Spartan-3A DSP FPGA Video Starter Kit

Software User Guide

UG514 (v1.1) March 15, 2010

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/17/08	1.0	Initial Xilinx release.
03/15/10	1.1	Various textual changes made; software versions updated; memory maps added.

Table of Contents

Chapter 4: Software Projects

Chapter 5: Software Development Flow

List of Figures

List of Tables

Preface

About This Guide

The *Spartan-3A DSP Video Starter Kit User Guide* provides information about how to use the Video Starter Kit (VSK) to begin experimenting with video processing using the Spartan®-3A DSP family of FPGAs.

Guide Contents

This manual contains the following chapters:

- Chapter 1, "Getting Started," details the preliminary information and tools needed to begin using the VSK.
- Chapter 2, "Libraries," provides information about the dependencies and details of the libraries used with the VSK.
- Chapter 3, "Drivers," gives an overview and software interface description for the VSK drivers.
- Chapter 4, "Software Projects," describes the software and base platform for the VSK.
- Chapter 5, "Software Development Flow,"provides steps and screenshots for the software compilation, viewing code, and downloading.

References

1. UG456, *Spartan-3A DSP FPGA Video Starter Kit User Guide*
2. *DVI Standard*, www.ddwg.org/lib/dvi_10.pdf
3. UG458, *XtremeDSP™ Solution FMC-Video Daughter Board Technical Reference Guide*
4. ISE Design Suite 11: Installation, Licensing, and Release Notes
5. DS643, *Multi-Port Memory Controller (MPMC) Data Sheet*
6. UG498, *XtremeDSP Development Platform: Spartan-3A DSP 3400A Edition User Guide*
7. DS610, *Spartan-3A DSP FPGA Family Data Sheet*

Additional Resources

To find additional documentation, see the Xilinx website at:

www.xilinx.com/support/documentation/index.htm.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

www.xilinx.com/support/mysupport.htm.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File →Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *User Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**`lowpwr`** ={**`on`**|**`off`**}
Vertical bar |	Separates items in a list of choices	**`lowpwr`** ={**`on`**|**`off`**}
Angle brackets < >	User-defined variable or in code samples	<directory name>

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis . . .	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Convention	Meaning or Use	Example
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	**`usr_teof_n`** is active low.

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

Getting Started

Introduction

The Xilinx Spartan®-3A DSP Video Starter Kit (VSK) provides a platform that allows you to quickly begin experimenting with video processing using the Spartan-3A DSP family of FPGAs. The purpose of this document is to provide important information for software developers who are working with this kit.

Figure 1-1 shows how the various software elements provided with VSK interact. The implication here is that any two layers sharing an edge have a dependency. The only exception to this is the edge between the VSK I/O Library and VSK Drivers, which do not interact. At the lowest level are the EDK libraries and drivers. This software layer supports all of the EDK IP such as the MicroBlaze™, MPMC, I2C controller, and UART. For details about the EDK IP used by VSK, see the "Base Platform" section in [Ref 1]. Video applications depend on all the layers shown, but only those above the EDK libraries and drivers are discussed in this document.

This document describes the libraries, drivers, and application software delivered as part of VSK, as well as the necessary tools to build upon it.

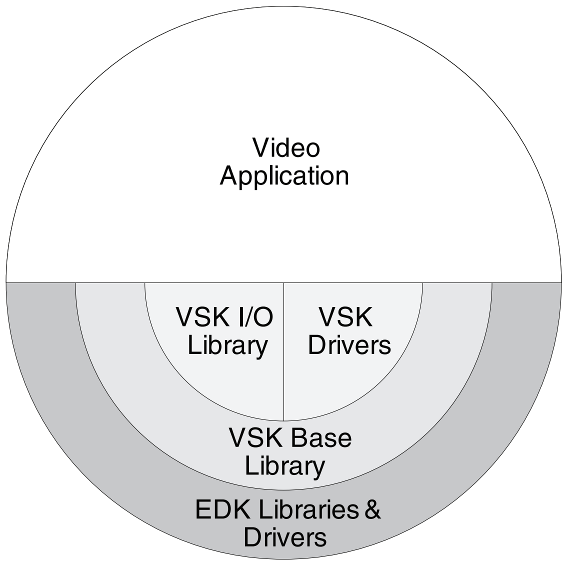


Figure 1-1: **VSK Software Model**

Code Location

To begin working with the VSK software, copy the `EDK_Demonstrations.zip` from the Demonstrations directory of the VSK CD to the hard drive. After the archive is unzipped, the code is visible. To access this archive online, go to:

www.xilinx.com/support/documentation/boards_and_kits/ug456.zip

The path names in this document are given relative to this EDK_Demonstrations directory.

The VSK libraries can be found under the following path:

```
EDK_Demonstrations\VSK_Repository\VSKProcessorIPLib\sw_services
```

The VSK drivers can be found under the following path:

```
EDK_Demonstrations\VSK_Repository\VSKProcessorIPLib\drivers
```

The application software for each of the VSK demonstrations can be found under:

```
EDK_Demonstrations\<Demo_Name>_Demo\<Demo_Name>_Sw\src
```

Note: The sources for the applications can also be viewed under the Applications tab when the design is opened in EDK.

Tools

To begin working with the VSK software, EDK 11.4 is required. EDK is part of the ISE® Design Suite. Information on installation and setup of EDK can be found at:

www.xilinx.com/support/documentation/dt_ise11-1.htm

For more information on working with the tools, see Chapter 5, “Software Development Flow.”

Chapter 2

Libraries

The libraries provided with VSK form the foundation of the software platform and provide the most direct access to the underlying video hardware. Functions in these libraries are required for initializing and configuring the hardware platform to initiate bringing video on and off of the board. At the lowest level is the VSK Base (vsk_base) library. The VSK I/O library is built on VSK Base and provides access to the video I/O interfaces.

The libraries included in an EDK design can be selected under **Software** → **Software Platform Settings** of Xilinx Platform Studio. For more information, see documents listed in "References."

Library Hierarchy

The VSK Base library is consists of six modules. All of these modules consist of C code and header files with the exception of vsk_iic_devices. vsk_iic_devices has only a header file since it is simply a listing of the addresses of all the hardware devices used by VSK.

Figure 2-1 shows the internal hierarchy within the VSK Base library.

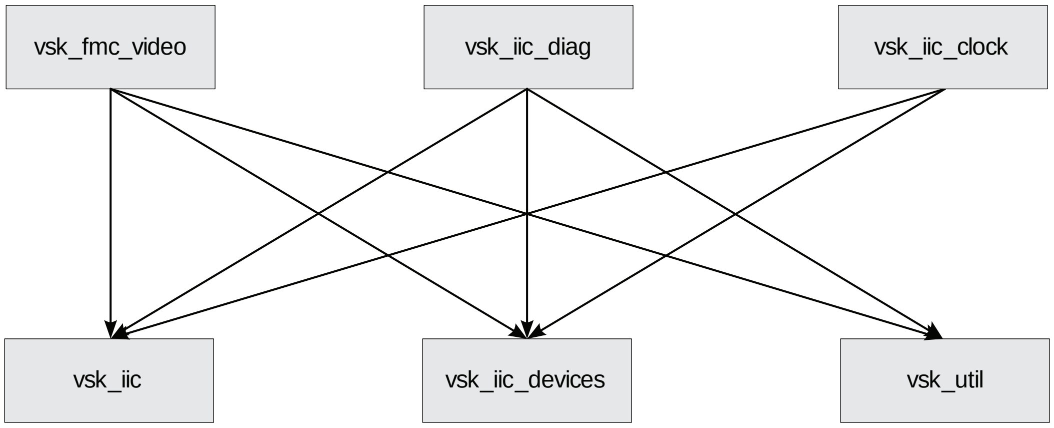


Figure 2-1: **VSK Base Library Dependencies**

Building on the VSK Base library is the VSK I/O library. The main distinction between the VSK Base library and the VSK I/O library is that each of the modules making up the VSK I/O library also has a corresponding PCORE which implements hardware for the video interface it is named after. The five VSK I/O modules and their dependencies are shown in Figure 2-2.

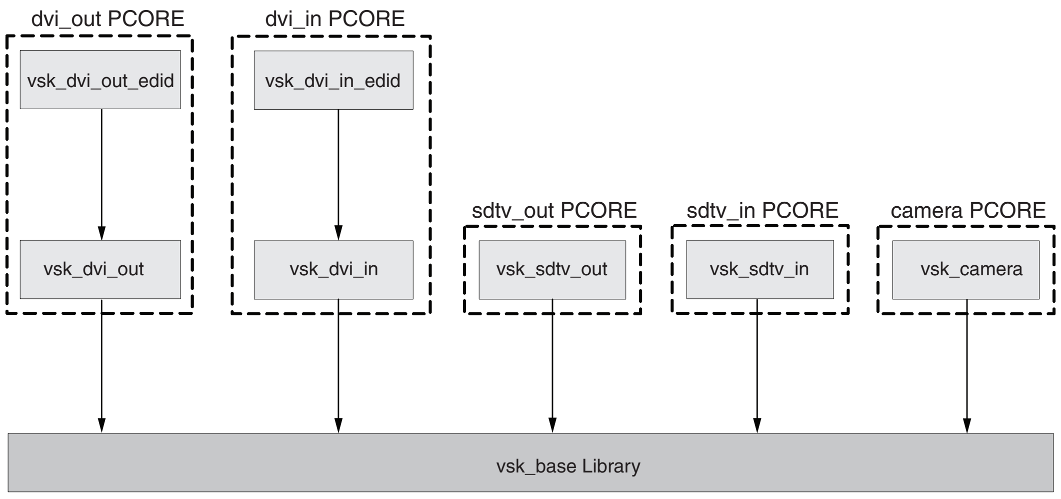


Figure 2-2: **VSK I/O Library Dependencies**

The dependencies described in this section are reflected in the configuration files for each of the libraries. As such, including one of the VSK I/O libraries in a design will automatically include the VSK Base library.

Library Details

VSK Base (v3.00a) Overview

The VSK Base library provides a set of functions and constants necessary for system initialization and communication. This library is required by all the VSK designs.

