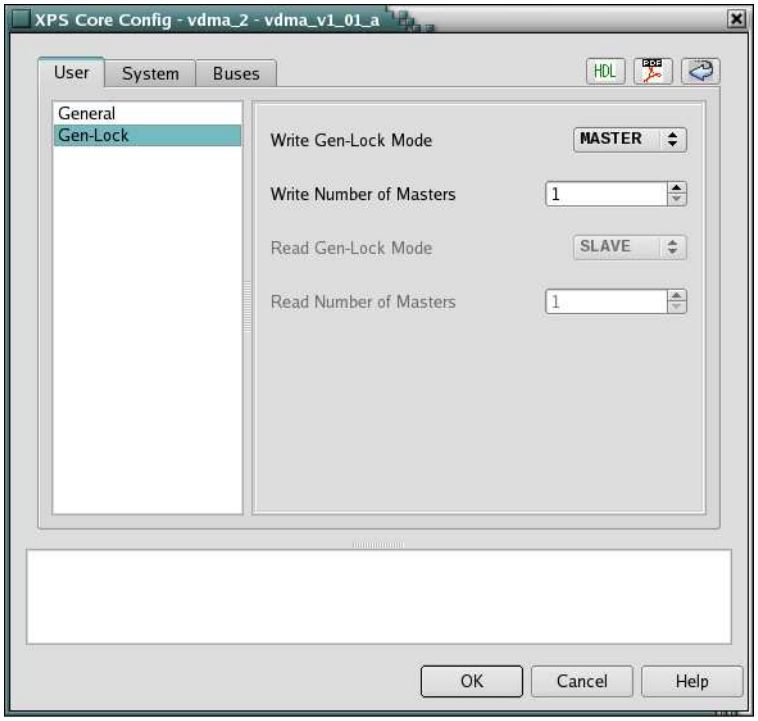

Figure 4: **Video DMA Gen-Lock Screen**

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.

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.

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.

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.

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.

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

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.

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.

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

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.

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

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**

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

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

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

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

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

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