XILINX®

XAPP1136 (v1.0) June 1, 2009

Integrating a Video Frame Buffer Controller (VFBC) in System Generator

Author: Douang Phanthavong and Jingzhao Ou

Summary

This application note provides the basic knowledge on how to integrate an embedded processor system with the Xilinx Multi-Port Memory Controller (MPMC) and Video Frame Buffer Controller (VFBC) IP cores in System Generator for DSP ("System Generator"). Users can then develop custom logic inside of System Generator, attach it to the imported processor system, and as shown in this example build a powerful validation platform through System Generator's hardware co-simulation capability.

Note: This application note assumes the reader has a basic understanding of the Video Starter Kit (VSK) and has used System Generator and Platform Studio before. If the reader is not familiar with these Xilinx tools but would like to go through the design example provided with this application note, follow the step-by-step instructions in the appendices.

Introduction

This application note emphasizes the tool and design flows rather than the technology details of the IP cores. The intent is to show how to connect the MPMC and the VFBC inside of System Generator using the Xilinx ML506 hardware platform. This application note could be used with other boards by applying the same design flows and methodologies. An example video application is provided to illustrate how to exercise and validate the VFBC using existing System Generator and Platform Studio integration. It is a simplified version of the reference design that is currently available in the Xilinx Video Starter Kit.

Integrating an off-the-shelf DDR (double-data-rate) or DDR2 SDRAM (synchronous dynamic random access memory) memory controller into a system is a time-consuming and significant task. Verifying that the design runs correctly is also not trivial. The complexity of the problem is further complicated when one considers that the memory subsystem is often just a small component of the complete design.

The MPMC is a parameterizable memory controller that supports SDRAM/DDR/DDR2 memory access. It provides access to various types of external memory by using o1ne to eight ports. Each port on the MPMC can be configured from a set of Personality Interface Modules (PIMs).

The VFBC is one of the PIMs provided by the MPMC. The uniqueness of the VFBC is that user IP or other interface circuits can read and write data in 2-dimensional sets regardless of the size and organization of the external memory. This enables users to access the external memory without having to know all of the details about the complicated external memory access protocols. Combined with the MPMC, the VFBC is an ideal solution for applications such as motion estimation, video scaling, on-screen displays, and video capture used in video surveillance, video conferencing and video broadcast.

With the MPMC and the VFBC being commonly used IP cores, it is crucial to provide seamless, transparent and easy-to-use tool flows when designing a complex system using these IP cores. Fortunately, the well-integrated tool flows between System Generator and Platform Studio enable designers to quickly put together a system in a matter of hours or days instead of weeks or months.

Review of Existing Tool Flow Capabilities Enabling the Use of the VFBC in System Generator

This section includes the following:

- "Integrated Tool Flow Based on System Generator"
- "System Generator Memory Map Interface"
- "Hardware Co-simulation"
- "System Generator Dual Clock Support for EDK Processors"

Integrated Tool Flow Based on System Generator

System Generator is the centerpiece of the integrated Xilinx DSP design tool flow shown in Figure 1. There are two flows between System Generator and Platform Studio: the EDK Pcore generation (export) flow and the HDL netlisting (import) flow. Both flows are realized through the System Generator EDK Processor block. Users can conveniently select a flow from a pull-down menu in the block GUI.

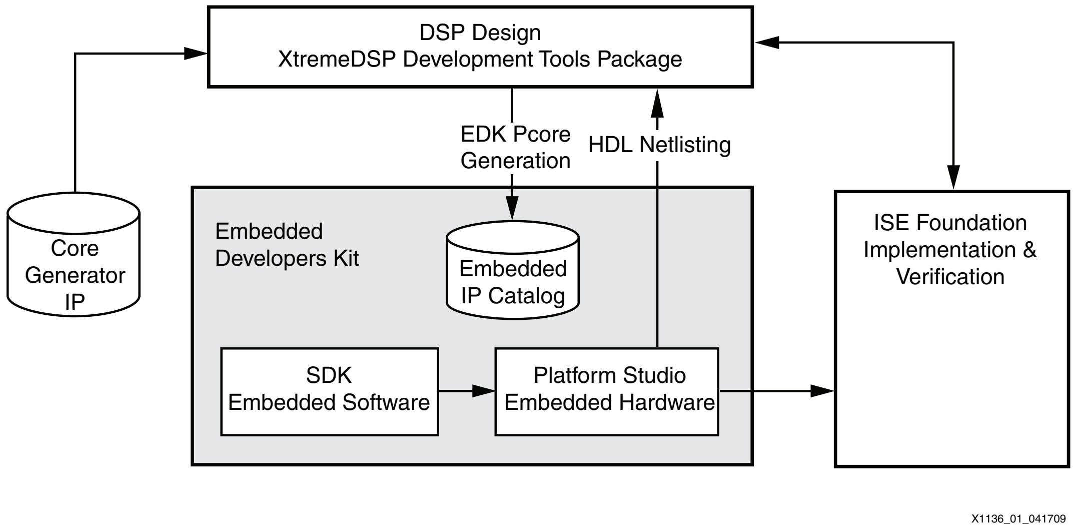


Figure 1: **Integrated DSP and Embedded Tool Flow**

- In the EDK Pcore Generation flow, a System Generator model is netlisted into a Platform Studio Pcore. Users can then add one or multiple copies of the System Generator Pcore as peripherals of the processor system in Platform Studio.
- In the HDL Netlisting flow, a MicroBlaze™ processor system described using Platform Studio is brought into a System Generator model. The low-level netlisting files of the System Generator model would contain the imported processor systems. Most importantly, users can run the imported processor system in hardware through hardware co-simulation for purposes such as accelerated simulation, validation and debug capability.

For both the EDK Pcore Generation flow and HDL Netlisting, System Generator provides two ways for the processor system to communicate with the System Generator model. One way is through an automatically-generated shared-memory based memory-map interface, which is described in "System Generator Memory Map Interface," page 3. The other way is through direct port manipulation. In the EDK Pcore Generation flow, a System Generator Gateway In/Out block is shown as a port on the generated Pcore. In the HDL Netlisting flow, which is shown in Figure 6, the top-level ports of the imported project can be exposed as ports on the EDK Processor block. Users can then connect these exposed top-level ports directly with other portions of the System Generator model. It is through the exposed top-level ports that a System Generator model interacts with the VFBC IP core.

In addition to the integration with Platform Studio, System Generator also has a tight integration with Project Navigator. Users can add multiple copies of the same or different System Generator model as sub-modules of an overall design in Project Navigator. This is a push-button flow, meaning that a System Generator model can be directly added into a Project Navigator project instead of having to manually add the low-level netlist files.

System Generator Memory Map Interface

System Generator provides a simple abstraction for easily adding custom logic into a processor. The basic idea is to allow memories in custom logic to be easily mapped into the processor memory address space. System Generator enables this through the use of Shared-Memory blocks provided in the System Generator block set.

As shown in Figure 2, after users add shared memory blocks to the processor memory map, System Generator automatically generates a bus adapter and a memory map interface for the processor and the user logic to communicate with each other. The three kinds of System Generator shared memories (that is, the From/To Register blocks, the From/To FIFO blocks, and the dual-port BRAM-based Shared Memory block) are all supported by the automatically-generated memory map interface. Corresponding software drivers and driver API documentation are also generated to access the user logic via software running on MicroBlaze.

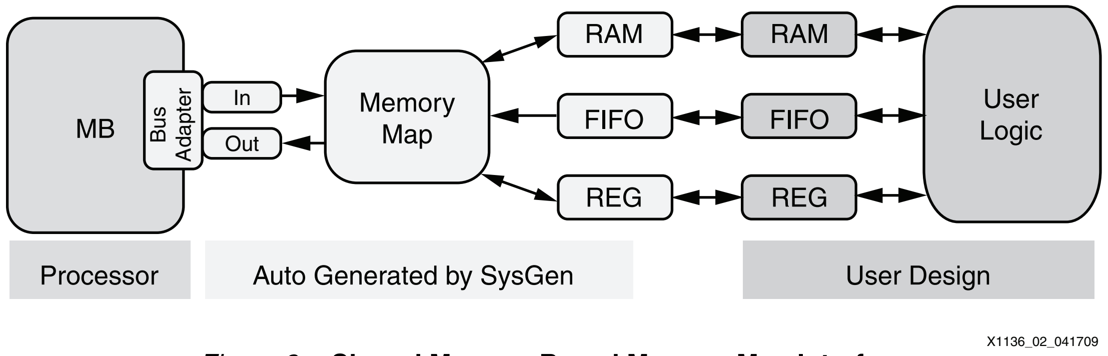


Figure 2: **Shared Memory Based Memory Map Interface**

Hardware Co-simulation

Using hardware co-simulation, the user can select a subsystem in a System Generator model to run in hardware while the rest of the model is simulated on a host PC. For the selected subsystem, System Generator generates a low-level implementation, attaches a hardware co-simulation module to it, and then generates an actual hardware implementation (that is, bitstream) to run on the hardware. During simulation, the hardware co-simulation module controls the clock to the design portion running in hardware and synchronizes and exchanges simulation data with the design portion simulated on the host PC. In free-running mode, the design portion running in hardware runs asynchronously with that running on the host PC. In single-step mode, both design portions run in lock-step mode.

JTAG and Ethernet point-to-point communication are the two most widely used hardware co-simulation interfaces. The JTAG hardware co-simulation is available in a wider range of hardware platforms, while the Ethernet point-to-point hardware co-simulation offers significant speed-ups in most design scenarios. This application note uses Ethernet point-to-point hardware considering the large amount of data transfer required by typical video applications.

Either the Simulink®-based or the MATLAB®-based hardware co-simulation techniques offered by System Generator can be used. The Simulink-based approach offers nice visualization and interaction with its rich modeling block set. On the other hand, using the MATLAB-based hardware co-simulation, the hardware created through the System Generator hardware co-simulation flow can be controlled through a MATLAB software scripting interface. The MATLAB-based approach offers significant simulation speed-ups over the Simulink-based approach. This is mainly accomplished by removing the Simulink modeling overheads.

System Generator Dual Clock Support for EDK Processors

System Generator supports dual clock wiring (Figure 3), which means that the imported processor system and the other portion of a System Generator model are driven by two independent clocks. One major benefit with dual clock wiring is that the MicroBlaze processor system and the System Generator user logic can run at different clock frequencies. For example, MicroBlaze can comfortably operate at 100 MHz, while a DSP FIR (finite impulse response) filter in System Generator can run at 400 MHz.

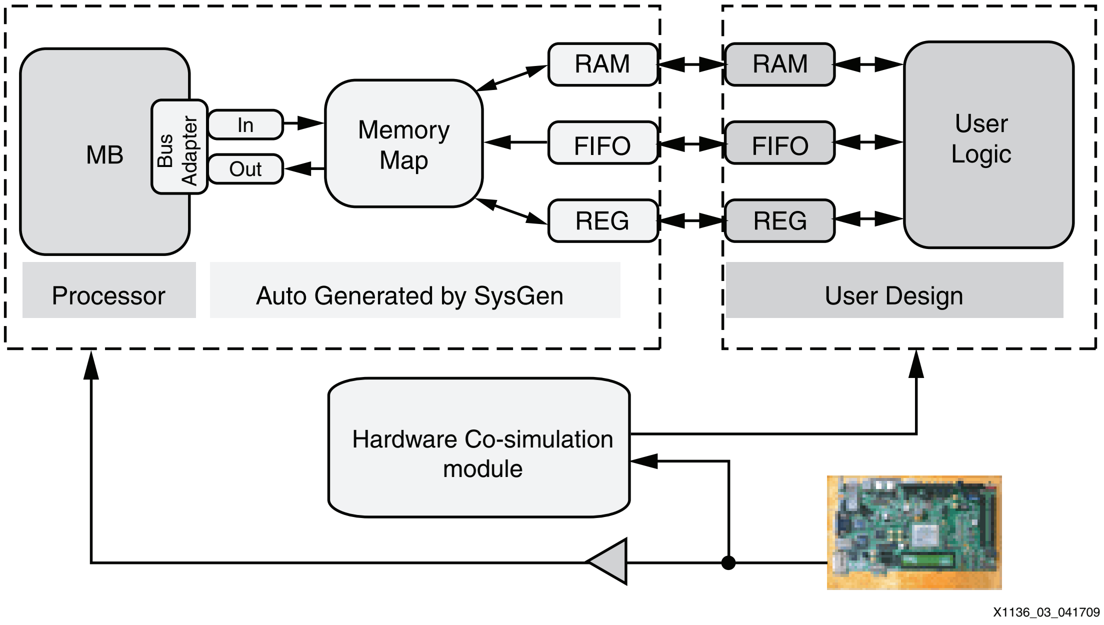


Figure 3: **Hardware Co-simulation with Dual Clock Wiring**

Integrating a VFBC in an Example Video Validation Platform in System Generator

This section includes the following:

- “Creating a Basic Processor System”
- “Configuring the MPMC and the VFBC”
- “Configuring the Input Clock”
- “Importing the Basic Processor System into System Generator”
- “Building the System Generator Validation Model”
- “Generating a Hardware Co-simulation Block”
- “Writing Software to Control VFBCs”
- “Performing Validation through Hardware Co-simulation”

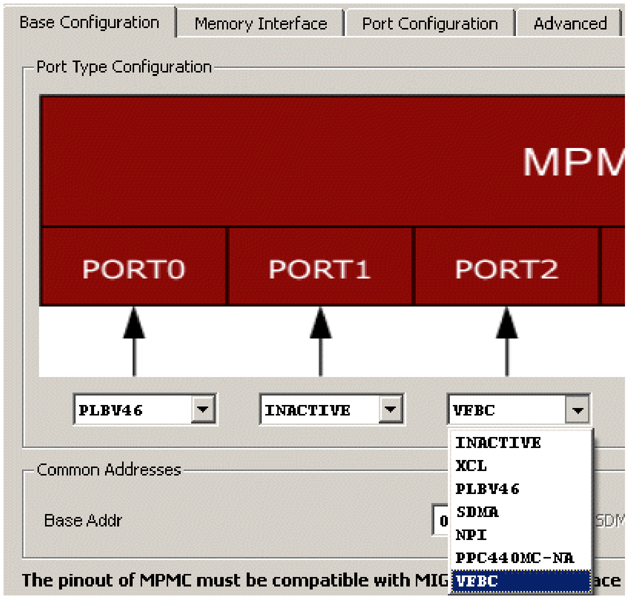
Creating a Basic Processor System

A basic processor system can be created using the Platform Studio Base System Builder (BSB) Wizard with these two peripherals: MPMC/DDR2 and RS232_Uart. The use of BSB Wizard is recommended even for an expert user. If necessary, the basic processor system can later be manually configured to fit specific project needs. See "Appendix I: Creating a Basic Processor System Using Base System Builder," page 17, for more details about using the BSB Wizard.