The sources making up VSK Base library are as follows:

- **vsk_fmc_video** – Functions for initializing FMC-Video
- **vsk_iic** – I2C interface routines
- **vsk_iic_clock** – Programmable clock driver
- **vsk_iic_devices** (header only) – Listing of VSK board-level device addresses
- **vsk_iic_diag** – Diagnostic menu for I2C devices
- **vsk_util** – Basic helper functions

Example

The following is a code snippet of calls into vsk_base which happens at the start of all the VSK demos in `vsk_top.c`:

```
// Initialize the IIC cores
vsk_iic_init();
// Initialize the Power Regulator to provide 3.3v to the FMC
vsk_fmc_power_init();
// Reset and Initialize the FMC devices
vsk_fmc_reset();
```

This code should appear at the start of any design making use of the VSK platform.

VSK I/O Overview

The VSK I/O library consists of five modules, each of which corresponds to a PCORE by the same name. Each of these PCOREs supports multiple modes and video resolutions. The code in these software modules does not communicate with the logic of the associated PCORE, but rather uses I2C programming to configure the associated hardware external to the FPGA. The I2C programming is handled by way of IIC PCOREs connected to the MicroBlaze™ processor.

Note: Because the devices VSK I/O libraries communicates with are not directly attached to the MicroBlaze processor, it is necessary to manually include them in your design. Simply including their associated PCORE is not sufficient for the libraries to be pulled in automatically.

camera (v2.00a)

The camera PCORE provides a connection to the camera that is included with the Xilinx Video Starter Kit. The camera is based on the Micron MT9V022 Digital Image Sensor.

The primary purpose of the camera library is to:

- Configure and initialize the camera.
- Enable and disable various features of the camera.

For an example of how to use this library, see the "Camera Frame Buffer Demo" in Chapter 4.

dvi_in (v2.00a)

The dvi_in PCORE provides a connection to the TFP403 DVI Receiver chip as well as to the AD9984A Video ADC chip on the FMC-Video card. The library for this PCORE uses IIC commands to configure these devices and associated logic.

The primary purpose of the dvi_in library is to:

- Configure the DVI input port for the desired mode (analog or digital) and resolution.
- Detect the presence of video on the DVI input port.
- Read and write the EDID code of the DVI input.

All of these functions are provided in the vsk_dvi_in source files except those for the EDID. The EDID functions are provided in vsk_dvi_in_edid. For an example of how to use this library, see "DVI Pass-Through Demo" or "DVI Frame Buffer Demo" in Chapter 4.

dvi_out (v3.00a)

The dvi_out PCORE provides a connection to the CH7301C DVI Transmitter Device on the Xilinx Video Starter Kit. The library for this PCORE uses IIC commands to configure these devices and associated logic.

The primary purpose of the dvi_out library is to:

- Configure the DVI output port for the desired resolution (analog and digital are always enabled).
- Detect the presence of a display on the DVI output port.
- Read the EDID code of the display connected to the DVI output.

All of these functions are provided in the vsk_dvi_out source files except those for the EDID. The EDID functions are provided in vsk_dvi_out_edid. For an example of how to use this library, see the "DVI Frame Buffer Demo," or "Camera Frame Buffer Demo" in Chapter 4.

sdtv_in (v2.00a)

The sdtv_in PCORE provides a connection to the ADV7180 chip on the FMC-Video card. The library for this PCORE uses IIC commands to configure these devices and associated logic.

The primary purpose of the sdtv_in library is to:

- Configure the video port for the desired interface (S-Video or Composite) and resolution (NTSC or PAL).
- Detect the presence and type of video on the selected interface.

For an example of how to use this library, see the "S-Video Frame Buffer Demo" in Chapter 4.

sdtv_out (v3.00a)

The sdtv_out PCORE provides a connection to the ADV7179 on the FMC-Video card. The library for this PCORE uses IIC commands to configure these devices and associated logic.

The primary purpose of the sdtv_out library is to configure the video port for the desired resolution (NTSC or PAL). The S-Video and Composite interfaces are always enabled while the sdtv_out library is in use.

For an example of how to use this library, see the "S-Video Frame Buffer Demo" in Chapter 4.

XILINX®

Chapter 3

Drivers

Similar to the VSK I/O Library, each of the VSK drivers is associated with a PCORE. However, the drivers are different because they communicate directly with the PCORE itself. The drivers are automatically included when the associated PCORE is added to a design.

Driver Details

de_gen (v2.00a)

Overview

The de_gen PCORE provides the ability to generate a data enable (DE) signal as well as a field signal for video streams that do not have them. For VSK, this means analog DVI and S-Video. The importance of a DE signal is that it can be used to distinguish active video from blanking.

Software Interface

There are two components to de_gen: a timing detector and a DE generator. The timing detector measures the horizontal and vertical size of the incoming image. This information is available through the function calls `read_DE_GEN_hsize()` and `read_DE_GEN_vsize()` which are defined in the de_gen driver (`de_gen.c`). For more information on the de_gen.c API, see the documentation in the header file:

```
\Demonstrations\EDK\EDK_Demonstrations\VSK_Repository\
VSKProcessorIPLib\drivers\de_gen_v2_00_a\src\de_gen.h
```

`de_generator.c` provides a debug menu for checking the status and configuring various aspects of the de_gen PCORE. The DE signal is generated by combining the information provided by the timing detector with front porch and back porch values that are programmed into it. The MicroBlaze™ processor writes the porch values to the block over the PLB interface. The de_gen block can be deactivated when the video source is digital DVI since the data enable signal is already generated by the source. For more information on the de_generator.c API, see the documentation in the header file:

```
\Demonstrations\EDK\EDK_Demonstrations\VSK_Repository\
VSKProcessorIPLib\drivers\de_gen_v2_00_a\src\de_generator.h
```

Examples of this driver in use can be found in the "DVI Pass-Through Demo," "DVI Frame Buffer Demo," and "S-Video Frame Buffer Demo" in Chapter 4.

display_controller (v3.00a)

Overview

The display_controller PCORE reads video frames out of memory from a VFBC interface of MPMC and displays them by applying the correct timing signals.

Software Interface

The display_controller PCORE generates properly timed video synchronization signals for the given resolution based on software configuration registers and the frequency of the display clock input. The display clock is generated by an external clock generation chip. Control over the display clock is provided by the vsk_iic_clock driver provided in the VSK Base library, described in "VSK Base (v3.00a) Overview" in Chapter 2.

Software configurable registers are used for the locations of the frame buffers, holding VFBC commands, and video sync timing information. These registers give the display_controller the flexibility to support a wide range of resolutions. The range of supported resolutions is limited by the frequency of the display clock (currently limited to 74.25 MHz) and the width of the timing registers. For more information on the display_controller.c API, see the documentation in the header file:

```
\Demonstrations\EDK\EDK_Demonstrations\VSK_Repository\
VSKProcessorIPLib\drivers\display_controller_v3_00_a\src\
display_controller.h
```

The highest resolution that is currently supported is 1920x1080p.

The memory mapping of frame buffers uses the same convention as video_to_vfbc. See "video_to_vfbc (v2.00a)," page 23, for details. For examples of how to use the display_controller, see any of the Frame Buffer demos.

For convenience, the video standards listed in Table 3-1 are pre-defined in `display_controller.h`. These are the only video standards with which the display_controller has been tested.

Table 3-1: **Video Standards Defined in display_controller.h**

Video Standard	Constant Name
640x480p @ 60 Hz	DISPLAY_RESOLUTION_VGA
720x480p @ 60 Hz	DISPLAY_RESOLUTION_480P60
720x576p @ 50 Hz	DISPLAY_RESOLUTION_576P50
800x600p @ 60 Hz	DISPLAY_RESOLUTION_SVGA
1024x768p @ 60 Hz	DISPLAY_RESOLUTION_XGA
1280x720p @ 50 Hz	DISPLAY_RESOLUTION_720P50
1280x720p @ 60 Hz	DISPLAY_RESOLUTION_720P60
1920x1080i @ 50 Hz	DISPLAY_RESOLUTION_1080I50
1920x1080i @ 60 Hz	DISPLAY_RESOLUTION_1080I60
1920x1080p @ 25 Hz	DISPLAY_RESOLUTION_1080P25
1920x1080p @ 30 Hz	DISPLAY_RESOLUTION_1080P30
NTSC	DISPLAY_RESOLUTION_NTSC
PAL	DISPLAY_RESOLUTION_PAL

0x0000	Control

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Reset	31	R/W	0	
Test Pattern Enable	30	R/W	0	Enable a horizontal line test pattern generated by the display controller.
Interlace Enable	29	R/W	0	
Reserved	28:0			

0x0004	Frame0 BaseAddress

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Frame0 BaseAddr	31:0	R/W	Def	Address of first frame

0x0008	Frame Address Offset

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Frame Offset	31:0	R/W	Def	Offset between frames

0x000C	Stride

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Stride	31:0	R/W	Def	VFBC Line offset

0x0010	Horizontal Control

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Horizontal front porch	31:16	R/W	Def	
Horizontal sync width	15:8	R/W	Def	
Horizontal back porch	7:0	R/W	Def	

0x0014	Vertical & Sync Pol Control

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Sync polarity	31:31	R/W	Def	Polarity for both horizontal and vertical sync
Reserved	30:24			
Vertical front porch	23:16	R/W	Def	
Vertical sync width	15:8	R/W	Def	
Vertical back porch	7:0	R/W	Def	

0x0018								Frame Size																							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Line width	31:16	R/W	Def	
Frame height	15:0	R/W	Def	

The default settings for the display controller used in the software projects are:

- PARAMETER C_DEFAULT_STRIDE = 0x00002000

  The VFBC must be on 128 byte boundaries, and 2560 bytes for VGA. We use four bytes per pixel. 640 x 4.

- PARAMETER C_DEFAULT_HSYNC = 0x00106030

  This line sets up `h_front_porch`, `h_sync` and `h_back_porch`.

- PARAMETER C_DEFAULT_VSYNC_POL = 0x000a0221

  This line sets up `sync_polarity`, `v_front_porch`, `v_syn_width`, and `v_back_porch`.

- PARAMETER C_DEFAULT_FRAME_SIZE = 0x028001e0

  This line is the line width and frame height. [640:480]

- PARAMETER C_DEFAULT_FRAME0_BASEADDR = 0x10000000

  This sets the address of where the frame buffer starts.

- PARAMETER C_DEFAULT_FRAME_ADDR_OFFSET = 0x0012c000

  This sets the frame size. 640 x 480 x 4 = 1228800 = 0x12c000

- PARAMETER C_VFBC_DATA_WIDTH = 32, DT = INTEGER, RANGE = (8,32)

  This sets the data width for the VFBC data. The default is 32, but must be changed to 8 if `c_chroma_444` is set to 0.

- PARAMETER C_CHROMA_444 = 1, DT = INTEGER, RANGE = (0,1), VALUES=(0=4:2:2, 1=4:4:4)

  This allows the display controller to work with 4:2:2 data or 4:4:4 data. If it is set to work with 4:2:2 data, then you must also set `c_vfbc_data_width` to 8.

gamma_plbw (v3.00b)

Overview

The gamma_plbw PCORE can be used to apply gamma conversions which are commonly associated with video displays. The gamma_plbw PCORE uses block memory-based look-up tables that can be read or written by the MicroBlaze processor in real-time. Although gamma conversion is the intended application of this block, it could be used to apply any function where a look-up table conversion is suitable.

Software Interface

The gamma_plbw driver allows for the programming of the look-up tables which are used to perform the gamma conversion. There are three independent tables, one for each color component (red, blue, and green).

An example of software using the gamma_plbw PCORE can be found in the C source file `gamma.c` (found in the DVI Pass-Through Demo).

This file implements the software routines that can communicate with the portions of the PCORE that can be read/written by the MicroBlaze processor. The same PCORE is also used in the Camera Frame Buffer Demo.

Due to the fact that this PCORE was developed in System Generator, the source for this driver is not located in the same place as the other drivers. The automatically-generated API documentation for this PCORE can be found in:

```
EDK_Demonstrations\VSK_Repository\VSKProcessorIPLib\pcores\
gamma_plbw_v3_00_b\doc\html\api\index.html
```

video_to_vfbc (v2.00a)

Overview

The video_to_vfbc PCORE manages the storing of video into frame buffers in external memory. It writes the video data to the VFBC interface on the MPMC memory controller. The configuration of video_to_vfbc dictates the way the video appears in external memory.

Software Interface

When a new frame of video is received, the block automatically increments to the next frame buffer to be written to and issues a command to the VFBC to start writing to the new buffer. Frames are stored sequentially in memory, as shown in Figure 3-8. For display purposes, the maximum number of frames stored at any given time is three.

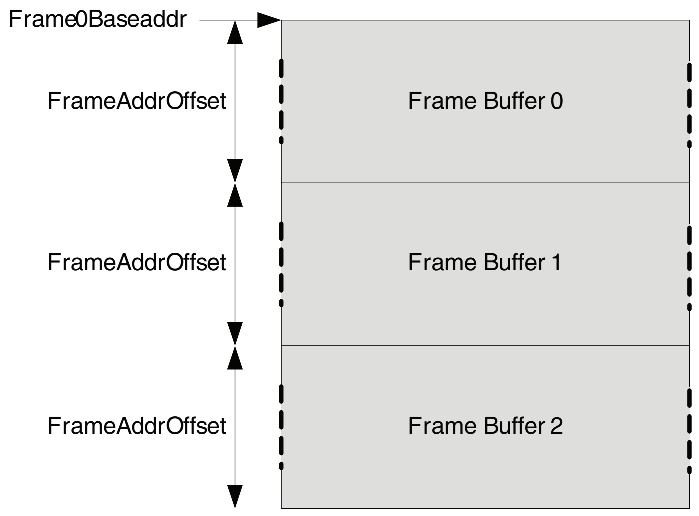


Figure 3-8: **Frame Buffer Memory Map**

To make the best use of memory, the size of each frame buffer is changed to match the video format that is currently in use. The function call `VIDEO_TO_VFBC_setBufferAddresses()` is used to configure these addresses into video_to_vfbc. This and other functions necessary to operate video_to_vfbc are provided in the file video_to_vfbc.c. For examples of how to use video_to_vfbc, see any of the Frame Buffer demos. For more information on the video_to_vfbc.c API, see the documentation in the header file:

```
\Demonstrations\EDK\EDK_Demonstrations\VSK_Repository\
VSKProcessorIPLib\drivers\video_to_vfbc_v2_00_a\src\video_
to_vfbc.h
```

The MPMC denomination should be consulted for information on the data endianness.

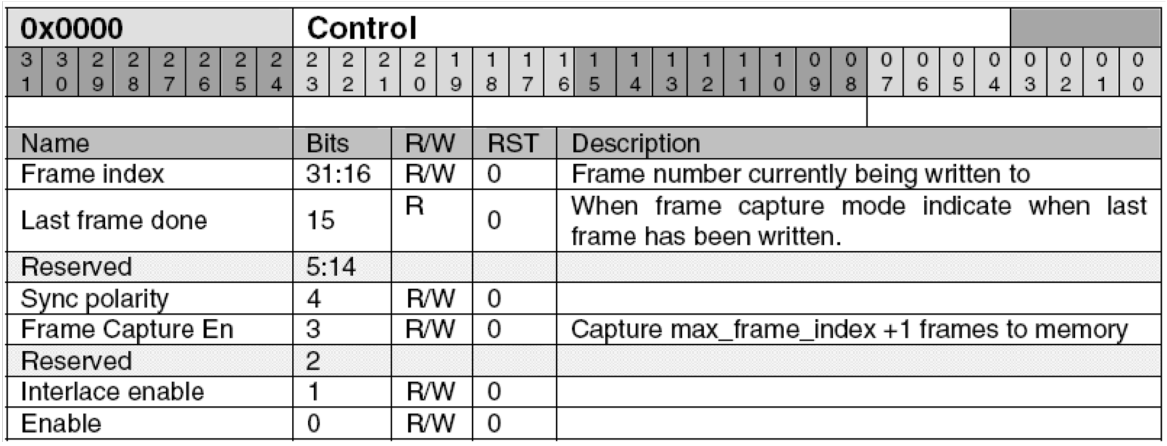
The following is an example of a 32-bit write:

```
#define RED  (0x0000FF00)
#define GREEN  (0x00FF0000)
#define BLUE  (0xFF000000)
```

For convenience, the video standards listed in Table 3-2 are pre-defined in video_to_vfbc.h. These are the only video standards with which video_to_vfbc has been tested.

Table 3-2: **Video Standards Defined in video_to_vfbc.h**

Video Standard	Constant Name
640x480p @ 60 Hz	VIDEO_TO_VFBC_RESOLUTION_VGA
720x480p @ 60 Hz	VIDEO_TO_VFBC_RESOLUTION_480P60
720x576p @ 50 Hz	VIDEO_TO_VFBC_RESOLUTION_576P50
800x600p @ 60 Hz	VIDEO_TO_VFBC_RESOLUTION_SVGA
1024x768p @ 60 Hz	VIDEO_TO_VFBC_RESOLUTION_XGA
1280x720p @ 50 Hz	VIDEO_TO_VFBC_RESOLUTION_720P50
1280x720p @ 60 Hz	VIDEO_TO_VFBC_RESOLUTION_720P60
1920x1080i @ 50 Hz	VIDEO_TO_VFBC_RESOLUTION_1080I50
1920x1080i @ 60 Hz	VIDEO_TO_VFBC_RESOLUTION_1080I60
1920x1080p @ 25 Hz	VIDEO_TO_VFBC_RESOLUTION_1080P25
1920x1080p @ 30 Hz	VIDEO_TO_VFBC_RESOLUTION_1080P30
NTSC	VIDEO_TO_VFBC_RESOLUTION_NTSC
PAL	VIDEO_TO_VFBC_RESOLUTION_PAL



0x0000	Control

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Frame index	31:16	R/W	0	Frame number currently being written to
Last frame done	15	R	0	When frame capture mode indicate when last frame has been written.
Reserved	5:14			
Sync polarity	4	R/W	0	
Frame Capture En	3	R/W	0	Capture max_frame_index +1 frames to memory
Reserved	2			
Interlace enable	1	R/W	0	
Enable	0	R/W	0	

0x0004								X Size																							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
X size	31:0	R/W	Def	Line length

0x0008								Y size																							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Y size	31:0	R/W	Def	Frame height

0x000C								Stride																							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Stride	31:0	R/W	Def	VFBC Line offset

0x0010								Frame0 BaseAddress																							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Frame0 BaseAddr	31:0	R/W	Def	Address of first frame

0x0014								Frame Address Offset																							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Frame Offset	31:0	R/W	Def	Offset between frames

0x0018								Max Frame Index																							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Name	Bits	R/W	RST	Description
Max frame index	31:0	R/W	Def	The last frame index to written to when in capture mode

The default settings for the display controller used in the software projects are:

- PARAMETER C_DEFAULT_MAX_FRAME_INDEX = 2

  This sets the memory buffer size to three frames.

Chapter 4

Software Projects

Base Platform

Overview

Each reference design included with the VSK is built from the same base platform (shown in Figure 4-1). The base platform is not delivered as part of VSK, but is mentioned here for reference purposes. Under all circumstances, software for the VSK is executed by the MicroBlaze™ processor.

The base platform includes from the following Processor IP blocks:

- MicroBlaze 32-bit Soft Microprocessor
- LMB Local Memory Bus
- LMB Block RAM Controller
- Block RAM Block Memory
- PLB Processor Local Bus
- XPS Uartlite
- XPS GPIO
- XPS IIC Controller
- XPS SYSACE Compact Flash Controller
- MPMC External Multi-port Memory Controller
- MDM Debug Module
- Clock Generator
- Processor System Reset

For details on any of the base platform blocks, see the EDK documentation listed in [Ref 1].