Configuring the MPMC and the VFBC

To configure the MPMC and the VFBC:

1. In the System Assembly view of Platform Studio, double click on `DDR2_SDRAM` to bring up the IP configuration dialog.
2. In the IP configuration dialog, select **PLB46** for PORT0, **VFBC** for PORT2 and PORT3, and leave the other ports as **INACTIVE**. Set the Base Address to `0x10000000`.



The pinout of MPMC must be compatible with MIG ace

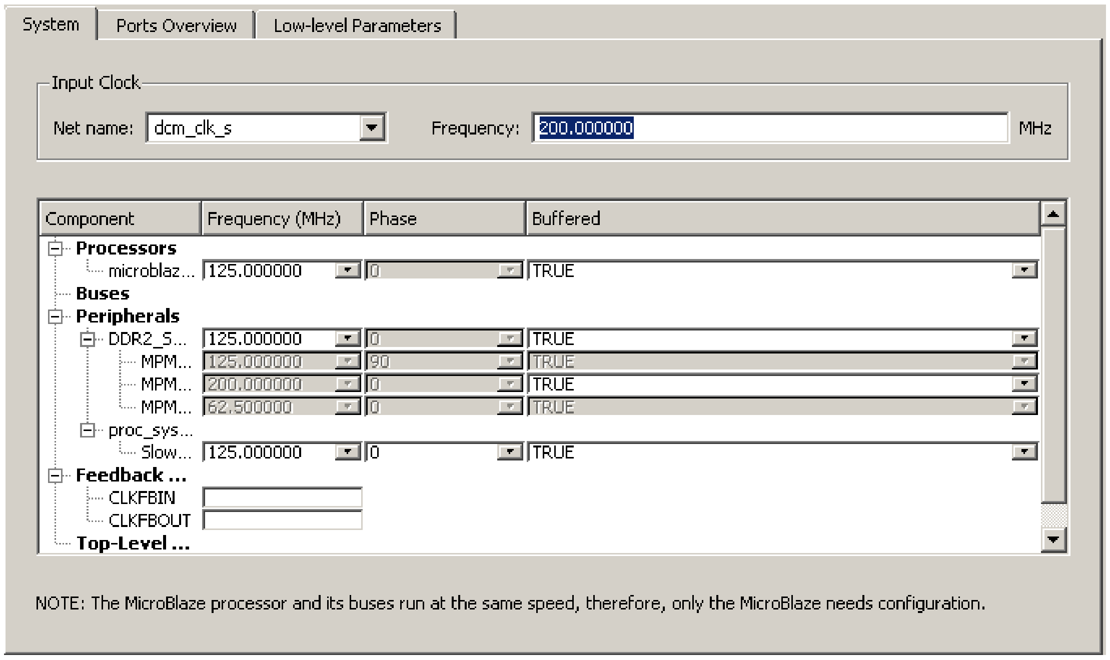
3. Click the Memory Interface tab and make sure that it targets mt4ht3264h-53e for ML506.
4. Accept all other default settings and hit **OK.**
5. Connect the VFBC ports to the top-level external ports of the processor system.

   There are two ways to accomplish this task. One is through the Platform Studio GUI. The user can configure one VFBC port using the GUI and duplicate the same configuration for the other ports by directly modifying the MHS (Microprocessor Hardware Specification) text file. Refer to the MHS file provided with this application note for the manual modification. For a non-expert user, Xilinx highly recommends using the GUI over the text editing method. The process can be sped up by selecting multiple VFBC nets before adding them to external ports in XPS (see "Appendix VI: Design Iterations between System Generator and XPS").

Configuring the Input Clock

The System Generator dual clock wiring (shown in Figure 3) is used for hardware co-simulation. The imported processor system is driven directly by the input clock to the System Generator hardware co-simulation module. The System Generator ML506 hardware co-simulation plug-in uses a 200 MHz input clock source, rather than the 100 MHz input clock source used by the BSB Wizard. Therefore, the input clock frequency of the clock_generator Pcore needs to be changed to 200 MHz.

In the System Assembly view of Platform Studio, double-click on the `clock_generator` peripheral. In the IP configuration dialog shown in Figure 4, change the `Required frequency (Hz)` of the input clock `CLKIN` from 100,000,000 to 200,000,000.



Net name: dcm_clk_s
Frequency: 200.000000 MHz

Component	Frequency (MHz)	Phase	Buffered
Processors			
microblaz...	125.000000	0	TRUE
Buses			
Peripherals			
DDR2_S...	125.000000	0	TRUE
MPM...	125.000000	90	TRUE
MPM...	200.000000	0	TRUE
MPM...	62.500000	0	TRUE
proc_sys...			
Slow...	125.000000	0	TRUE
Feedback ...			
CLKFBIN			
CLKFBOUT			
Top-Level ...			

NOTE: The MicroBlaze processor and its buses run at the same speed, therefore, only the MicroBlaze needs configuration.

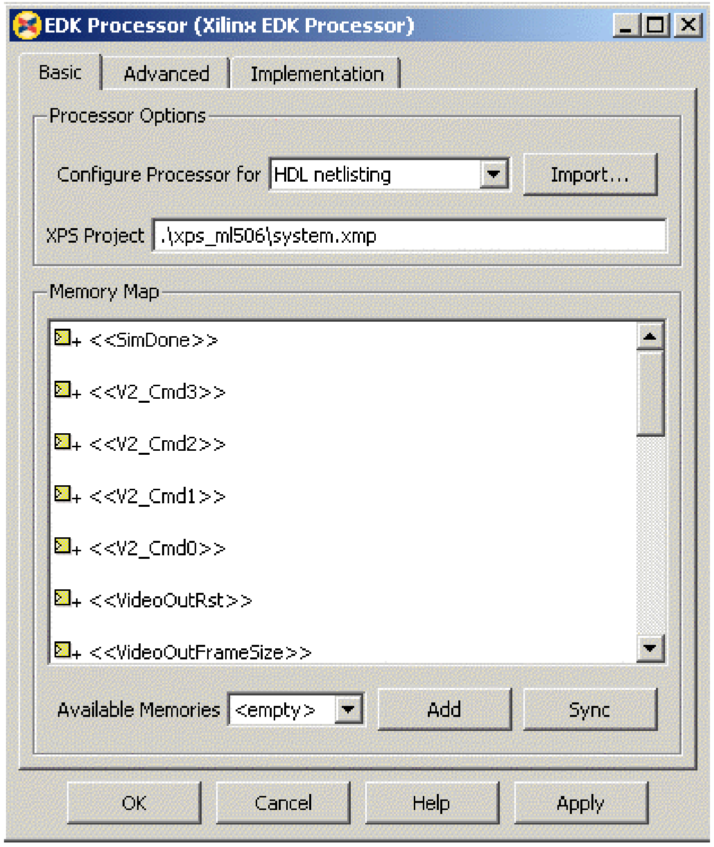
Figure 4: **`clock_generator` IP Configuration GUI**

Importing the Basic Processor System into System Generator

After creating and configuring the basic processor system, perform the following steps to connect it into a System Generator model.

1. Add an EDK Processor block from the System Generator block set library to the model.
2. Double-click on the EDK block to bring up the block GUI.
3. In the **Basic** tab, select `HDL netlisting` from the pull-down menu and import the XPS project that was created in the previous step.

   After successfully importing the XPS project, hit the **Add** button and the **Apply** button to add all the available shared memory blocks in the model to the processor memory map.

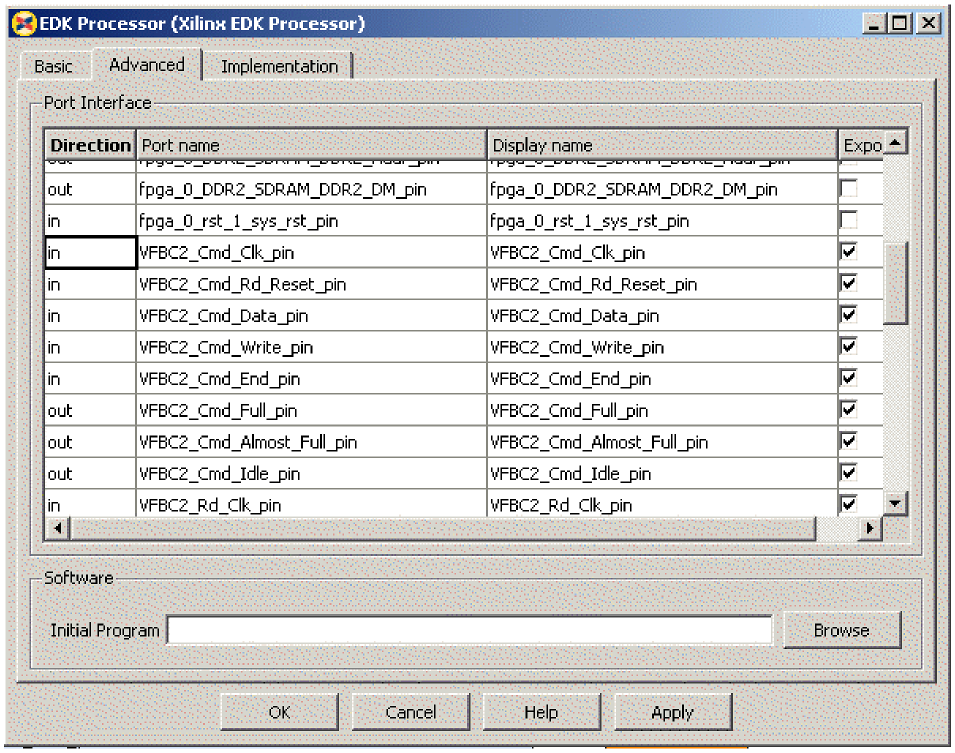


Configure Processor for: HDL netlisting

XPS Project: .\xps_ml506\system.xmp

Available Memories: <empty>

4. In the **Advanced** tab, expose all of the VFBC ports (port names starting with VFBC2_) by selecting the check boxes in the Exposed column.



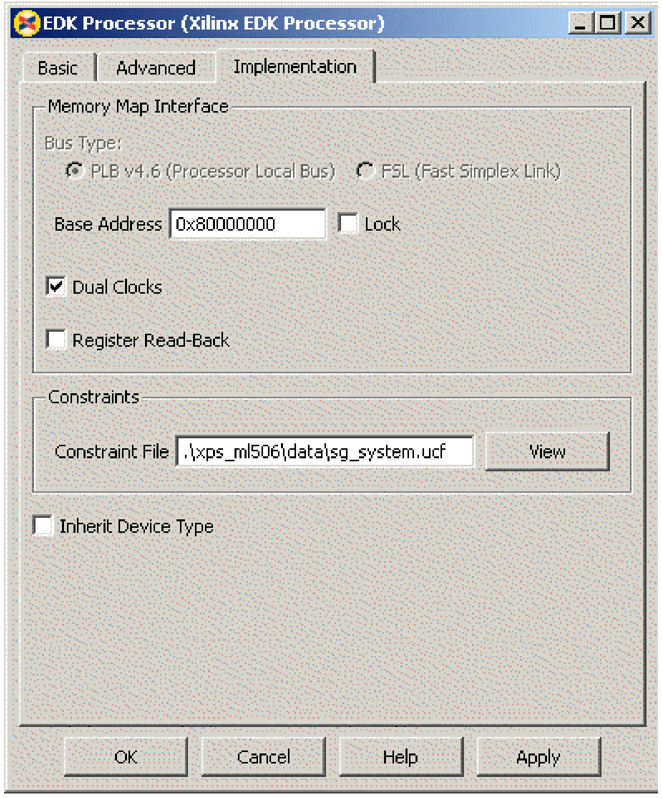
Port Interface

Direction	Port name	Display name	Expo
out	fpga_0_DDR2_SDRAM_DDR2_DM_pin	fpga_0_DDR2_SDRAM_DDR2_DM_pin	☐
in	fpga_0_rst_1_sys_rst_pin	fpga_0_rst_1_sys_rst_pin	☐
in	VFBC2_Cmd_Clk_pin	VFBC2_Cmd_Clk_pin	✓
in	VFBC2_Cmd_Rd_Reset_pin	VFBC2_Cmd_Rd_Reset_pin	✓
in	VFBC2_Cmd_Data_pin	VFBC2_Cmd_Data_pin	✓
in	VFBC2_Cmd_Write_pin	VFBC2_Cmd_Write_pin	✓
in	VFBC2_Cmd_End_pin	VFBC2_Cmd_End_pin	✓
out	VFBC2_Cmd_Full_pin	VFBC2_Cmd_Full_pin	✓
out	VFBC2_Cmd_Almost_Full_pin	VFBC2_Cmd_Almost_Full_pin	✓
out	VFBC2_Cmd_Idle_pin	VFBC2_Cmd_Idle_pin	✓
in	VFBC2_Rd_Clk_pin	VFBC2_Rd_Clk_pin	✓

Software

Initial Program

5. In the **Implementation** tab, select the `Dual Clock` check box. Then click **OK** to close the EDK Processor block GUI.



Memory Map Interface

Bus Type: [✓ PLB v4.6 (Processor Local Bus)] FSL (Fast Simplex Link)

Base Address 0x80000000 ☐ Lock

☑ Dual Clocks

☐ Register Read-Back

Constraints

Constraint File .\xps_ml506\data\sg_system.ucf

☐ Inherit Device Type

6. When targeting a Xilinx ML506 board, perform the following steps.

   Note: For other boards, these steps are not needed.

   - For convenience, the contents of the "`edkprocessor_ddr2fix.m`" utility has been added to the Initialization tab of the EDK Processor block to remove the IO pads for the MPMC/DDR2 IP cores to avoid double-buffering issues during synthesis. The utility will be executed every time the block is configured.
   - Alternatively, the utility can be manually executed by selecting the EDK Processor block and typing the following command in the MATLAB console: **`edkprocessor_ddr2fix(gcbh)`**

7. Connect the embedded system to the System Generator control logic as shown in Figure 5. This should be the same as the `uBlaze_subsystem` subsystem in the `vfbc.mdl` model.

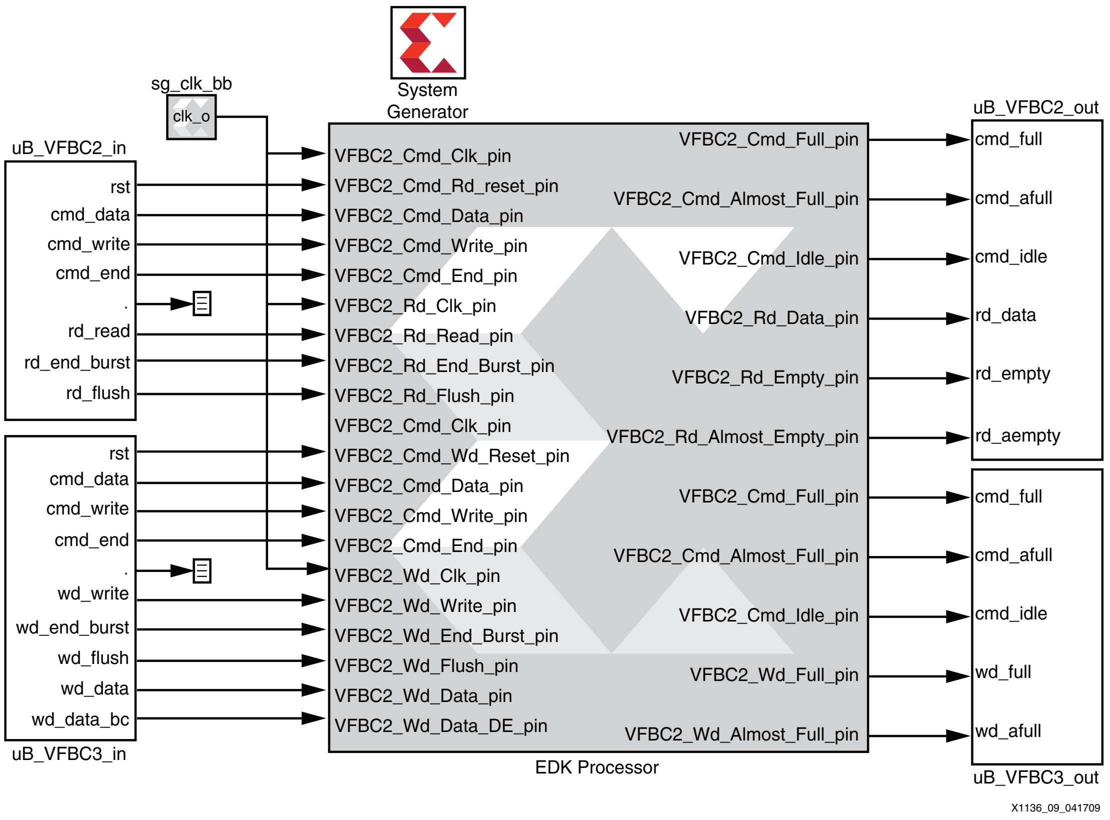


Figure 5: **Connecting the EDK Processor Block with Other System Generator Blocks**

Building the System Generator Validation Model

This section presents the top level view of the System Generator model and discusses memory-mapped shared memory blocks and the `PassThru_SubSystem` subsystem.

Top Level View

The top level view of the System Generator model is shown in Figure 6. The major modeling blocks and subsystems include the following.

- Embedded MALAB Function (`read_YUV_420`) is an embedded MATLAB function used to process input simulation text files and write stimulus signals into the `video_in` FIFO.
- Embedded MALAB Function (`write_YUV_420`) is an embedded MATLAB function used to process output simulation text files and read output simulation results from the `video_out` FIFO.
- The `Stimulus_vector` subsystem passes the frame parameters from the Simulink modeling environment to the MicroBlaze processor running in hardware.
- The `PassThru_SubSystem` subsystem includes the imported MicroBlaze processor system and the user logic described in System Generator. This subsystem is pushed down to the hardware during hardware co-simulation. This subsystem is discussed in further detail in "The `PassThru_SubSystem` Subsystem," page 11.
- The `Stop Simulation` block waits for the MicroBlaze (running in hardware) to declare that it has finished processing the frame data and then stops the Simulink simulation.

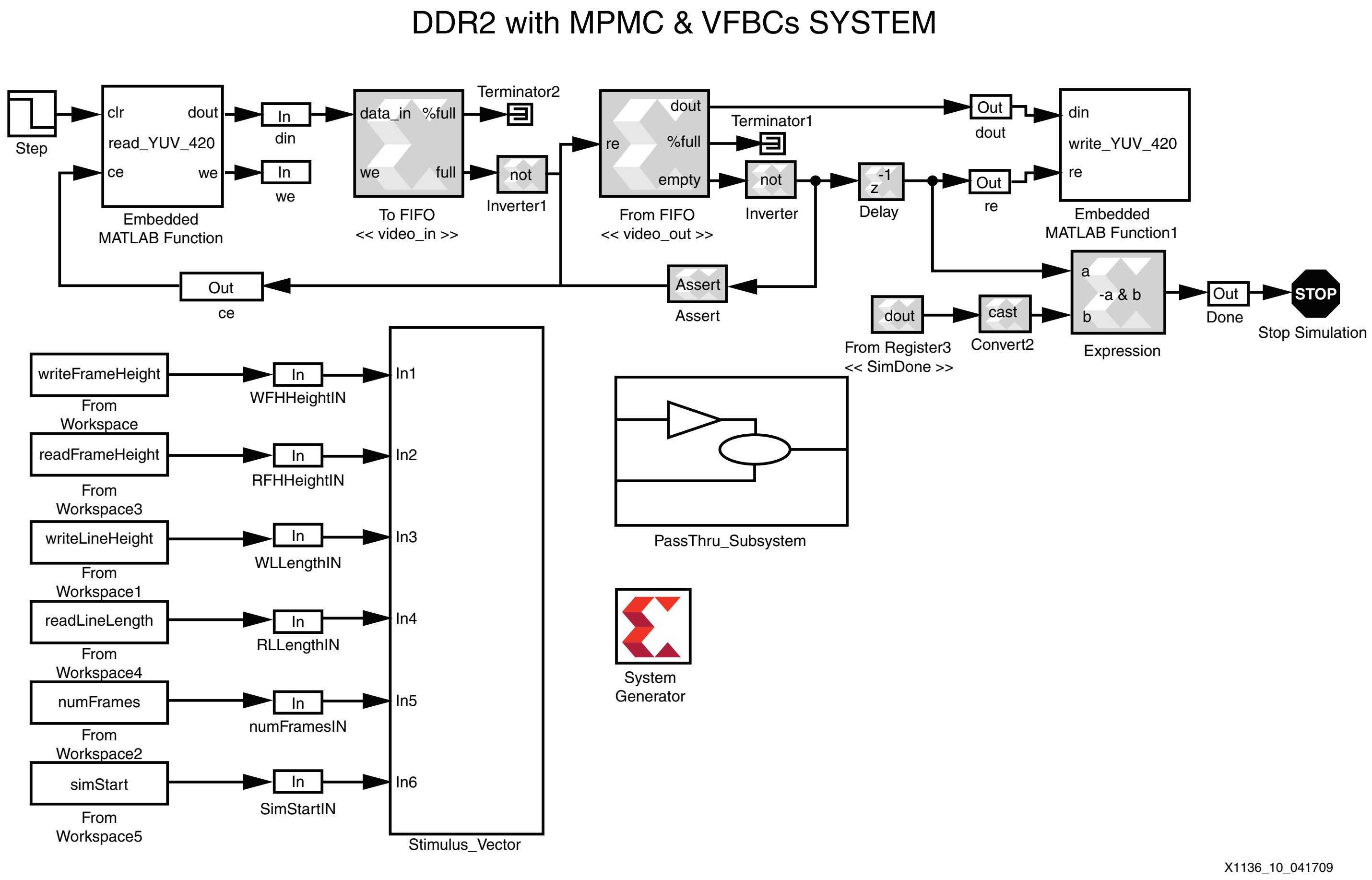


Figure 6: **Top-level View of the System Generator Model**

Memory-Mapped Shared Memory Blocks

All the From/To Register blocks and From/To FIFO blocks in the System Generator model are added to the MicroBlaze memory map. That way, they can be accessed by the MicroBlaze through the provided software driver and by the user logic in the System Generator model, and vice versa.

The seven From/To Register blocks shown in Table 1 are used to pass the various simulation parameters (for example, *`writeFrameHeight`*, *`writeLineLength`*, *`numFrames`*) from the Simulink modeling environment to the MicroBlaze processor running in hardware. By doing so, customers can easily change their simulation parameters and the VFBC commands, thus significantly increasing the simulation iteration time. For example, if users want to try out new VFBC commands, they can modify and re-compile the C code and update the hardware co-simulation bit stream. After that, the users are ready to restart hardware co-simulation with the new VFBC commands. There is no need to rerun the time-consuming synthesis, map, and place-and-route to generate a new hardware co-simulation block.

Two shared FIFO blocks are used in the validation platform. The `video_in` To FIFO block is used to pass the input video stimulus from the host PC to the MicroBlaze processor running in hardware. The `video_out` From FIFO block is used to retrieve the output video data from the MicroBlaze processor and send it back to the host PC.

Table 1: **Functionalities of the Shared Memories in the System Generator Model**

Name	Type	Function
WriteFrameHeight	To Register	Pass write frame height to MicroBlaze
WriteLineLength	To Register	Pass write frame length to MicroBlaze
numFrames	To Register	Pass the number of frame to MicroBlaze
ReadFrameHeight	To Register	Pass read frame height to MicroBlaze
ReadLineLength	To Register	Pass read frame length to MicroBlaze
SimStart	To Register	Inform MicroBlaze that the hardware co-simulation has started
SimDone	From Register	Wait for MicroBlaze to finish processing the data

The `PassThru_SubSystem` Subsystem

The `PassThru_SubSystem` subsystem is comprised of four major subsystems: `uBlaze_subsystem`, `Video_In`, `PassThru`, and `Video_Out`. During validation through hardware co-simulation, these four subsystems are pushed down to run on the hardware.

- `uBlaze_subsystem` (shown in Figure 5) contains a basic processor system with a MicroBlaze processor, a MPMC/DDR2/VFBC peripheral, and an RS232_Uart_Lite peripheral.
- `VideoIn(VFBC3)` reads the input stimulus data from the Embedded MATLAB Function block (`read_YUV_420`) utilizing a `video_in` To FIFO block. The data is then processed and stored in the external DDR2 SDRAM through the VFBC3 port.
- `VideoOut(VFBC2)` retrieves the data stored in the DDR2 memory through the VFBC2 port.

MicroBlaze communicates with the DDR2 SDRAM by issuing commands to VFBC2 and VFBC3 on the MPMC peripheral through the PLB interface. The video data flow is controlled by MicroBlaze, which is shown in Figure 7 and explained following the figure.

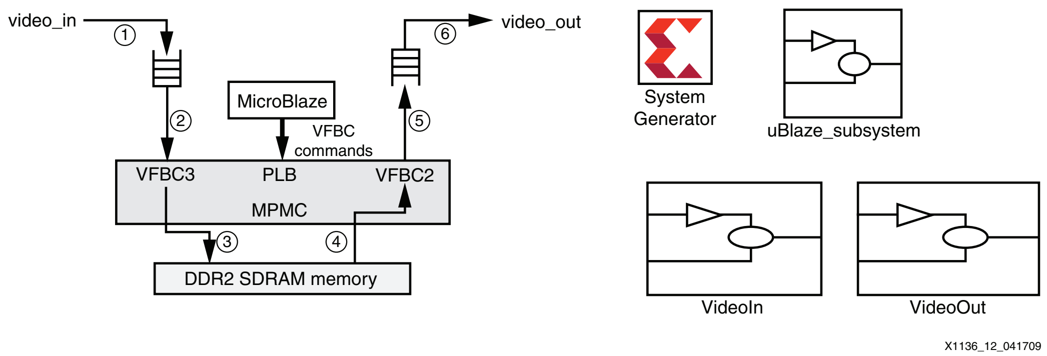


Figure 7: **Video Data Flow Controlled by MicroBlaze Block Diagram (left) and Actual System Generator Model (right)**

Step 1: The System Generator hardware co-simulation module delivers the input video data from Simulink to the `video_in` FIFO running on the FPGA device.

Steps 2 and 3: MicroBlaze issues commands to the MPMC peripheral so that the video data is transferred from the `video_in` FIFO to the external DDR2 SDRAM memory through VFBC3.