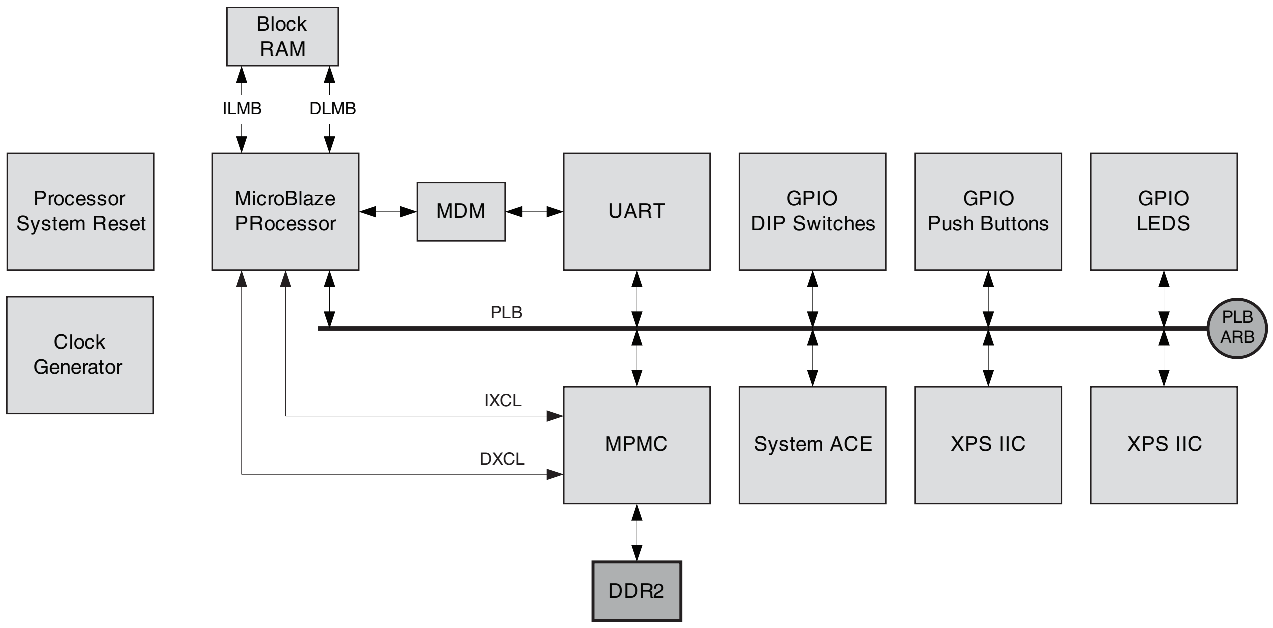


Figure 4-1: **Base Platform Block Diagram**

Frame0Baseaddr for the DVI Frame Buffer and S-Video Frame Buffer Demos is 0x10000000. To support the resolution 1920x1080, the DVI Frame Buffer Demo uses the maximum FrameAddrOffset of any of the demos, which is 0x7E9000 (8,294,400 bytes). The Camera Frame Buffer Demo is different in that the Frame0Baseaddr is 0x10400000 and the number of frame buffers can increase to reach the end of memory (0x1FFFFFFF). This is to support real-time video capture.

The base platform is built around the MicroBlaze processor. Instructions and data can be stored in the local block RAM or in the external DDR2 memory attached to the MPMC memory controller. See Table 4-1.

Table 4-1: **Base Platform Memory Map**

IP Core	Version	Memory Address		Instances	Bus
		Low	High		
MicroBlaze Processor	7.20.d	-	-	microblaze_0	LMB, PLB, XCL
LMB Block RAM IF Controller	2.10.b	0x00000000	0x00000FFFF	dlmb_cntlr, ilmb_cntlr	LMB
MPMC DDR2 Controller	5.04.a	0x10000000	0x1FFFFFFF	mpmc_0	XCL
xps_gpio	2.00.a	0x81400000	0x8140FFFF	DIP_Switches_8Bit	PLB
xps_gpio	2.00.a	0x81420000	0x8142FFFF	LEDs_8Bit	PLB
xps_gpio	2.00.a	0x81440000	0x8144FFFF	Push_Buttons_Position	PLB
xps_iic	2.02.a	0x81600000	0x8160FFFF	xps_iic_0	PLB
xps_iic	2.02.a	0x81620000	0x8162FFFF	xps_iic_1	PLB
xps_sysace	1.01.a	0x83600000	0x8360FFFF	SysACE_CompactFlash	PLB

Table 4-1: **Base Platform Memory Map *(Cont'd)***

IP Core	Version	Memory Address		Instances	Bus
		Low	High		
xps_uartlite	1.01.a	0x84000000	0x8400FFFF	RS232_Uart	PLB
mdm	1.00.g	0x84400000	0x8440FFFF	debug_module	PLB

Note: By default all the demos execute their application software from the external memory space made available by MPMC (0x10000000 - 0x1FFFFFFF). This memory space is shared with video frame buffers in the case of frame buffer demos.

Directory Structure

The files for the demo designs can be found on the supplied CD or on the web as described in Chapter 1, "Getting Started." The following directory structure applies to each of the demos contained in the `EDK_Demonstrations` directory (for example, `\EDK_Demonstrations\DVI_Pass_Through_Demo\...`).

Of particular interest for software developers are the following items in each demo directory:

- `\<Design>_Sw\src` – The application software code specific to the demo.
- `system.xmp` – Top-level project design file. XPS reads this file and graphically displays its contents in the XPS user interface.
- `executable.elf` – The original compiled software executable for the demo.

Common Conventions

All of the demo software share some commonalities in terms of how they are structured. For example, `vsk_top.c` contains the main() function for all the demo software. This is the area from which all the application software begins executing. Another common aspect is that all the demos make use of a serial console for user interaction. These menus are hierarchical, and generally a separate menu is provided by each source file.

DVI Pass-Through Demo

Note: For more information on the DVI Pass-Through design, see the DVI Pass-Through Example Application in UG456.

Overview

The processor in this demo has the following responsibilities:

- Initializing the VSK peripherals.
- Communicating with the PC HyperTerminal.
- Controlling the Video Processing Pipeline by reading and writing control registers in the system.

Table 4-2 shows the DVI-Pass-Through Demo memory map.

Table 4-2: **DVI Pass-Through Demo Memory Map**

IP Core	Version	Memory Address		Instances	Bus
		Low	High		
MicroBlaze Processor	7.20.d	-	-	microblaze_0	LMB, PLB, XCL
LMB Block RAM IF Controller	2.10.b	0x00000000	0x00000FFFF	dlmb_cntlr, ilmb_cntlr	LMB
MPMC DDR2 Controller	5.04.a	0x10000000	0x1FFFFFFF	mpmc_0	XCL
xps_gpio	2.00.a	0x81400000	0x8140FFFF	DIP_Switches_8Bit	PLB
xps_gpio	2.00.a	0x81420000	0x8142FFFF	LEDs_8Bit	PLB
xps_gpio	2.00.a	0x81440000	0x8144FFFF	Push_Buttons_Position	PLB
xps_iic	2.02.a	0x81600000	0x8160FFFF	xps_iic_0	PLB
xps_iic	2.02.a	0x81620000	0x8162FFFF	xps_iic_1	PLB
xps_sysace	1.01.a	0x83600000	0x8360FFFF	SysACE_CompactFlash	PLB
xps_uartlite	1.01.a	0x84000000	0x8400FFFF	RS232_Uart	PLB
mdm	1.00.g	0x84400000	0x8440FFFF	debug_module	PLB
rgb_2d_fir_plbw	3.00.b	0xC0600000	0xC060FFFF	rgb_2d_fir	PLB
de_gen	2.00.a	0xC1000000	0xC100FFFF	de_gen_0	PLB
gamma_plbw	3.00.b	0xC2600000	0xC260FFFF	gamma_in	PLB
gamma_plbw	3.00.b	0xCCE00000	0xCCE0FFFF	gamma_out	PLB

This demo extends the Base Platform by the addition of a video processing pipeline (shown in Figure 4-2) consisting of the following PCOREs:

- dvi_in – Described in "dvi_in (v2.00a)" in Chapter 2.
- dvi_out – Described in "dvi_out (v3.00a)" in Chapter 2.
- de_gen – Described in "de_gen (v2.00a)" in Chapter 3.
- gamma – Described in "gamma_plbw (v3.00b)" in Chapter 3.
- rgb_2d_fir_plbw

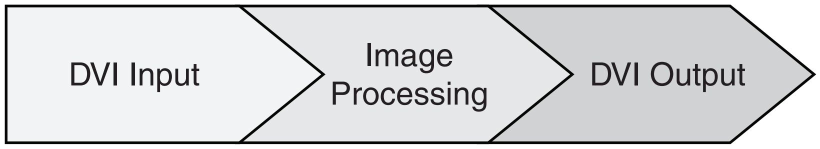


Figure 4-2: **DVI Pass-Through Pipeline**

Since the RGB 2-D FIR filter was designed in System Generator, it has automatically-generated API documentation available at:

```
EDK_Demonstrations\DVI_Pass_Through_Demo\pcores\
rgb_2d_fir_plbw_v3_00_b\doc\html\api\index.html
```

Source Files

The source files for the DVI Pass-Through Demo are organized to match the menu hierarchy as shown in Figure 4-3.

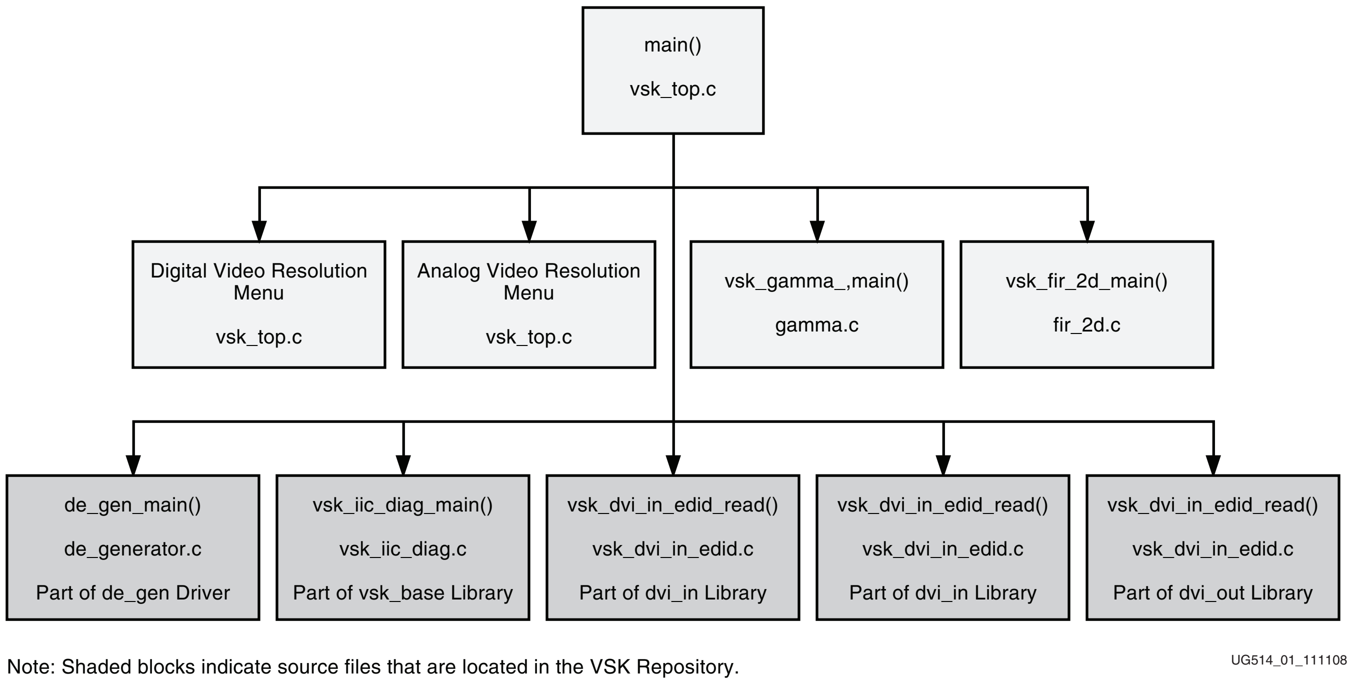


Figure 4-3: **DVI Pass-Through Demo Menu Software Flow Diagram**

All source code files can be found in the DVI Pass-Through project directory at:

```
EDK_Demonstrations\DVI_Pass_Through_Demo\DVI_Pass_Through_Sw\src
```

vsk_top.c

The top-level C source code file for the DVI Pass-Through demo software application is `vsk_top.c`. This file performs the initialization of the VSK at power-up and then creates the menu system that is used in the HyperTerminal.

gamma.c

The software associated with the Gamma Processing PCORE can be found in the C source file `gamma.c`. This file implements the software routines that communicate with the portions of the PCORE that can be read/written by the MicroBlaze processor.

filter_2d.c

The software associated with the filter processing portion of the demo can be found in the C source file `filter_2d.c`. It implements the software routines that read and write the filter coefficients and gain value of the 2-D FIR filter. Using these routines, the coefficient values and the gain can be adjusted in real-time by the MicroBlaze processor.

DVI Frame Buffer Demo

Note: For more information on the DVI Frame Buffer Demo design, see the DVI Frame Buffer Demo Example Application in UG456.

Overview

The system is controlled by a MicroBlaze processor that has the following responsibilities:

- Initializing the VSK peripherals.
- Communicating with the PC HyperTerminal.
- Controlling the Video Frame Buffer Pipeline by reading and writing control registers in the system.

Table 4-3 shows the DVI Frame Buffer memory map.

Table 4-3: **DVI Frame Buffer Memory Map**

IP Core	Version	Memory Address		Instances	Bus
		Low	High		
MicroBlaze Processor	7.20.d	-	-	microblaze_0	LMB, PLB, XCL
LMB BRAM IF Controller	2.10.b	0x00000000	0x00000FFFF	dlmb_cntlr, ilmb_cntlr	LMB
MPMC DDR2 Controller	5.04.a	0x10000000	0x1FFFFFFF	mpmc_0	XCL, PLB
xps_gpio	2.00.a	0x81400000	0x8140FFFF	DIP_Switches_8Bit	PLB
xps_gpio	2.00.a	0x81420000	0x8142FFFF	LEDs_8Bit	PLB
xps_gpio	2.00.a	0x81440000	0x8144FFFF	Push_Buttons_Position	PLB
xps_iic	2.02.a	0x81600000	0x8160FFFF	xps_iic_0	PLB
xps_iic	2.02.a	0x81620000	0x8162FFFF	xps_iic_1	PLB
xps_sysace	1.01.a	0x83600000	0x8360FFFF	SysACE_CompactFlash	PLB
xps_uartlite	1.01.a	0x84000000	0x8400FFFF	RS232_Uart	PLB
mdm	1.00.g	0x84400000	0x8440FFFF	debug_module	PLB
MPMC Port Monitor	5.04.a	0xB0000000	0xB000FFFF	mpmc_0	PLB
video_to_vfbc	2.00.a	0xC0E00000	0xC0E0FFFF	video_to_vfbc_0	PLB
de_gen	2.00.a	0xC1000000	0xC100FFFF	de_gen_0	PLB
display_controller	3.00.a	0xCB400000	0xCB40FFFF	display_controller_0	PLB

This demo extends the Base Platform by the addition of a frame buffer pipeline (shown in Figure 4-4) consisting of the following VSK PCOREs:

- dvi_in – Described in "dvi_in (v2.00a)" in Chapter 2.
- dvi_out – Described in "dvi_out (v3.00a)" in Chapter 2.
- de_gen – Described in "de_gen (v2.00a)" in Chapter 3.
- video_to_vfbc – Described in "video_to_vfbc (v2.00a)" in Chapter 3.
- display_controller – Described in "display_controller (v3.00a)" in Chapter 3.

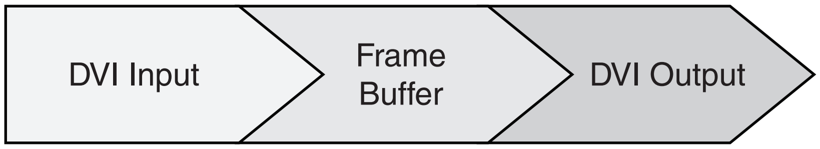


Figure 4-4: **DVI Frame Buffer Pipeline**

Source Files

The source files for the DVI Frame Buffer Demo are organized to match the menu hierarchy as shown in Figure 4-5.

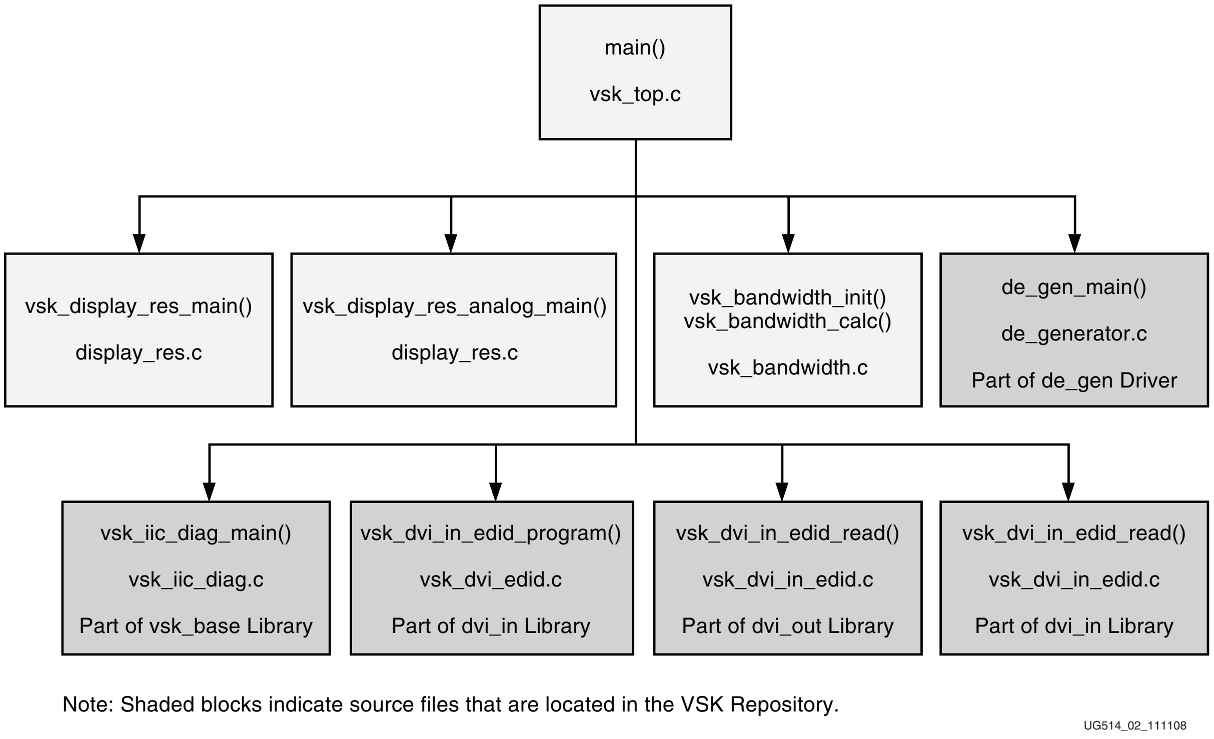


Figure 4-5: **DVI Frame Buffer Demo Menu Software Flow Diagram**

All source code files can be found in the DVI Frame Buffer project directory structure at:

```
EDK_Demonstrations\DVI_Frame_Buffer_Demo\DVI_Frame_Buffer_Sw\src
```

vsk_top.c

The top-level C source code file for the DVI Frame Buffer demo software application is `vsk_top.c`. This file performs the initialization of the VSK at power-up and then creates the menu system that is used in the HyperTerminal.

display_res.c

The software found in this file implements the necessary calls to the programmable clock, display_controller, and video_to_vfbc to enable switching between the various supported video standards.

vsk_bandwidth.c

The software associated with reporting memory bandwidth utilization data can be found in the C source file `vsk_bandwidth.c`. It implements the software routines that gather and display memory bandwidth for each port on the MPMC.

For more information about this feature, see [Ref 1].