Steps 4 and 5: MicroBlaze issues commands to the MPMC peripheral so that the video data is transferred from the external DDR2 SDRAM memory to the `video_out` FIFO to through VFBC2.

Step 6: The System Generator hardware co-simulation module delivers the output video data to Simulink from the `video_out` FIFO running on the FPGA device. The `video_out` simulation results are then written to `<Current Directory>/level5/test00x/results/YUV.out.txt` for validation purposes

As discussed in "VFBC Transfer Commands," page 22, each VFBC command has 128 bits. MicroBlaze sends out a 32-bit data word on each PLB transaction. Therefore, as shown in Figure 8, four 32-bit From Register blocks are used to read in commands from MicroBlaze. This user logic is used in VideoIn and VideoOut blocks.

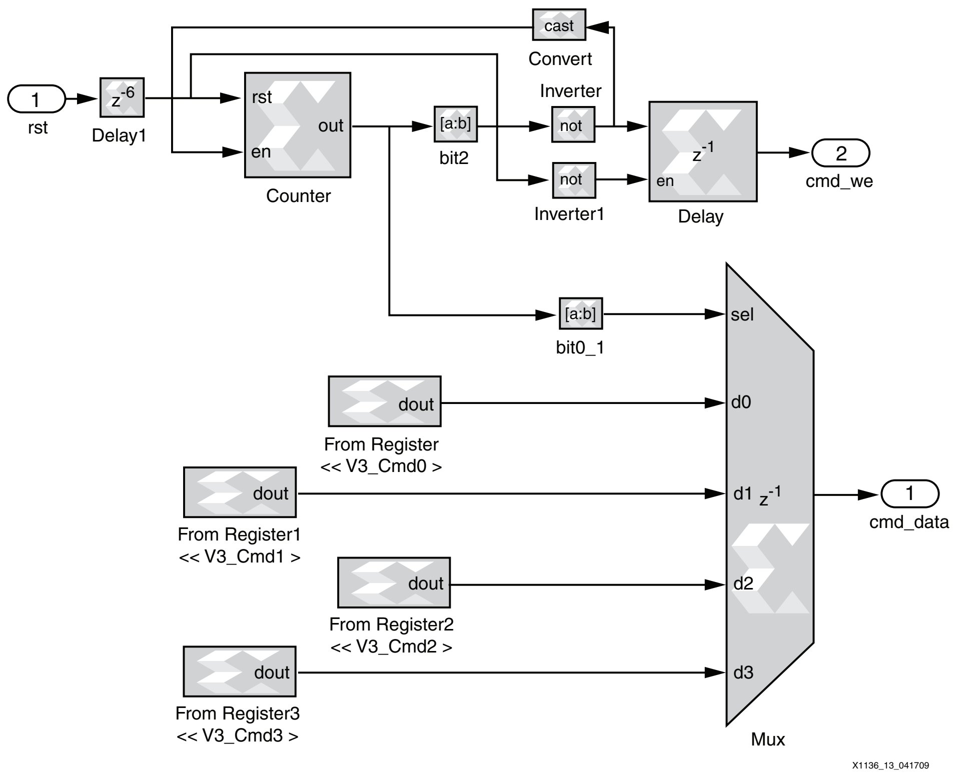
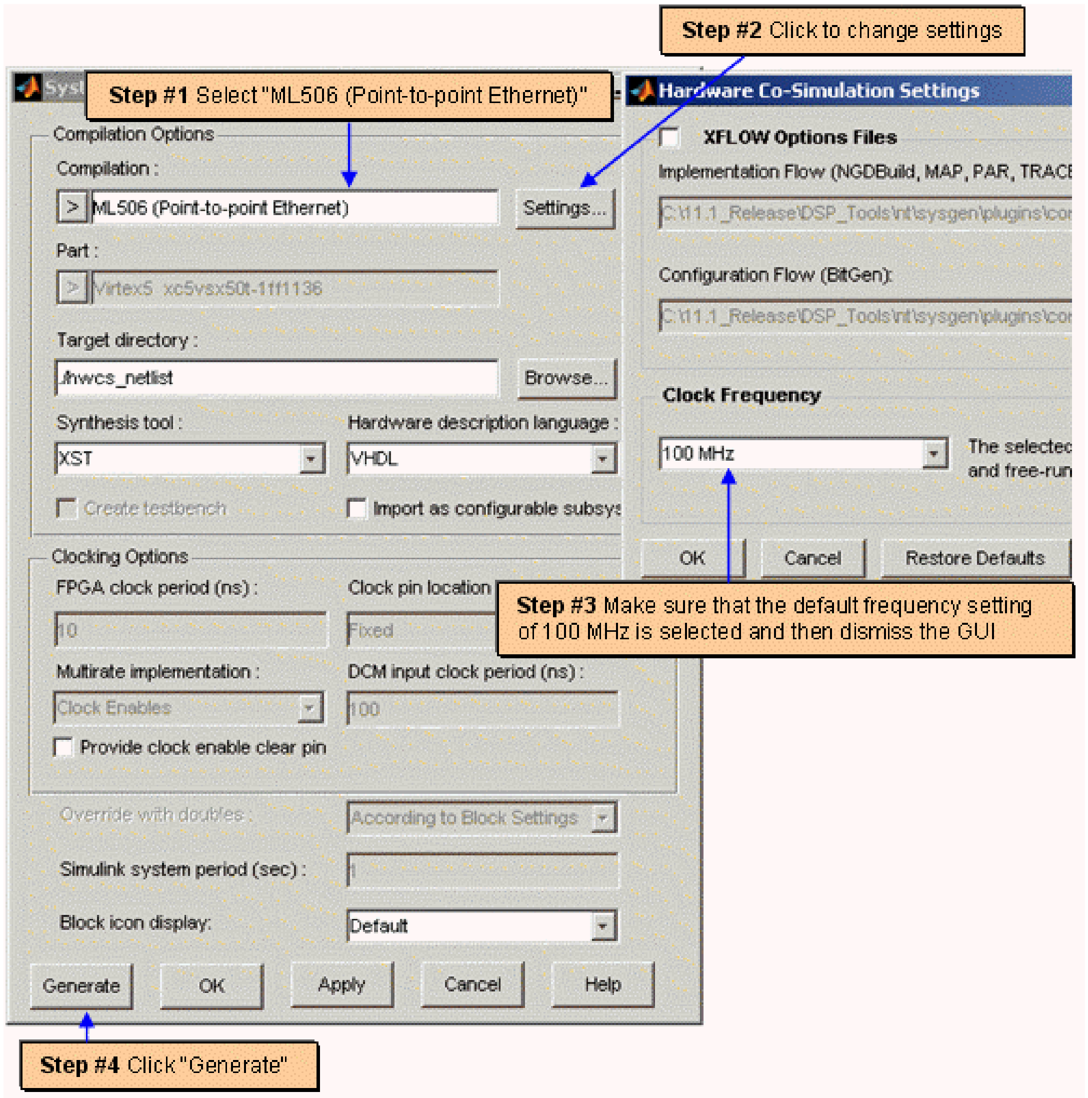


Figure 8: **Handling VFBC Commands with Four 32-bit From Register Blocks**

Generating a Hardware Co-simulation Block

The basic processor system and the System Generator control logic can run in the hardware through hardware co-simulation. Make sure that all the Simulink workspace variables and other simulation parameters are set up properly. After that, double-click on the System Generator block at the hierarchy level at which the hardware is to be run. Follow the step sequence shown in Figure 9 to generate the hardware co-simulation block.


Step #1 Select "ML506 (Point-to-point Ethernet)"

Step #2 Click to change settings

Step #3 Make sure that the default frequency setting of 100 MHz is selected and then dismiss the GUI

Step #4 Click "Generate"

Compilation Options
Compilation : ML506 (Point-to-point Ethernet)
Part : Virtex5 xc5vsx50t-1ff1136
Target directory : ./hwcs_netlist
Synthesis tool : XST
Hardware description language : VHDL
Create testbench
Import as configurable subsys

Clocking Options
FPGA clock period (ns) : 10
Clock pin location : Fixed
Multirate implementation : Clock Enables
DCM input clock period (ns) : 100
Provide clock enable clear pin

Override with doubles : According to Block Settings
Simulink system period (sec) : 1
Block icon display: Default

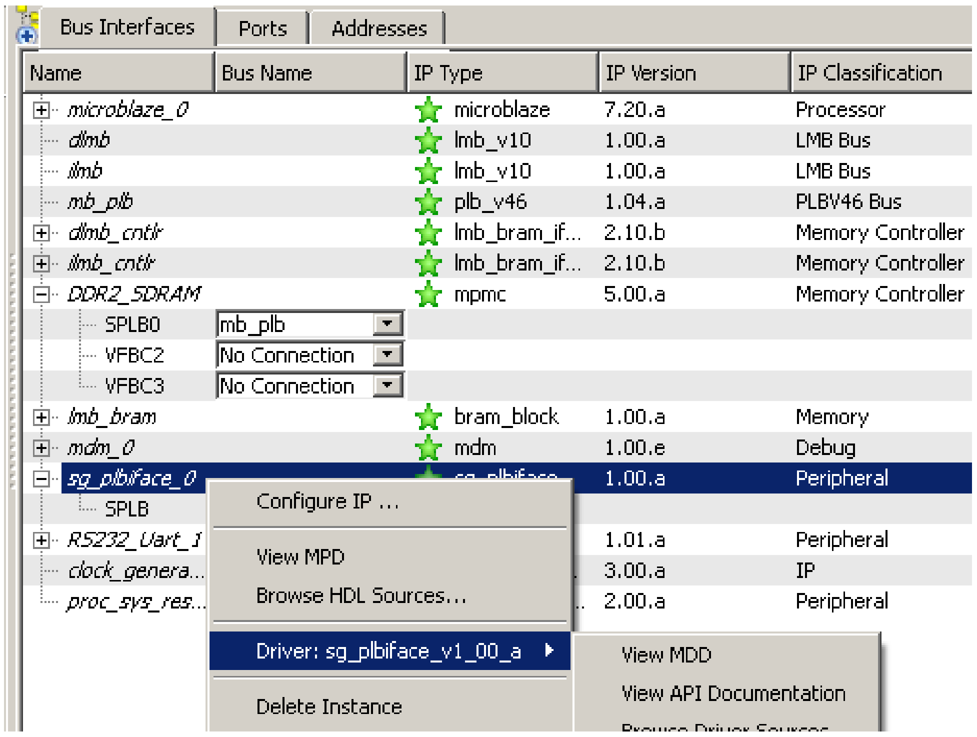
Hardware Co-Simulation Settings
XFLOW Options Files
Implementation Flow (NGDBuild, MAP, PAR, TRACE
C:\11.1_Release\DSP_Tools\nt\sysgen\plugins\co
Configuration Flow (BitGen):
C:\11.1_Release\DSP_Tools\nt\sysgen\plugins\co
Clock Frequency
100 MHz
The selected
and free-run

Figure 9: **Steps to generate the Hardware Co-simulation Block**

Writing Software to Control VFBCs

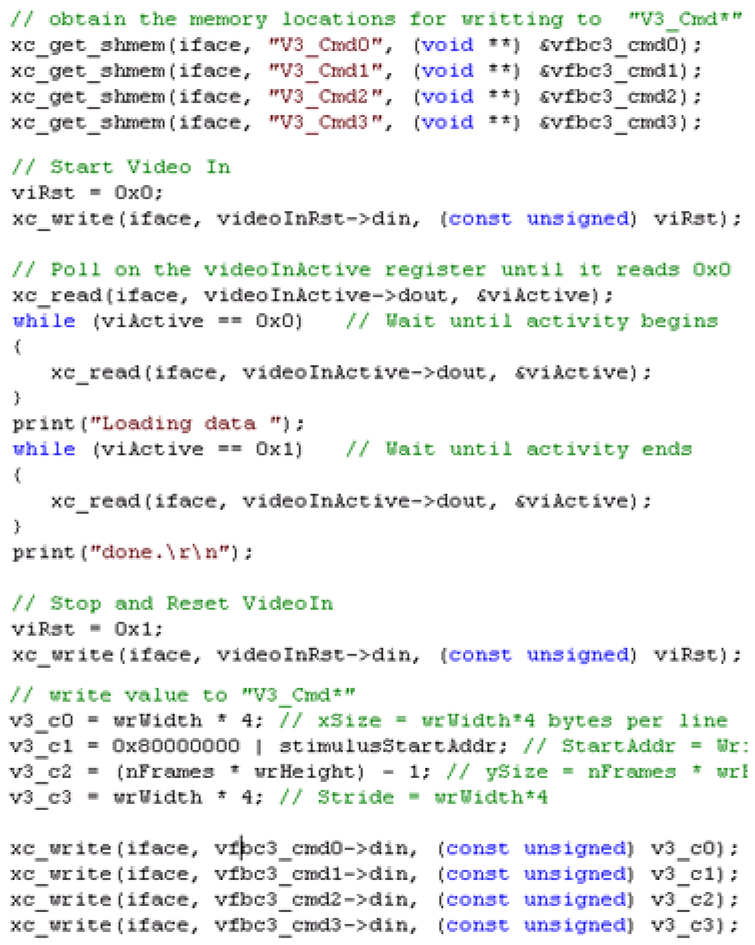
After the hardware co-simulation block is successfully generated, the user can then write C code to perform various tasks, such as obtaining the simulation parameters and other settings statuses from the System Generator shared memories before issuing Read/Write commands to DDR_SDRAM through the selected VFBC ports. A polling method can also be used from a simple while-loop to check for a certain state of a valid control signal. This method is commonly used to check for a valid state of To/From FIFO Shared Memories to prevent writing to a full FIFO or reading from an empty FIFO.

The C code uses the software driver API provided by System Generator. Follow the steps in Figure 10 to examine the API documentation of the System Generator `sg_plbiface_0` peripheral from the XPS GUI. Note that the names of the shared memories accessible through the API are identical to those found in the System Generator model.