Camera Frame Buffer Demo

For more information on the Camera Frame Buffer Demo design, see the Camera Frame Buffer Demo Example Application in UG456.

Overview

The system is controlled by a MicroBlaze processor that has the following responsibilities:

- Initializing the VSK peripherals.
- Communicating with the PC HyperTerminal.
- Controlling the Video Processing and Frame Buffer Pipelines by reading and writing control registers in the system.
- Capturing video, images, and settings to CompactFlash.

Table 4-4 shows the Camera Frame Buffer memory map.

Table 4-4: **Camera Frame Buffer Memory Map**

IP Core	Version	Memory Address		Instances	Bus
		Low	High		
MicroBlaze Processor	7.20.d	-	-	microblaze_0	LMB, PLB, XCL
LMB Block RAM IF Controller	2.10.b	0x00000000	0x00000FFFF	dlmb_cntlr, ilmb_cntlr	LMB
MPMC DDR2 Controller	5.04.a	0x10000000	0x1FFFFFFF	mpmc_0	XCL
xps_gpio	2.00.a	0x81400000	0x8140FFFF	DIP_Switches_8Bit	PLB
xps_gpio	2.00.a	0x81420000	0x8142FFFF	LEDs_8Bit	PLB
xps_gpio	2.00.a	0x81440000	0x8144FFFF	Push_Buttons_Position	PLB
xps_iic	2.02.a	0x81600000	0x8160FFFF	xps_iic_0	PLB
xps_iic	2.02.a	0x81620000	0x8162FFFF	xps_iic_1	PLB
xps_sysace	1.01.a	0x83600000	0x8360FFFF	SysACE_CompactFlash	PLB
xps_uartlite	1.01.a	0x84000000	0x8400FFFF	RS232_Uart	PLB
mdm	1.00.g	0x84400000	0x8440FFFF	debug_module	PLB
video_to_vfbc	2.00.a	0xC0E00000	0xC0E0FFFF	video_to_vfbc_0	PLB
vsk_camera_vop_plbw	3.00.b	0xC9E00000	0xC9E0FFFF	vsk_camera_vop_plbw_0	PLB
display_controller	3.00.a	0xCB400000	0xCB40FFFF	display_controller_0	PLB
gamma_plbw	3.00.b	0xCC600000	0xCC60FFFF	gamma_plbw_0	PLB

This demo extends the DVI Frame Buffer Demo with the following modifications (see Figure 4-6):

- camera (instead of dvi_in) – Described in "camera (v2.00a)" in Chapter 2.
- Addition of gamma – Described in "gamma_plbw (v3.00b)" in Chapter 3.
- Addition of Camera Processing (vsk_camera_vop_plbw).

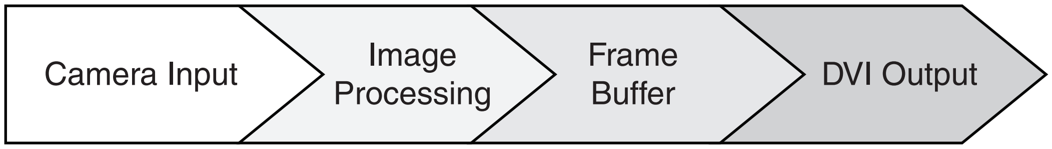


Figure 4-6: **Camera Frame Buffer Pipeline**

Since the Camera Processing was designed in System Generator, it has automatically-generated API documentation available at:

```
EDK_Demonstrations\Camera_Frame_Buffer_Demo\pcores\
vsk_camera_vop_plbw_v3_00_b\doc\html\api\index.html
```

Source Files

The source files for the Camera Frame Buffer Demo are organized to match the menu hierarchy as shown in Figure 4-7.

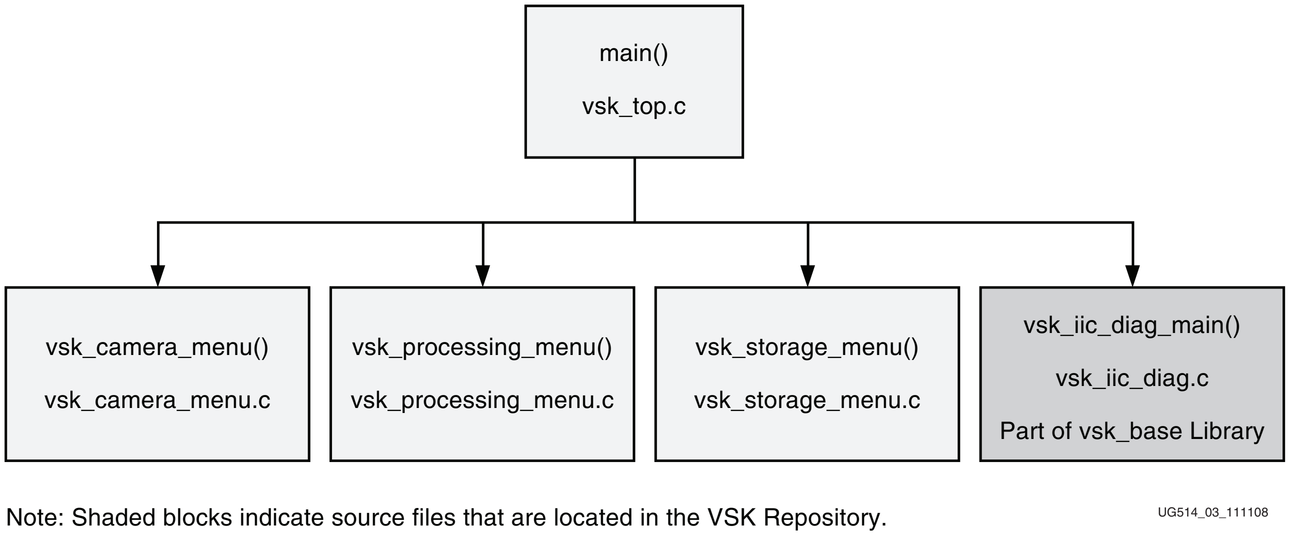


Figure 4-7: **Camera Frame Buffer Demo Menu Software Flow Diagram**

All source code files can be found in the Camera Frame Buffer project directory structure at:

```
EDK_Demonstrations\Camera_Frame_Buffer_Demo\Camera_Frame_Buffer_Sw\src
```

vsk_top.c

The top-level C source code file for the Camera Frame Buffer demo software application is `vsk_top.c`. This file performs the initialization of the VSK at power-up and then creates the menu system that is used in the HyperTerminal.

vsk_camera_menu.c

This file provides a menu of commands to configure the camera into various modes of operation. These commands make use of functions provided by the camera library described in "camera (v2.00a)" in Chapter 2, as well as some functions provided by `vsk_processing_menu.c`.

vsk_processing_menu.c

This file provides a menu of commands for configuring the Camera Processing (vsk_camera_vop_plbw) as well as the gamma PCORE. These commands make use of the Camera Processing drivers (described in "Overview," page 34) as well as the gamma drivers described in "gamma_plbw (v3.00b)" in Chapter 3.

vsk_storage_menu.c

This file provides the storage capabilities of the demo that access CompactFlash. The code that converts files from the format stored in memory to BMP format is also provided here. Still and video capture modes are included in this file, as well as reading/writing of Camera Processing settings.

The commands in this file make use of the EDK xilfatfs library, the video_to_vfbc and display_controller drivers, as well as functions provided by `vsk_processing_menu.c`. xilfatfs documentation is provided as part of the EDK installation and can be found at:

```
$XILINX\EDK\sw\lib\sw_services\xilfatfs_v1_00_a\doc\
xilfatfs_v1_00_a.pdf
```

See Chapter 3, "Drivers" for more information on video_to_vfbc and display_controller.

S-Video Frame Buffer Demo

Note: For more information on the S-Video Frame Buffer Demo design, see the S-Video Frame Buffer Demo Example Application in UG456.

Overview

The processor in this demo has the following responsibilities:

- Initializing the VSK peripherals.
- Communicating with the PC HyperTerminal.
- Controlling the Video Processing Pipeline by reading and writing control registers in the system.

Table 4-5 shows the S-Video Frame Buffer memory map.

Table 4-5: **S-Video Frame Buffer Memory Map**

IP Core	Version	Memory Address		Instances	Bus
		Low	High		
MicroBlaze Processor	7.20.d	-	-	microblaze_0	LMB, PLB, XCL
LMB BRAM IF Controller	2.10.b	0x00000000	0x00000FFFF	dlmb_cntlr, ilmb_cntlr	LMB
MPMC DDR2 Controller	5.04.a	0x10000000	0x1FFFFFFF	mpmc_0	XCL

Table 4-5: **S-Video Frame Buffer Memory Map *(Cont'd)***

IP Core	Version	Memory Address		Instances	Bus
		Low	High		
xps_gpio	2.00.a	0x81400000	0x8140FFFF	DIP_Switches_8Bit	PLB
xps_gpio	2.00.a	0x81420000	0x8142FFFF	LEDs_8Bit	PLB
xps_gpio	2.00.a	0x81440000	0x8144FFFF	Push_Buttons_Position	PLB
xps_iic	2.02.a	0x81600000	0x8160FFFF	xps_iic_0	PLB
xps_iic	2.02.a	0x81620000	0x8162FFFF	xps_iic_1	PLB
xps_sysace	1.01.a	0x83600000	0x8360FFFF	SysACE_CompactFlash	PLB
xps_uartlite	1.01.a	0x84000000	0x8400FFFF	RS232_Uart	PLB
mdm	1.00.g	0x84400000	0x8440FFFF	debug_module	PLB
video_to_vfbc	2.00.a	0xC0E00000	0xC0E0FFFF	video_to_vfbc_0	PLB
de_gen	2.00.a	0xC1000000	0xC100FFFF	de_gen_0	PLB
sdtv_gamma_plbw	3.00.b	0xC2400000	0xC240FFFF	sdtv_gamma_plbw_0	PLB
ycbcr_2d_fir_plbw	3.00.b	0xC3E00000	0xC3E0FFFF	ycbcr_2d_fir_plbw_0	PLB
display_controller	3.00.a	0xCB400000	0xCB40FFFF	display_controller_0	PLB