Name	Bus Name	IP Type	IP Version	IP Classification
microblaze_0		microblaze	7.20.a	Processor
dlmb		lmb_v10	1.00.a	LMB Bus
ilmb		lmb_v10	1.00.a	LMB Bus
mb_plb		plb_v46	1.04.a	PLBV46 Bus
dlmb_cntlr		lmb_bram_if...	2.10.b	Memory Controller
ilmb_cntlr		lmb_bram_if...	2.10.b	Memory Controller
DDR2_SDRAM		mpmc	5.00.a	Memory Controller
SPLB0	mb_plb			
VFBC2	No Connection			
VFBC3	No Connection			
lmb_bram		bram_block	1.00.a	Memory
mdm_0		mdm	1.00.e	Debug
sg_plbiface_0		sg_plbiface	1.00.a	Peripheral
SPLB				
RS232_Uart_1			1.01.a	Peripheral
clock_genera...			3.00.a	IP
proc_sys_res...			2.00.a	Peripheral

Figure 10: **Access the API Documentation of the System Generator Pcore**



```
// obtain the memory locations for writting to  "V3_Cmd*"
xc_get_shmem(iface, "V3_Cmd0", (void **) &vfbc3_cmd0);
xc_get_shmem(iface, "V3_Cmd1", (void **) &vfbc3_cmd1);
xc_get_shmem(iface, "V3_Cmd2", (void **) &vfbc3_cmd2);
xc_get_shmem(iface, "V3_Cmd3", (void **) &vfbc3_cmd3);

// Start Video In
viRst = 0x0;
xc_write(iface, videoInRst->din, (const unsigned) viRst);

// Poll on the videoInActive register until it reads 0x0
xc_read(iface, videoInActive->dout, &viActive);
while (viActive == 0x0)   // Wait until activity begins
{
   xc_read(iface, videoInActive->dout, &viActive);
}
print("Loading data ");
while (viActive == 0x1)   // Wait until activity ends
{
   xc_read(iface, videoInActive->dout, &viActive);
}
print("done.\r\n");

// Stop and Reset VideoIn
viRst = 0x1;
xc_write(iface, videoInRst->din, (const unsigned) viRst);

// write value to "V3_Cmd*"
v3_c0 = wrWidth * 4; // xSize = wrWidth*4 bytes per line
v3_c1 = 0x80000000 | stimulusStartAddr; // StartAddr = Wr
v3_c2 = (nFrames * wrHeight) - 1; // ySize = nFrames * wr
v3_c3 = wrWidth * 4; // Stride = wrWidth*4

xc_write(iface, vfbc3_cmd0->din, (const unsigned) v3_c0);
xc_write(iface, vfbc3_cmd1->din, (const unsigned) v3_c1);
xc_write(iface, vfbc3_cmd2->din, (const unsigned) v3_c2);
xc_write(iface, vfbc3_cmd3->din, (const unsigned) v3_c3);
```

One useful technique is to use a while-loop to wait for a shared FIFO to be in the correct state before performing read/write operations to it from the C code.

While it is highly recommended to use the automatically-generated API for code portability, the user may want to perform read/write operations to the shared memories directly through pointer arithmetic due to performance consideration. See the Driver Performance Optimization section in the software API documentation for more details. The complete C code is provided in the reference design files.

After finishing editing the C code, add a new Software Application Project named SysGen_VFBC in XPS, which is shown in Figure 11. Double click the Sources folder of the software project and add to it the C code located at **<Current Directory>/C-code/vfbc.c**.

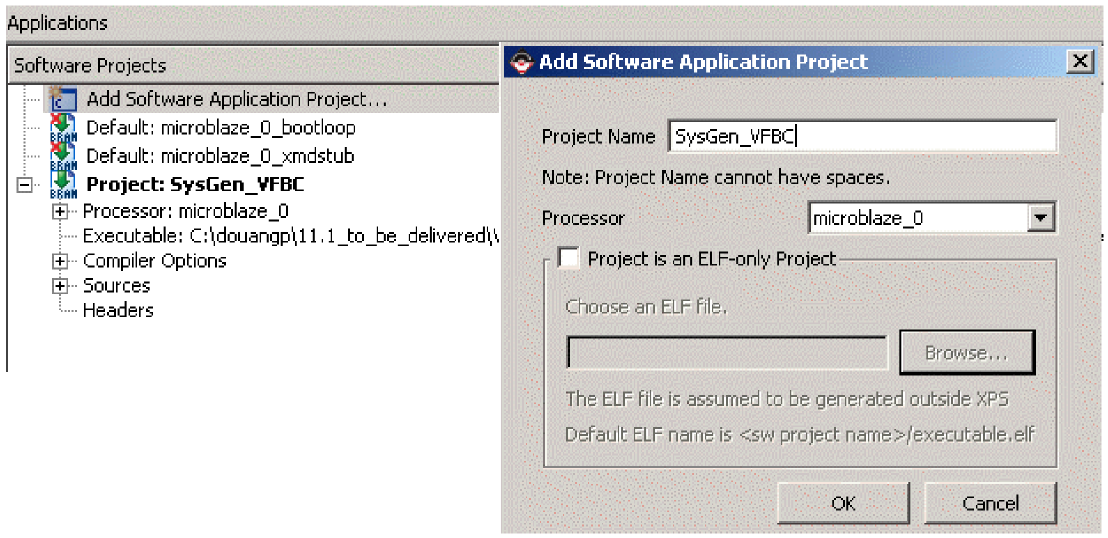

Applications
Software Projects
Add Software Application Project...
Default: microblaze_0_bootloop
Default: microblaze_0_xmdstub
Project: SysGen_VFBC
Processor: microblaze_0
Executable: C:\douangp\11.1_to_be_delivered\\
Compiler Options
Sources
Headers
Add Software Application Project
Project Name
SysGen_VFBC
Note: Project Name cannot have spaces.
Processor
microblaze_0
Project is an ELF-only Project
Choose an ELF file.
Browse...
The ELF file is assumed to be generated outside XPS
Default ELF name is <sw project name>/executable.elf
OK
Cancel

Figure 11: **Adding a Software Application Project in XPS**

Performing Validation through Hardware Co-simulation

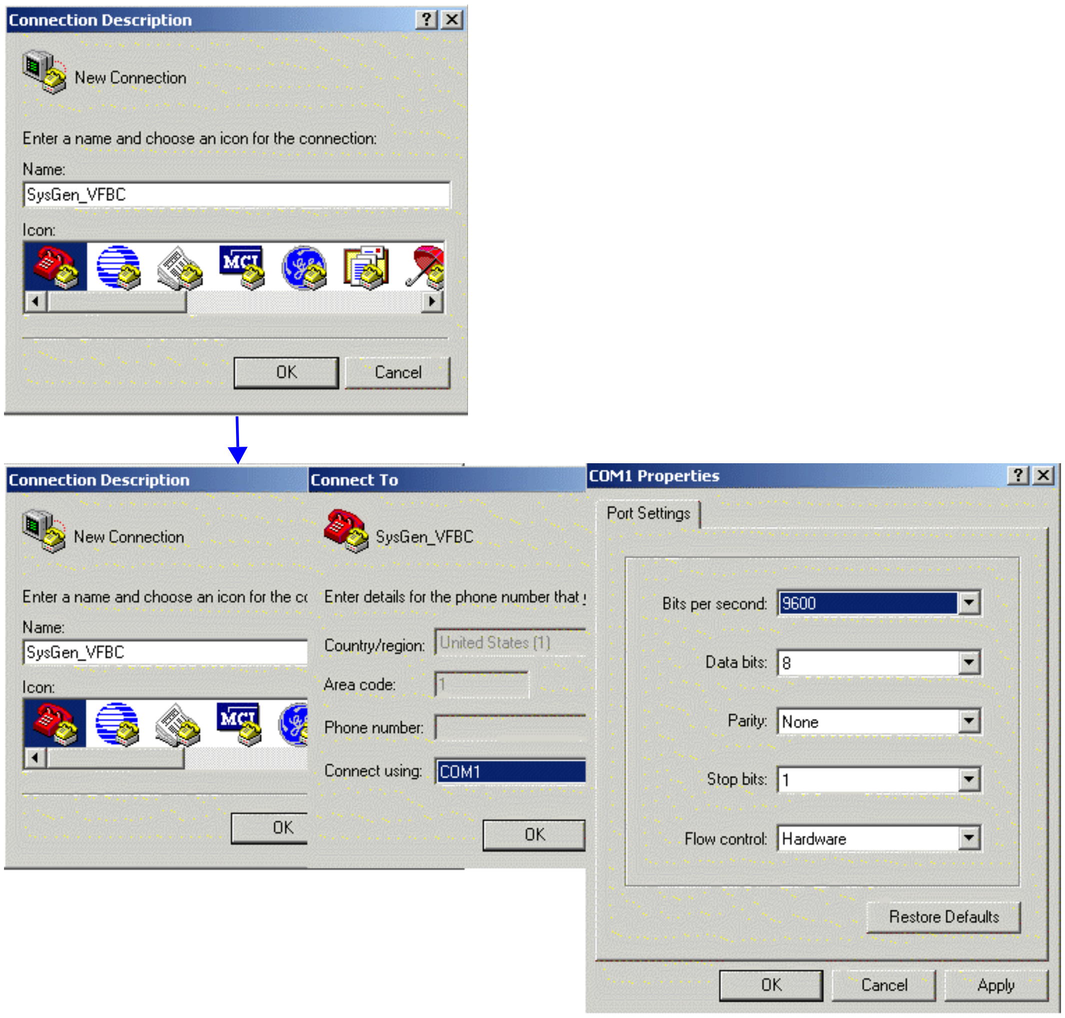
This section provides information on connecting the hardware, setting up the Hyper Terminal program and input stimulus, and running hardware co-simulation.

Connecting the Hardware

1. Connect the ML506 board to the computer via a standard 9-pin serial cable. A USB adaptor may be used if the computer does not have a serial port.
2. Connect the Ethernet cable from the computer to the ML506 board.
3. Connect a 5-volt power supply to the ML506 board.
4. Turn on the power of the ML506 board.
5. If needed, refer to the System Generator User Guide to set up a 10/100 Fast Ethernet or a Gigabit Ethernet Adapter on the PC.

Setting Up the Hyper Terminal Program

1. Launch HyperTerminal program: **All Program** → **Accessories** → **Communications** → **HyperTerminal**
2. Select **File** → **New Connections** and set up as follows:



Enter a name and choose an icon for the connection:

Name: SysGen_VFBC

SysGen_VFBC

Enter details for the phone number that

Country/region: United States (1)

Area code: 1

Phone number:

Connect using: COM1

Bits per second: 9600

Data bits: 8

Parity: None

Stop bits: 1

Flow control: Hardware

Setting Up the Input Stimulus

If not already created, create the following subfolders:

- `<Current Directory>\level5\test001\results`
- `<Current Directory>\level5\test001\stimuli\YUV.in.hdr`
- `<Current Directory>\level5\test002\results`
- `<Current Directory>\level5\test002\stimuli\YUV.in.hdr`
- `<Current Directory>\level5\test003\results`
- `<Current Directory>\level5\test003\stimuli\YUV.in.hdr`

The `YUV.in.hdr` files are header files that contain simulation parameters. These parameters are read into the Simulink workspace during the generation of the hardware co-simulation block and during hardware co-simulation. These files may be edited directly to change the frame resolution.

Executing `stimulus_generator.m` in the MATLAB console generates the stimulus text file at `<Current Directory>\level5\test00x\stimuli\YUV.in.txt` file. This M-code file needs to be executed only once.

Running Hardware Co-simulation

Two MATLAB software scripts are provided for convenience.

- Execute `simulink_hwcs_script.m` to run the Simulink-based hardware co-simulation flow.
- Execute `matlab_hwcs_script_top.m` to run the MATLAB hardware co-simulation flow.

A rough estimation of the simulation times of these two hardware co-simulation flows is presented in Table 2.

Table 2: **Simulation Times of the Two Hardware Co-simulation Flows**

	800x600		**1280x720**		**1920x1080**	
Number of Frames	Simulink HWCS	MATLAB HWCS	Simulink HWCS	MATLAB HWCS	Simulink HWCS	MATLAB HWCS
1	1729 sec	22 sec	3294 sec	29 sec	7440 sec	43 sec
3	N/A	34 sec	N/A	51 sec	N/A	94 sec
5	N/A	45 sec	N/A	75 sec	N/A	147 sec

As demonstrated, the MATLAB hardware co-simulation, for this particular application, significantly improves simulation speed with the same number of frames and video resolutions. This is mainly accomplished by removing the Simulink modeling overhead.

Conclusion

This application note provided the basic knowledge on how to integrate an embedded processor system with the Xilinx Multi-Port Memory Controller (MPMC) and Video Frame Buffer Controller (VFBC) IP cores in System Generator for DSP (“System Generator”). Users can apply the same design flows and methodologies to develop other similar applications tailored specifically to their needs and design environments.

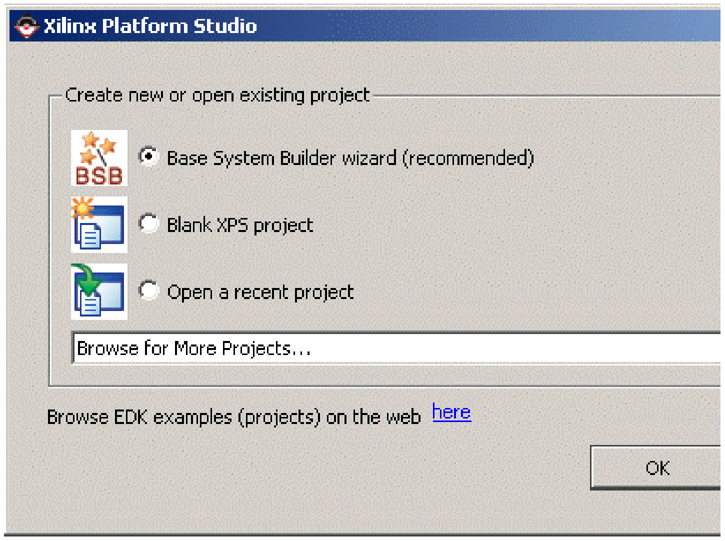
This application note presented a truly unique capability in the development of DSP hardware with an embedded processor unit using Xilinx ISE® Design Suite and the ML506 hardware board. By utilizing the tight integration within ISE Design Suite, design teams can quickly implement and iterate on a complex design as described in this application note. With System Generator hardware co-simulation capability, users can take advantage of an accelerated design validation process and leverage a similar validating platform for their designs.

Appendix I: Creating a Basic Processor System Using Base System Builder

Appendix I provides instructions for creating a basic processor system using the BSB.

Note: The following screen shots may be slightly different from what the user sees if different tool versions are used. If an advanced user is already familiar with the BSB, skip the following steps and build a basic processor system targeting the ML506 Evaluation Platform and configure it with the following two peripherals: DDR2 with MPMC and UART_Lite with 9600 baud rate.

1. Select Base System Builder Wizard and hit **OK**.



Xilinx Platform Studio

Create new or open existing project

(●) Base System Builder wizard (recommended)

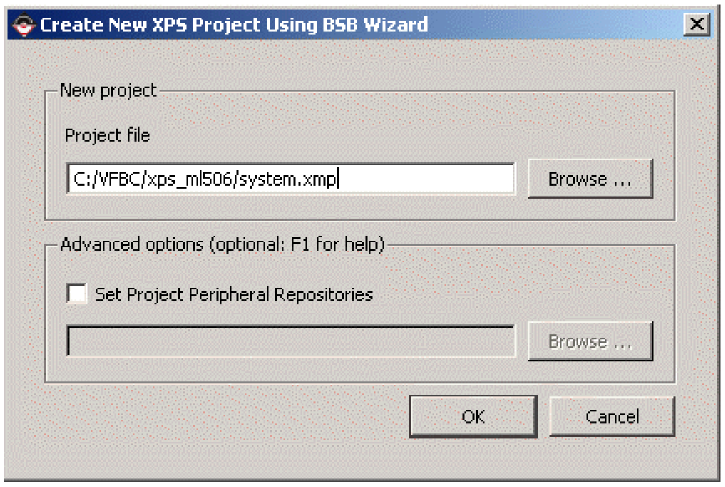
( ) Blank XPS project

( ) Open a recent project

Browse for More Projects...

Browse EDK examples (projects) on the web here

2. Provide the Project file location and hit **OK**.



Create New XPS Project Using BSB Wizard

New project

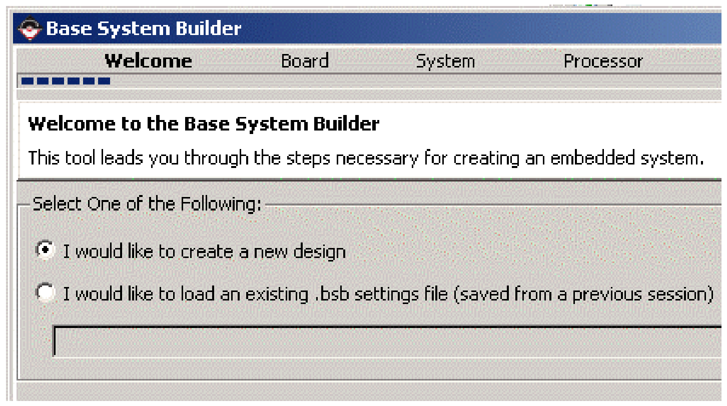
Project file

C:/VFBC/xps_ml506/system.xmp

Advanced options (optional: F1 for help)

[ ] Set Project Peripheral Repositories

3. Click **Next**.



Base System Builder

Welcome to the Base System Builder

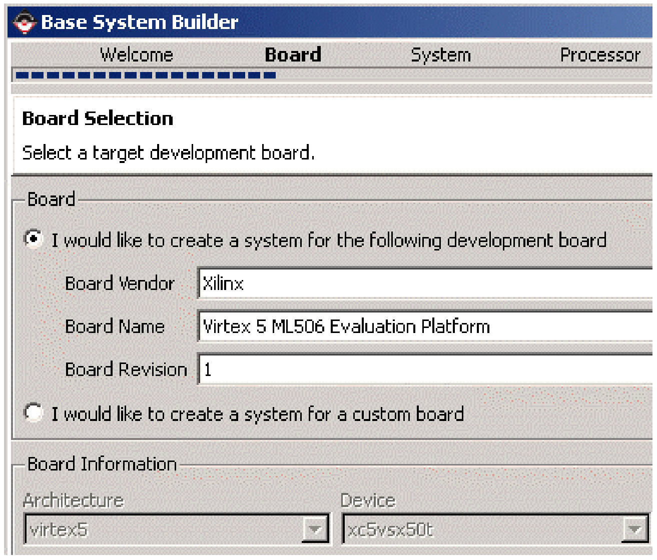
This tool leads you through the steps necessary for creating an embedded system.

Select One of the Following:

(●) I would like to create a new design

( ) I would like to load an existing .bsb settings file (saved from a previous session)

4. Select `Virtex 5 ML506 Evaluation Platform` and click **Next**.



Base System Builder

Board Selection

Select a target development board.

Board

(●) I would like to create a system for the following development board

Board Vendor: Xilinx

Board Name: Virtex 5 ML506 Evaluation Platform

Board Revision: 1

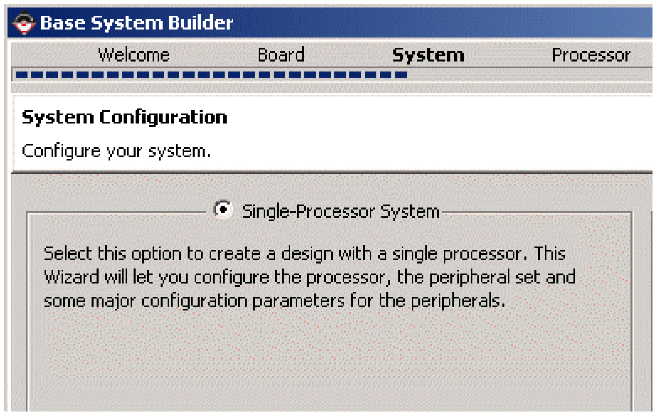
( ) I would like to create a system for a custom board

Board Information

Architecture: virtex5

Device: xc5vsx50t

5. Select a single processor system and click **Next**.



Base System Builder

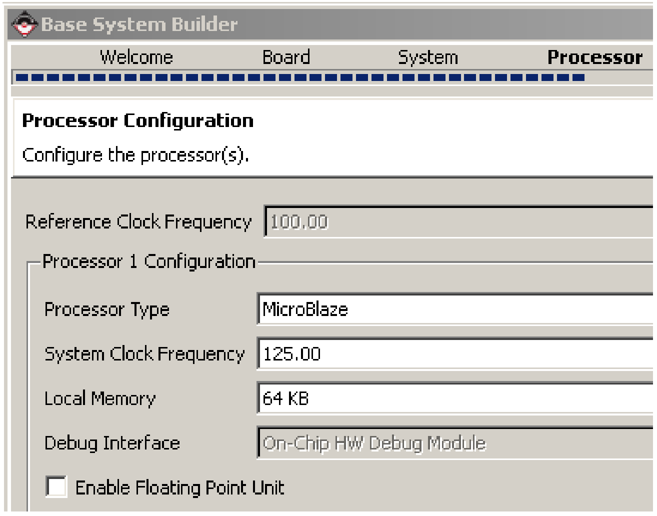
System Configuration

Configure your system.

(●) Single-Processor System

Select this option to create a design with a single processor. This Wizard will let you configure the processor, the peripheral set and some major configuration parameters for the peripherals.

6. Set the parameters as follows and click **Next**.



Base System Builder

Processor Configuration

Configure the processor(s).

Reference Clock Frequency: 100.00

Processor 1 Configuration

Processor Type: MicroBlaze

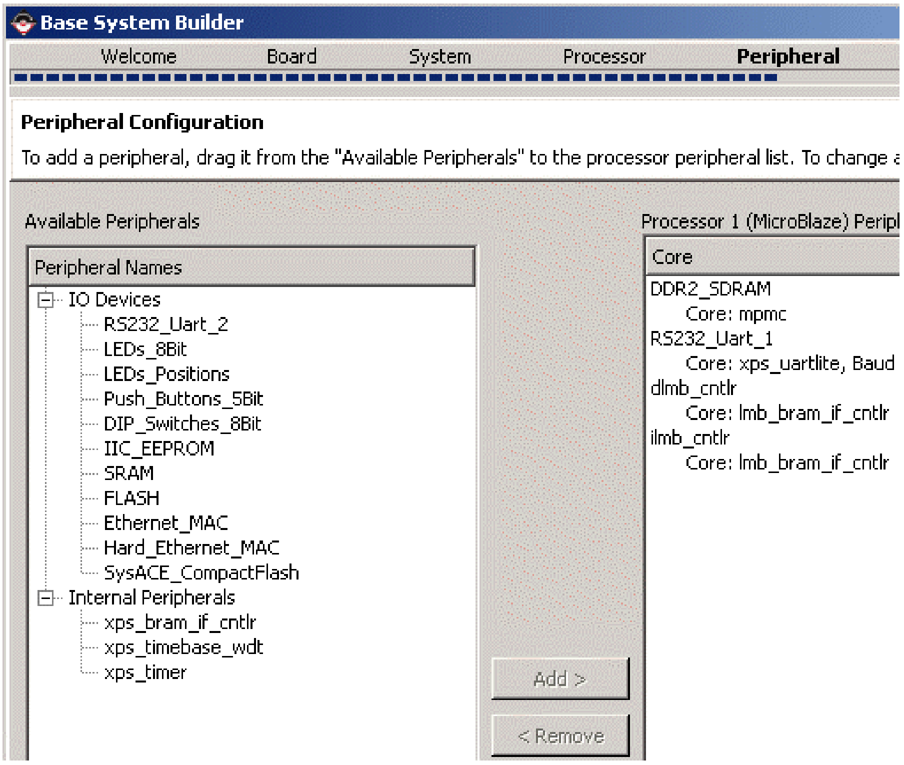
System Clock Frequency: 125.00

Local Memory: 64 KB

Debug Interface: On-Chip HW Debug Module

☐ Enable Floating Point Unit

7. Use the **Remove** button to remove unused **IO Devices** and **Internal Peripherals** under **Processor 1** at the right-hand side of the screen; click **Next**.



Base System Builder

Welcome Board System Processor **Peripheral**

Peripheral Configuration

To add a peripheral, drag it from the "Available Peripherals" to the processor peripheral list. To change a

Available Peripherals

Peripheral Names
- IO Devices
  - RS232_Uart_2
  - LEDs_8Bit
  - LEDs_Positions
  - Push_Buttons_5Bit
  - DIP_Switches_8Bit
  - IIC_EEPROM
  - SRAM
  - FLASH
  - Ethernet_MAC
  - Hard_Ethernet_MAC
  - SysACE_CompactFlash
- Internal Peripherals
  - xps_bram_if_cntlr
  - xps_timebase_wdt
  - xps_timer

Processor 1 (MicroBlaze) Peripl

Core
- DDR2_SDRAM
  - Core: mpmc
- RS232_Uart_1
  - Core: xps_uartlite, Baud
- dlmb_cntlr
  - Core: lmb_bram_if_cntlr
- ilmb_cntlr
  - Core: lmb_bram_if_cntlr

8. Click **Next** in the **Cache configuration** screen.
9. Click **Next** in the **Application configuration** screen.
10. Click **Finish** in the **Summary** screen.

Appendix II: VFBC Basics

Appendix II includes information on the following:

- "Frame Mode and Display Window"
- "VFBC Transfer Commands"

Frame Mode and Display Window

The example application uses two VFBCs, one for writing the 1080p source and one for the 480p display. The video frames are stored in external DDR2 memory. Figure 12 shows an example of a 1080p @ 60fps video source being written to a frame stored in external memory. A 480p @ 60fps display is reading a 720x480 window of the 1920x1080 source video from the stored frame.

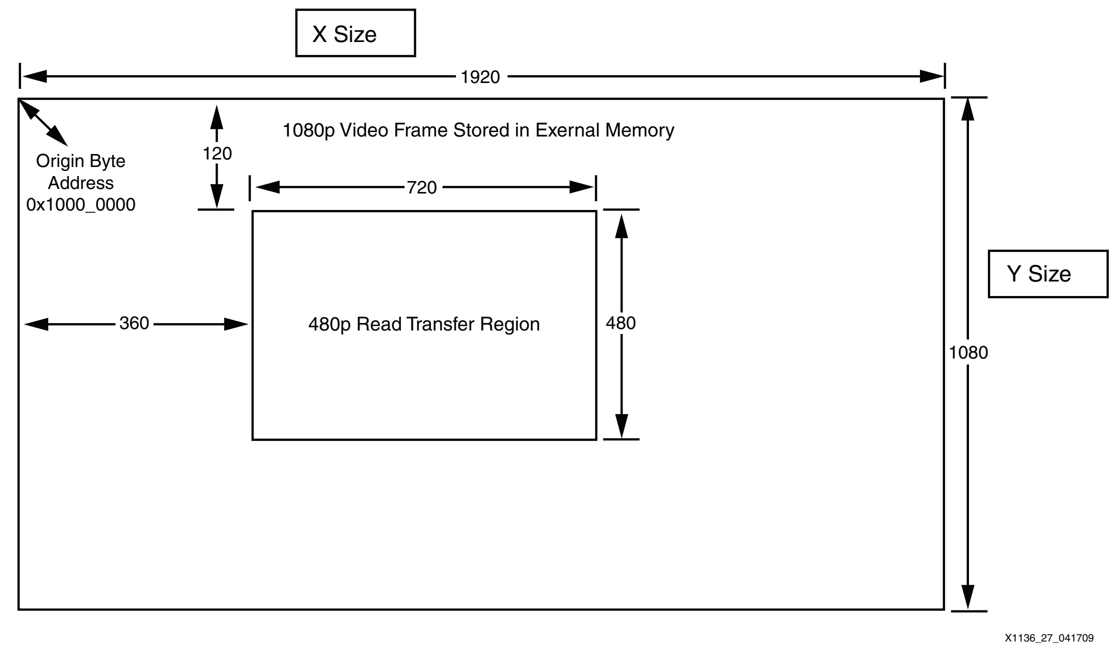

X Size
1920
Origin Byte Address 0x1000_0000
120
1080p Video Frame Stored in Exernal Memory
720
360
480p Read Transfer Region
480
Y Size
1080
X1136_27_041709

Figure 12: **Example VFBC Configuration**

In this example, the 1080p source video frame is stored at address 0x10000000. Assuming that each pixel is stored at 32-bits-per-pixel resolution, each pixel is 4 bytes. Because the X Size is stored in number of bytes, the X Size is 1920*4 (7680 bytes). This corresponds to the hexadecimal number of `0x1E00`.