This demo extends the Base Platform by the addition of a video processing pipeline (shown in Figure 4-8) consisting of the following PCOREs:

- sdtv_in – Described in "sdtv_in (v2.00a)" in Chapter 2.
- sdtv_out – Described in "sdtv_out (v3.00a)" in Chapter 2.
- de_gen – Described in "de_gen (v2.00a)" in Chapter 3.
- video_to_vfbc – Described in "video_to_vfbc (v2.00a)" in Chapter 3.
- display_controller – Described in "display_controller (v3.00a)" in Chapter 3.
- sdtv_gamma_plbw
- ycbcr_2d_fir_plbw

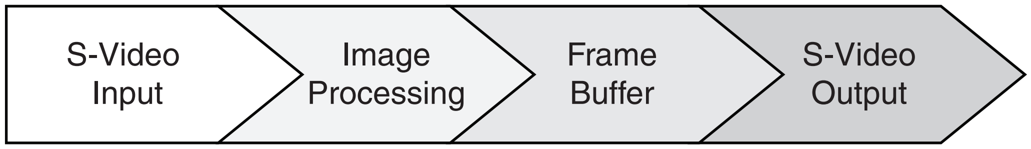


Figure 4-8: **S-Video Frame Buffer Pipeline**

Since the SDTV Gamma and YCbCr 2-D FIR filter were designed in System Generator, they have automatically-generated API documentation available at:

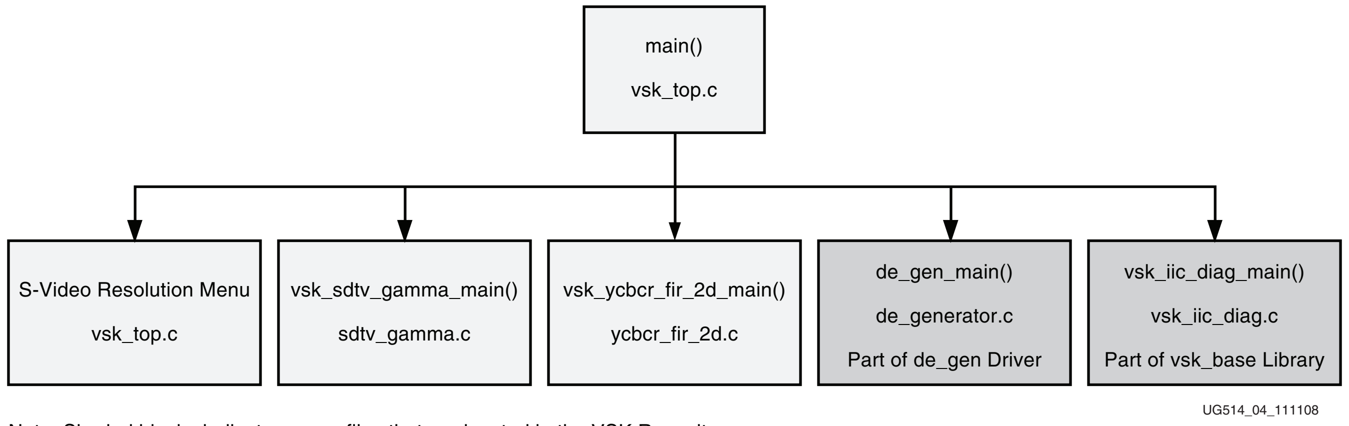
```
EDK_Demonstrations\S-Video_Frame_Buffer_Demo\pcores\
sdtv_gamma_plbw_v3_00_b\doc\html\api\index.html
```

and

```
EDK_Demonstrations\S-Video_Frame_Buffer_Demo\pcores\
ycbcr_2d_fir_plbw_v3_00_b\doc\html\api\index.html
```

Source Files

The source files for the S-Video Frame Buffer Demo are organized to match the menu hierarchy as shown in Figure 4-9.



Note: Shaded blocks indicate source files that are located in the VSK Repository.

Figure 4-9: **S-Video Frame Buffer Demo Menu Software Flow Diagram**

All source code files can be found in the S-Video Frame Buffer project directory structure at:

```
EDK_Demonstrations\S-Video_Frame_Buffer_Demo\
S-Video_Frame_Buffer_Sw\src
```

vsk_top.c

The top-level C source code file for the S-Video Frame Buffer demo software application is `vsk_top.c`. This file performs the initialization of the VSK at power-up and then creates the menu system that is used in the HyperTerminal.

sdtv_gamma.c

The software associated with the SDTV Gamma Processing PCORE can be found in the C source file `gamma.c`. This file implements the software routines that communicate with the portions of the PCORE that can be read/written by the MicroBlaze processor.

ycbcr_filter_2d.c

The software associated with the filter processing portion of the demo can be found in the C source file `ycbcr_filter_2d.c`. It implements the software routines that read and write the filter coefficients and gain value of the YCbCr 2-D FIR filter. Using these routines, the coefficient values and the gain can be adjusted in real-time by the MicroBlaze processor.

Chapter 5

Software Development Flow

EDK must be installed to develop software using VSK. For details about the hardware and software requirements for VSK development, see "Development Environment" in UG456, *Spartan-3A DSP FPGA Video Starter Kit User Guide*. It is also necessary to have a basic understanding of the C programming language.

Note: This section explains how to use the Xilinx Platform Studio for quick software changes. Xilinx also offers a full Software Development Kit (SDK) that should be used for more in-depth software development, which is beyond the scope of this documentation. For information on this resource, see the Embedded Development Kit Documentation.

Software Compilation

1. Start EDK by selecting **Start** →**Programs** →**Xilinx ISE Design Suite** →**EDK** →**Xilinx Platform Studio**.
2. From the **Create new or open existing project** dialog shown in Figure 5-1, choose **Open a recent project**.
   a. If this is the first time you are opening one of the VSK projects:
      - Choose **Browse for More Projects...** from the pop-up
      - Browse to your EDK_Demonstrations directory and open one of the VSK Demos by opening `system.xmp` (for example, `EDK_Demonstrations\Camera_Frame_Buffer_Demo\system.xmp`)

   b. If you have opened a VSK project previously, choose the project from the pop-up menu.
3. Choose **Software** →**Build All User Applications**.

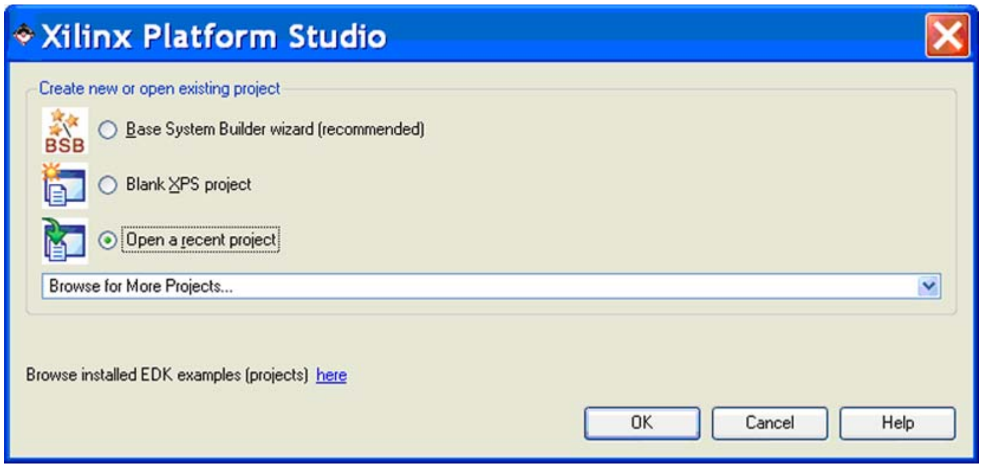
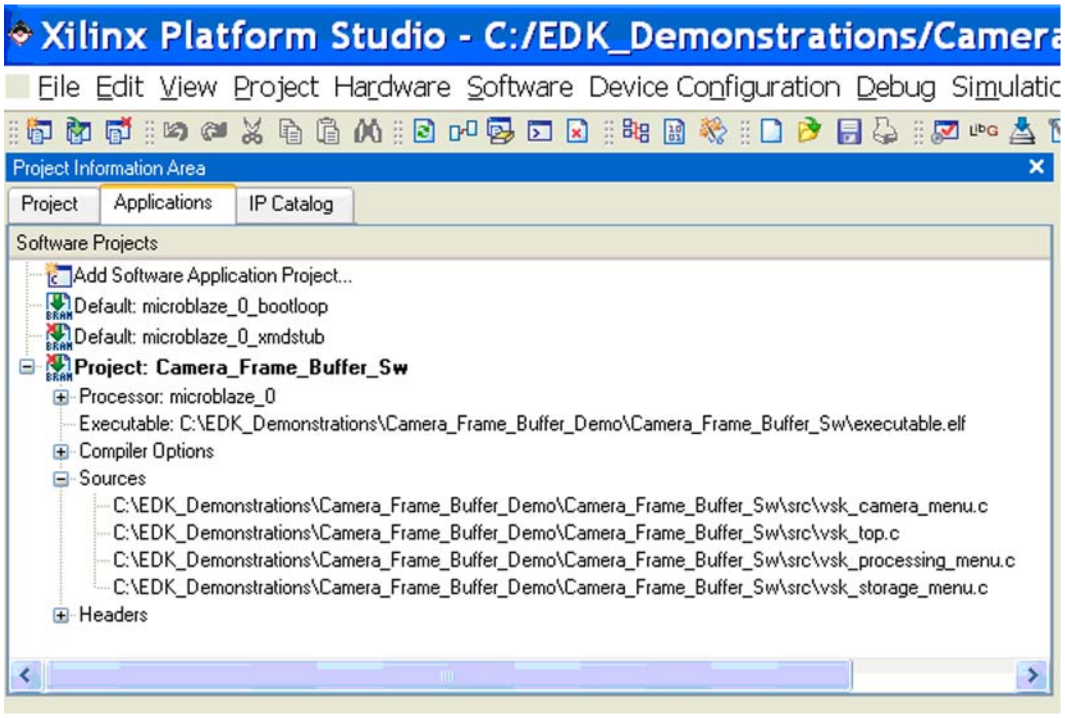

Xilinx Platform Studio
Create new or open existing project
BSB
Base System Builder wizard (recommended)
Blank XPS project
Open a recent project
Browse for More Projects...
Browse installed EDK examples (projects) here
OK
Cancel
Help

Figure 5-1: **Create New or Open Existing Project**

Viewing Code

To view the code, first verify that all the steps in "Software Compilation" were completed. Then, proceed through the following steps:

1. In the upper left corner of your screen, locate the **Project Information Area**. Click the **Applications** tab to bring forward the **Software Projects** pane.
2. In the software projects pane, locate **Project: <*project name*>_Sw** (for example, **Project: Camera_Frame_Buffer_Sw** in the case of the Camera Frame Buffer Demo).
3. Ensure the Project tree view is expanded by clicking the "+" sign next to it.
4. Under the Project tree, expand the Sources tree by clicking the "+" sign next to it. A list of the source files in the project is displayed, as shown in Figure 5-2.
5. To view one of the source files, double-click a file in the Sources list.
6. Because the software is too large to fit into block RAM, right click on the project and make sure that "Mark to Initialize BRAMS" is not checked.