In this case the X Size and the Stride (also stored in number of bytes) are the same value, `0x1E00`. The Y Size is stored as the number of lines minus one, 1080-1 (1079) or the hexadecimal number `0x437`. The following table shows the command packet for the 1080p source video. For example, this packet could be written to the VFBC during each source video blank interval.

Command Word 0		Command Word 1		Command Word 2		Command Word 3	
31:15 Reserved	14:0 X size	31 Write_NotRead	30:0 Start Addr.	31:24 Reserved	23:0 Y Size	31:24 Reserved	23:0 Stride
`0x0000_1E00`		`0x9000_0000`		`0x0000_0437`		`0x0000_1E00`	

The 480p display in this example expects to read a 720x480 portion of the 1080p source from the external memory starting from the 360th pixel on the 120th line. The X Size for the display is 720*4 (2880 bytes). This corresponds to the hexadecimal number `0xB40`. The Stride remains the same as the 1080p source command, `0x1E00` (or 1920*4) because the video is stored as a 1080p frame in external memory. The Y Size is 480 - 1 (479) or the hexadecimal number `0x1DF`.

The Start Address for the 480p video display includes the line and pixel offset information and is calculated by adding (120*1920 + 360)*4 to the origin or base address of the 1080p frame. This corresponds to a final start address of `0x100E_15A0`.

The following table shows the command packet for the 480p video display hardware.

Note: The Write_NotRead bit in Command Word 1 is now zero to denote a read transfer.

Command Word 0		Command Word 1		Command Word 2		Command Word 3	
31:15 Reserved	14:0 X size	31 Write_NotRead	30:0 Start Addr.	31:24 Reserved	23:0 Y Size	31:24 Reserved	23:0 Stride
`0x0000_0B40`		`0x100E_15A0`		`0x0000_01DF`		`0x0000_1E00`	

Several video functions can be performed by changing the Start Address within Command Word 1 during each video blank interval. For example:

- To cycle through multiple frame stores in external memory
- To perform a pan-scan on a 480p display of a rescaled 16:9 source when combined with a video scaler.

VFBC Transfer Commands

These commands are handled by both software and hardware though MicroBlaze, shared memory blocks and custom control logic.

Line Mode

The following example is of a 576p frame being read from external memory as individual lines. This example shows one transfer which is repeated for each line in the video. Each 576p video frame includes 576 line transfers. Each transfer has a different Start Address.

At the beginning of each line during the horizontal blank interval, the VFBC command interface and read interface must be reset for at least two clock cycles. Following the reset, the read command is written to the VFBC command interface. The VFBC read interface becomes non-empty several cycles following the command, and data can be popped off of the read interface FIFO.

Figure 13 shows a transfer for a single 576p line.

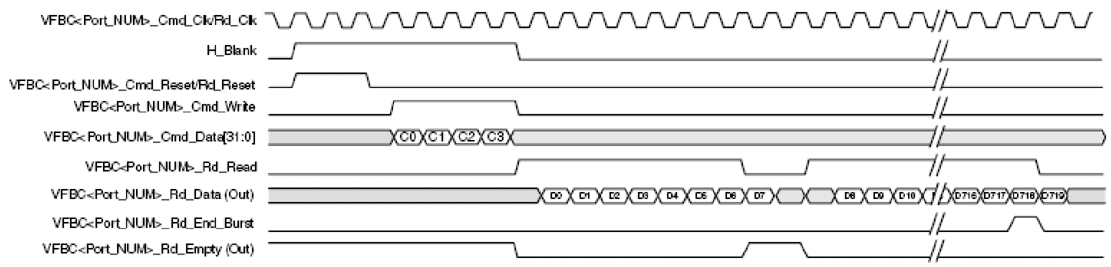

VFBC<Port_NUM>_Cmd_Clk/Rd_Clk
H_Blank
VFBC<Port_NUM>_Cmd_Reset/Rd_Reset
VFBC<Port_NUM>_Cmd_Write
VFBC<Port_NUM>_Cmd_Data[31:0]
VFBC<Port_NUM>_Rd_Read
VFBC<Port_NUM>_Rd_Data (Out)
VFBC<Port_NUM>_Rd_End_Burst
VFBC<Port_NUM>_Rd_Empty (Out)
C0
C1
C2
C3
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D716
D717
D718
D719

Figure 13: **Line Mode 576p Transfer**

Note: The `VFBC<Port_Num>_Rd_End_Burst` signal must be asserted on the last word transfer because 720 is not a multiple of the VFBC burst size of 32 words. In this example the assertion occurs on the 719th pixel read. The following table shows the command words written to the command interface during the horizontal blank interval for the transfer of the first 576p line. The X Size is 720*4 bytes. The Y Size must be zero (denoting a single line transfer), and the Stride is ignored and can be any value. This example shows the Stride set to zero. The next line transfer has a Start Address of 0x1000_00B40. Each subsequent line transfer Start Address increments by 0xB40 during each horizontal blank interval.

Command Word 0		Command Word 1		Command Word 2		Command Word 3	
31:15 Reserved	14:0 X size	31 Write_NotRead	30:0 Start Addr.	31:24 Reserved	23:0 Y Size	31:24 Reserved	23:0 Stride
0x0000_0B40		0x1000_000		0x0000_0000		0x0000_0000	

A Simple Interlacing and De-interlacing Example

The VFBC PIM can be used for simple video processing such as interlacing or de-interlacing. The following tables show the VFBC commands to write a 480i source into a 480p frame store. As with the 1080p to 480p example, the X Size is set to 720*4 (0xB40). The Y size is set to 240-1 (0xEF) because a 480i field contains 240 lines.

There is a different VFBC command for each top and bottom field. The data is interleaved into the frame store by configuring the Stride to be two line lengths (2880*2, or 0x1680) and offsetting the bottom field Start Address by 720*4 (0xB40).

Command Word 0		Command Word 1		Command Word 2		Command Word 3	
31:15 Reserved	14:0 X size	31 Write_NotRead	30:0 Start Addr.	31:24 Reserved	23:0 Y Size	31:24 Reserved	23:0 Stride
`0x0000_0B40`		`0x9000_000`		`0x0000_00EF`		`0x000_1680`	

The 480p command is similar to the 480i command packets, except the Stride is now a single 480p line, 720*4 (0xB40), the same as the X Size. The Y Size is set to 480-1 (0x1DF). The Start Address is the same start address as the first line of the 480i top field (0x1000_0000). The following table shows the 480p command packet.

Command Word 0		Command Word 1		Command Word 2		Command Word 3	
31:15 Reserved	14:0 X size	31 Write_NotRead	30:0 Start Addr.	31:24 Reserved	23:0 Y Size	31:24 Reserved	23:0 Stride
`0x0000_0B40`		`0x1000_000`		`0x0000_001DF`		`0x0000_0B40`	

Refer to the Xilinx Video Starter Kit website for more information about the VFBC transfers, the VFBC HDL interface, and example systems.

Appendix III: Design File Description

The following table provides design file descriptions.

File Name	Description
matlab_hwcs_script.m	MATLAB hardware co-simulation script
matlab_hwcs_script_top.m	MATLAB hardware co-simulation top-level script
vfbc.mdl	The main System Generator model
vfbc_hwcs.mdl	The System Generator model for hardware co-simulation
Sg_clk_config.m	The Blackbox configuration file for clock extraction
Simulink_hwcs_script.m	The Simulink based hardware co-simulation script
Stimulus_generator.m	The VFBC stimulus vector generator script
Edkprocessor_ddr2fix.m	The DDR2 IO pad workaround script
read420Header.m	The hardware co-simulation script for parsing the header text file

Appendix IV: Simulation Results

The following are sample runs for each frame size from the Hyper Terminal and MATLAB command window.

```
Begin Simulation.

Start Memory Test
Memory test successful!
Stop Memory Test

wrFrameHeight = 600
wrLIneLength = 800
rdFrameHeight = 600
rdLineLength = 800
numFrames = 1

Initializing done.
v3_c0 = 3200
v3_c1 = 90000000
v3_c2 = 599
v3_c3 = 3200
viFrameSize = 480000
Loading data .1
. done.
Storing data . . . done.

Simulation Done!
```

```
Begin Simulation.

Start Memory Test
Memory test successful!
Stop Memory Test

wrFrameHeight = 720
wrLIneLength = 1280
rdFrameHeight = 720
rdLineLength = 1280
numFrames = 1

Initializing done.
v3_c0 = 5120
v3_c1 = 90000000
v3_c2 = 719
v3_c3 = 5120
viFrameSize = 921600
Loading data . . . done.
Storing data . . . done.

Simulation Done!
```

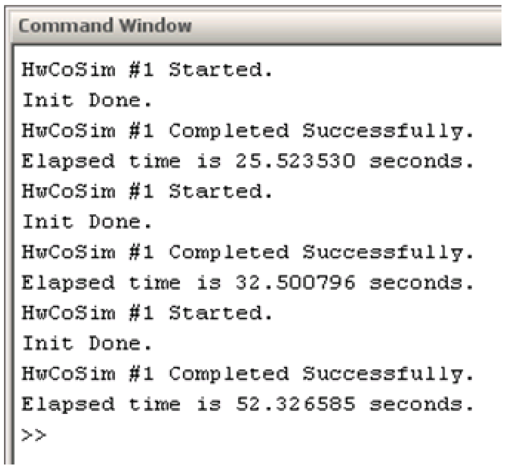
```
Begin Simulation.

Start Memory Test
Memory test successful!
Stop Memory Test

wrFrameHeight = 1080
wrLIneLength = 1920
rdFrameHeight = 1080
rdLineLength = 1920
numFrames = 1

Initializing done.
v3_c0 = 7680
v3_c1 = 90000000
v3_c2 = 1079
v3_c3 = 7680
viFrameSize = 2073600
Loading data . . . done.
Storing data . . . done.

Simulation Done!
```



```
HwCoSim #1 Started.
Init Done.
HwCoSim #1 Completed Successfully.
Elapsed time is 25.523530 seconds.
HwCoSim #1 Started.
Init Done.
HwCoSim #1 Completed Successfully.
Elapsed time is 32.500796 seconds.
HwCoSim #1 Started.
Init Done.
HwCoSim #1 Completed Successfully.
Elapsed time is 52.326585 seconds.
>>
```

Appendix V: Tool and IP Versions

The tool and IP versions used for this application note include the following:

- Xilinx ISE Design Suite 11.1 software
- System Generator 11.1
- MATLAB software R2008a

Appendix VI: Design Iterations between System Generator and XPS

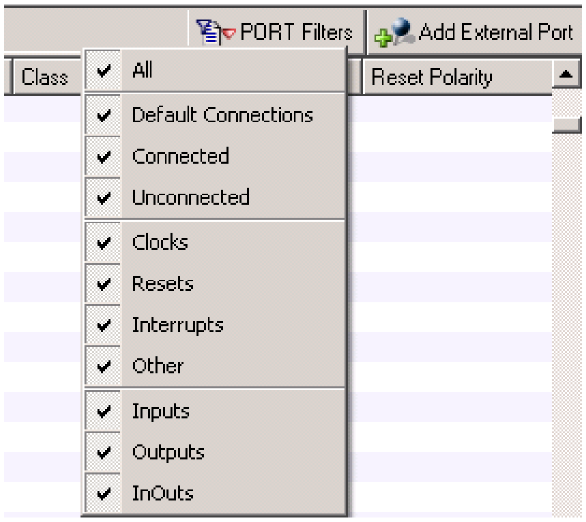
Appendix VI provides instructions for the following:

- "How to Add Multiple Nets to External Ports Using XPS GUI"
- "How to Change the MPMC Version from 4.02.a to 4.00.a"

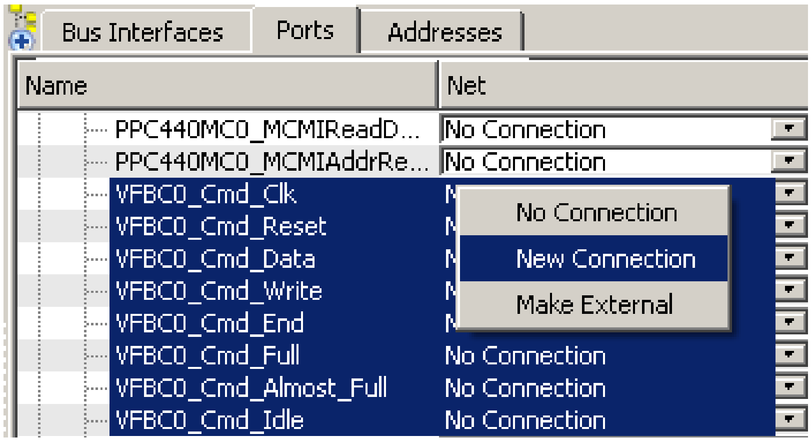
How to Add Multiple Nets to External Ports Using XPS GUI

For this application, the DDR2-SDRAM peripheral needs to be configured by bringing up VFBC3 and VFBC2 internal nets to the top-level design as external ports. By doing this, these ports can later be exposed by System Generator during importing as described in "Importing the Basic Processor System into System Generator," page 7. These exposed top-level ports can then be connected directly with other portions of the System Generator model. It is through the exposed top-level ports that a System Generator model interacts with the VFBC IP core.