Xilinx Platform Studio - C:/EDK_Demonstrations/Camera
File Edit View Project Hardware Software Device Configuration Debug Simulatic
Project Information Area
Project
Applications
IP Catalog

```
Software Projects
  Add Software Application Project...
  Default: microblaze_0_bootloop
  Default: microblaze_0_xmdstub
  Project: Camera_Frame_Buffer_Sw
    Processor: microblaze_0
    Executable: C:\EDK_Demonstrations\Camera_Frame_Buffer_Demo\Camera_Frame_Buffer_Sw\executable.elf
    Compiler Options
    Sources
      C:\EDK_Demonstrations\Camera_Frame_Buffer_Demo\Camera_Frame_Buffer_Sw\src\vsk_camera_menu.c
      C:\EDK_Demonstrations\Camera_Frame_Buffer_Demo\Camera_Frame_Buffer_Sw\src\vsk_top.c
      C:\EDK_Demonstrations\Camera_Frame_Buffer_Demo\Camera_Frame_Buffer_Sw\src\vsk_processing_menu.c
      C:\EDK_Demonstrations\Camera_Frame_Buffer_Demo\Camera_Frame_Buffer_Sw\src\vsk_storage_menu.c
    Headers
```

Figure 5-2: **Applications Pane with Project and Sources Expanded**

Downloading

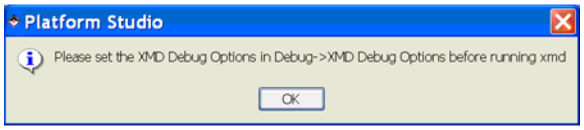
Verify that all the steps in "Software Compilation" were completed. In addition, ensure that a JTAG cable connected is connected to your computer and installed correctly.

1. Power up the VSK.
2. From the menu, choose a demo to modify the software (see [Ref 1]).

   See Appendix C of UG456 for information on downloading your design to hardware.

Note: If you are also making hardware changes, you will need to recompile and download your hardware before proceeding to step 3.

3. In EDK, select **Debug** → **Launch XMD...**
4. If this is the first time XMD has been run, the dialog shown in Figure 5-3 appears.



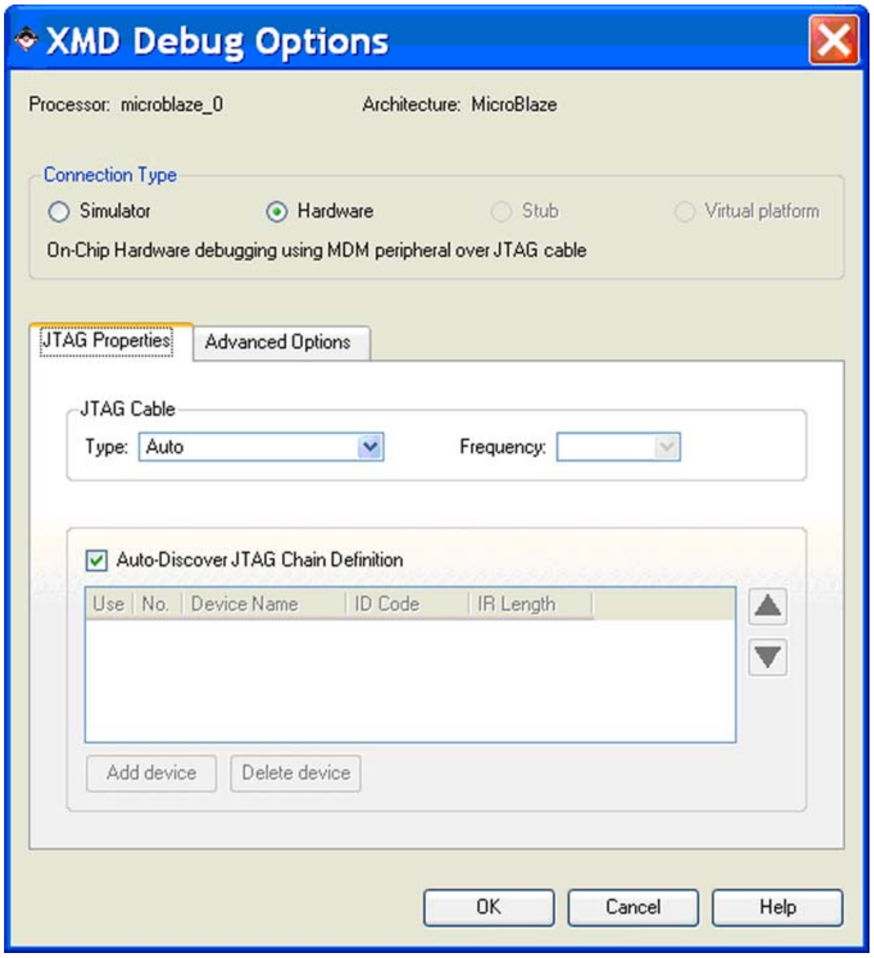
Platform Studio

Please set the XMD Debug Options in Debug->XMD Debug Options before running xmd

OK

Figure 5-3: **Reminder to Set XMD Options**

5. Click **OK**.
6. The next dialog shows the configuration for XMD. Verify that the configuration matches Figure 5-4 and click **OK**.



XMD Debug Options

Processor: microblaze_0 Architecture: MicroBlaze

Connection Type

Simulator [✓ Hardware] Stub Virtual platform

On-Chip Hardware debugging using MDM peripheral over JTAG cable

JTAG Properties | Advanced Options

JTAG Cable

Type: Auto Frequency:

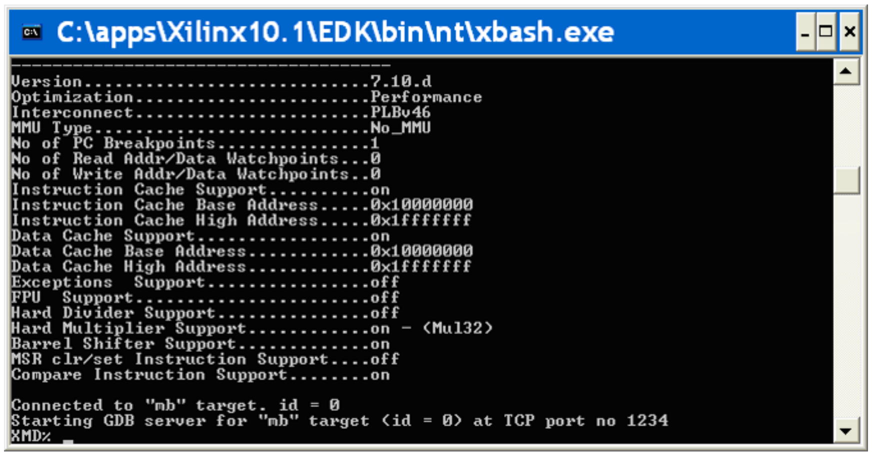
[✓] Auto-Discover JTAG Chain Definition

Use	No.	Device Name	ID Code	IR Length

Figure 5-4: **XMD Configuration**

7. After some time, the XMD prompt appears as shown in Figure 5-5. If XMD connected to the MicroBlaze™ processor successfully, the feedback shows:

   ```
   Connected to "mb" target. id = 0
   ```



```
C:\apps\Xilinx10.1\EDK\bin\nt\xbash.exe
-------------------------------------
Version............................7.10.d
Optimization.......................Performance
Interconnect.......................PLBv46
MMU Type...........................No_MMU
No of PC Breakpoints...............1
No of Read Addr/Data Watchpoints...0
No of Write Addr/Data Watchpoints..0
Instruction Cache Support..........on
Instruction Cache Base Address.....0x10000000
Instruction Cache High Address.....0x1fffffff
Data Cache Support.................on
Data Cache Base Address............0x10000000
Data Cache High Address............0x1fffffff
Exceptions  Support................off
FPU  Support.......................off
Hard Divider Support...............off
Hard Multiplier Support............on - (Mul32)
Barrel Shifter Support.............on
MSR clr/set Instruction Support....off
Compare Instruction Support........on

Connected to "mb" target. id = 0
Starting GDB server for "mb" target (id = 0) at TCP port no 1234
XMD% _
```

Figure 5-5: **XMD Successfully Started**

8. To load the locally-compiled executable, type the following commands at the XMD prompt:

   ```
   XMD% stop
   XMD% dow Camera_Frame_Buffer_Sw/executable.elf
   XMD% run
   ```

   Note: If you are working with one of the other demos, simply replace “Camera_Frame_Buffer” in the preceding command with the name of the demo you are using. If you wish to run the original version of software that came with the kit, omit the text before the “/” (so the command becomes “dow executable.elf”). This is the same for every demo.

- If everything worked properly, there will be some output in HyperTerminal and/or on the screen.
- If there were any problems, refer to the XMD documentation supplied with EDK.

Debug

The debugger is accessible in EDK from the menu option **Debug** → **Launch Software Debugger...** Details on how to use this feature can be found in the EDK documentation, as listed in “References.”