1. Select **All** for Port Filters on the top right-hand corner.



2. Select multiple internal nets from DDR2_SDRAM peripheral, right-click, and select **Make External**.
3. Examine for correct net names and ports being brought up to the top-level as external ports.

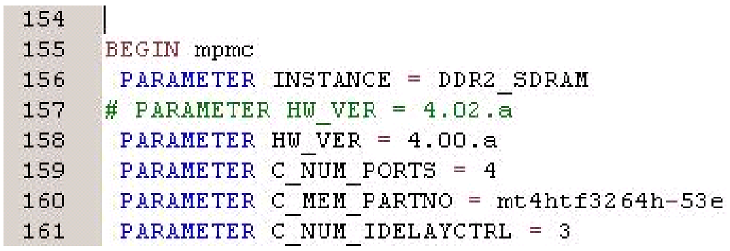


Name	Net
PPC440MC0_MCMIReadD...	No Connection
PPC440MC0_MCMIAddrRe...	No Connection
VFBC0_Cmd_Clk	N
VFBC0_Cmd_Reset	N
VFBC0_Cmd_Data	N
VFBC0_Cmd_Write	N
VFBC0_Cmd_End	N
VFBC0_Cmd_Full	No Connection
VFBC0_Cmd_Almost_Full	No Connection
VFBC0_Cmd_Idle	No Connection

How to Change the MPMC Version from 4.02.a to 4.00.a

After the design flows between System Generator and XPS are complete, certain peripherals in the current XPS base system may be changed or modified. This task is accomplished by either using the XPS GUI or directly editing the **MHS** file. In certain cases, it may be easier to make a direct edit to the **MHS** file as shown in the following example steps.

1. In XPS, open system.mhs -> modify it as shown:



```
BEGIN mpmc
 PARAMETER INSTANCE = DDR2_SDRAM
# PARAMETER HW_VER = 4.02.a
 PARAMETER HW_VER = 4.00.a
 PARAMETER C_NUM_PORTS = 4
 PARAMETER C_MEM_PARTNO = mt4htf3264h-53e
 PARAMETER C_NUM_IDELAYCTRL = 3
```

2. Hardware -> Clean Hardware.
3. Save Project.
4. Back in System Generator, repeat the procedure in "Importing the Basic Processor System into System Generator," page 7, by re-importing the XPS project.

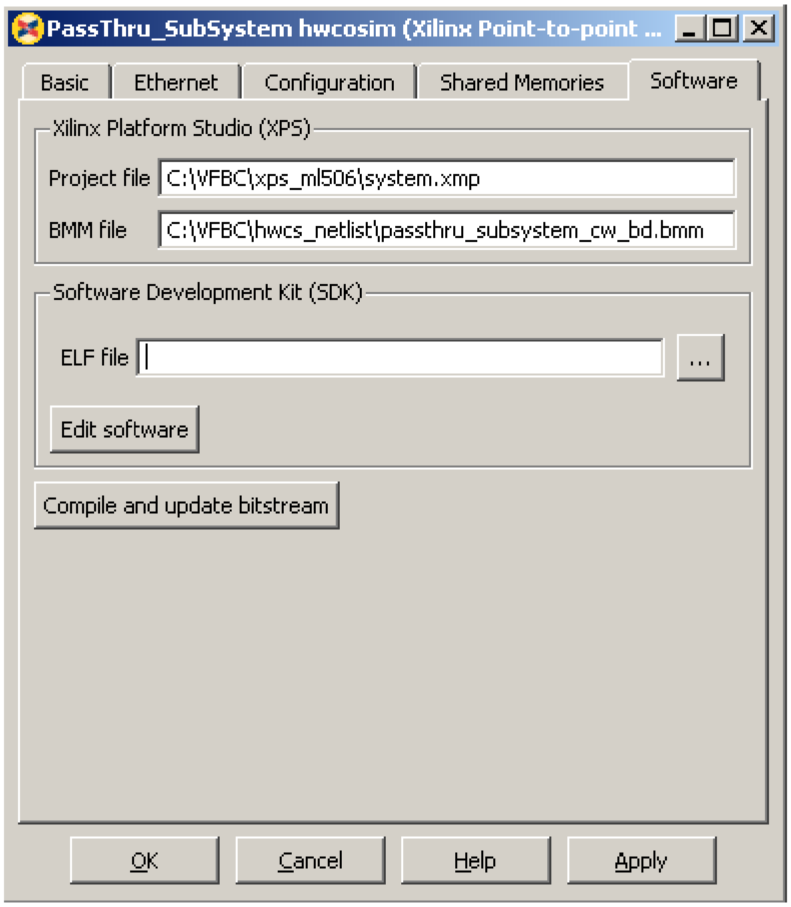
Appendix VII: How to Migrate from XPS to SDK Design Flows

Adding and managing software applications in XPS is deprecated starting in release 11.1 and will become obsolete starting in release 12.1. The existing XPS project provided by this application note may still continue to be used until 12.1, but Xilinx highly recommends migrating the software portion to the SDK flow to take advantage of flexibility and advanced features in adding and managing software. Follow the step-by-step instructions that follow to migrate from an existing XPS flow to an SDK design flow. The same generic steps can also be used on other Sysgen/EDK design demos to accomplish the same migration paths between XPS and SDK.

How to Migrate to the SDK Flow

1. Open the System Generator model with the hardware co-sim block (vfbc_hwcs.mdl), double-click on the HWCS block, and click on the **Edit software** button to launch SDK (Software Development Kits).

Note: The ELF file does not have to be entered at this point.


PassThru_SubSystem hwcosim (Xilinx Point-to-point ...
Basic
Ethernet
Configuration
Shared Memories
Software
Edit software
Compile and update bitstream
OK
Cancel
Help
Apply

Xilinx Platform Studio (XPS)

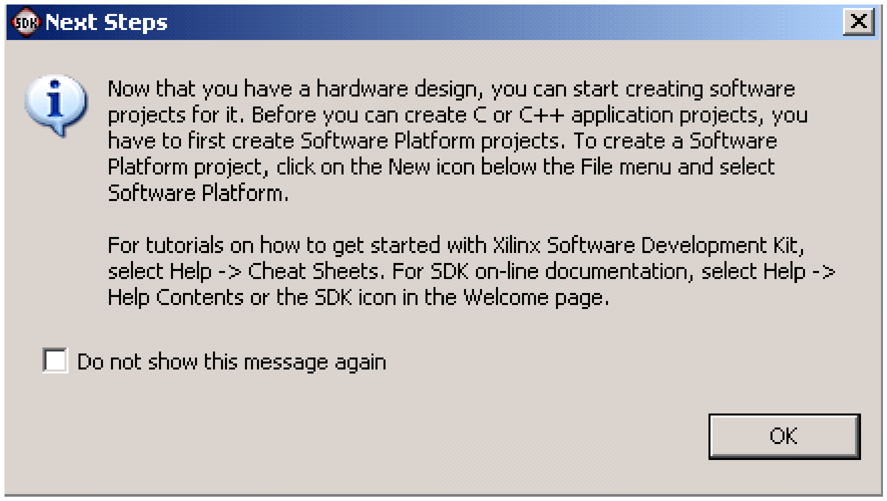
Project file C:\VFBC\xps_ml506\system.xmp

BMM file C:\VFBC\hwcs_netlist\passthru_subsystem_cw_bd.bmm

Software Development Kit (SDK)

ELF file

2. Click **OK** to start creating the software project.

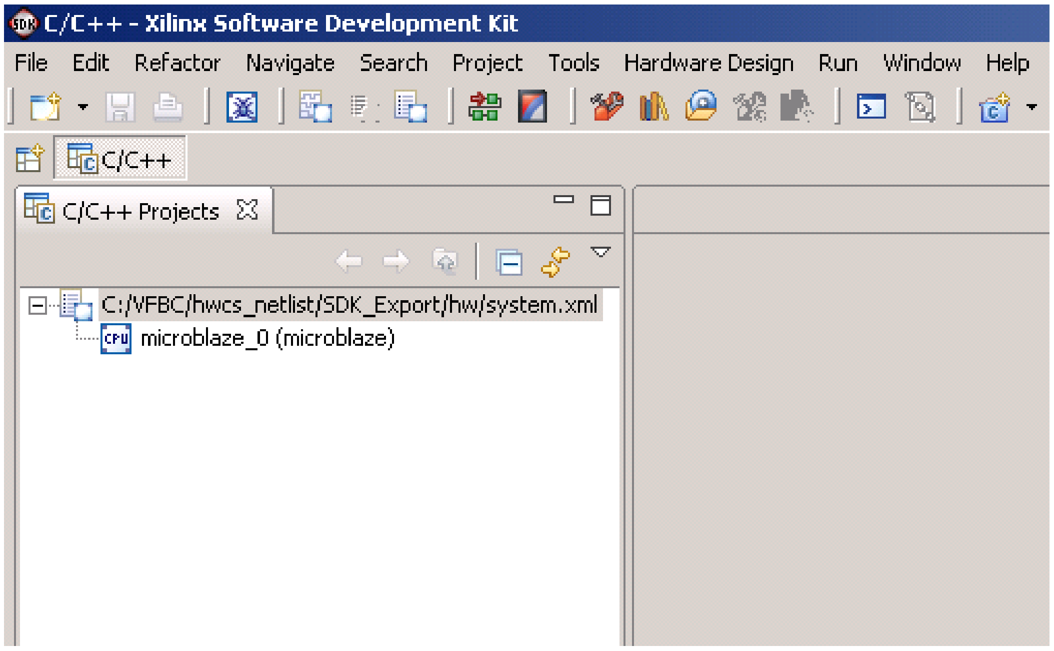

Next Steps

Now that you have a hardware design, you can start creating software projects for it. Before you can create C or C++ application projects, you have to first create Software Platform projects. To create a Software Platform project, click on the New icon below the File menu and select Software Platform.

For tutorials on how to get started with Xilinx Software Development Kit, select Help -> Cheat Sheets. For SDK on-line documentation, select Help -> Help Contents or the SDK icon in the Welcome page.

☐ Do not show this message again

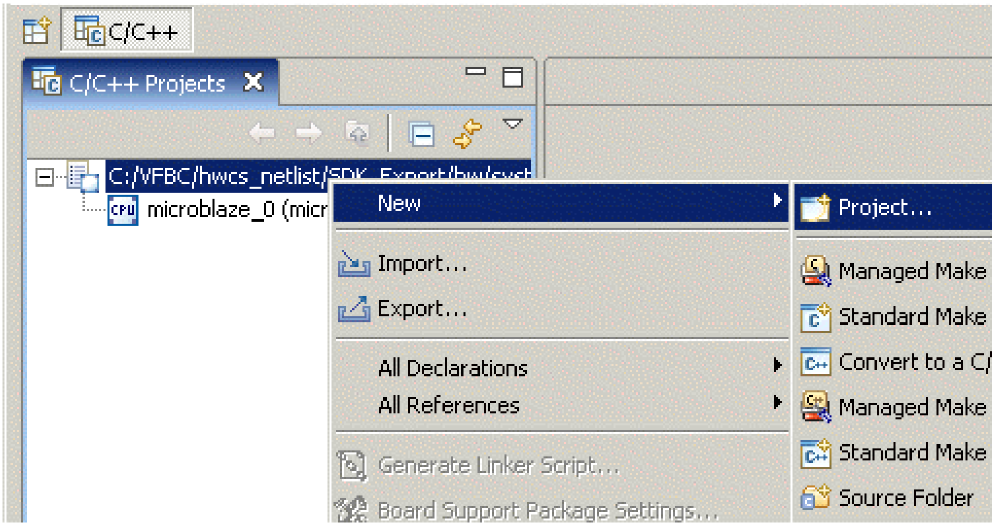
OK


C/C++ - Xilinx Software Development Kit

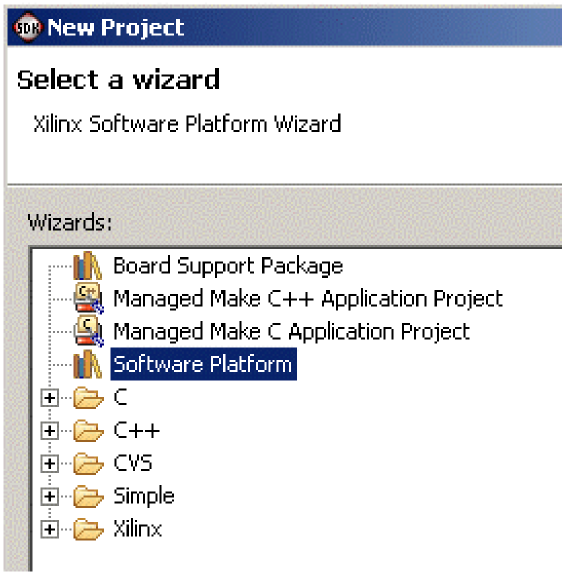
File Edit Refactor Navigate Search Project Tools Hardware Design Run Window Help
C/C++
C/C++ Projects

C:/VFBC/hwcs_netlist/SDK_Export/hw/system.xml
microblaze_0 (microblaze)

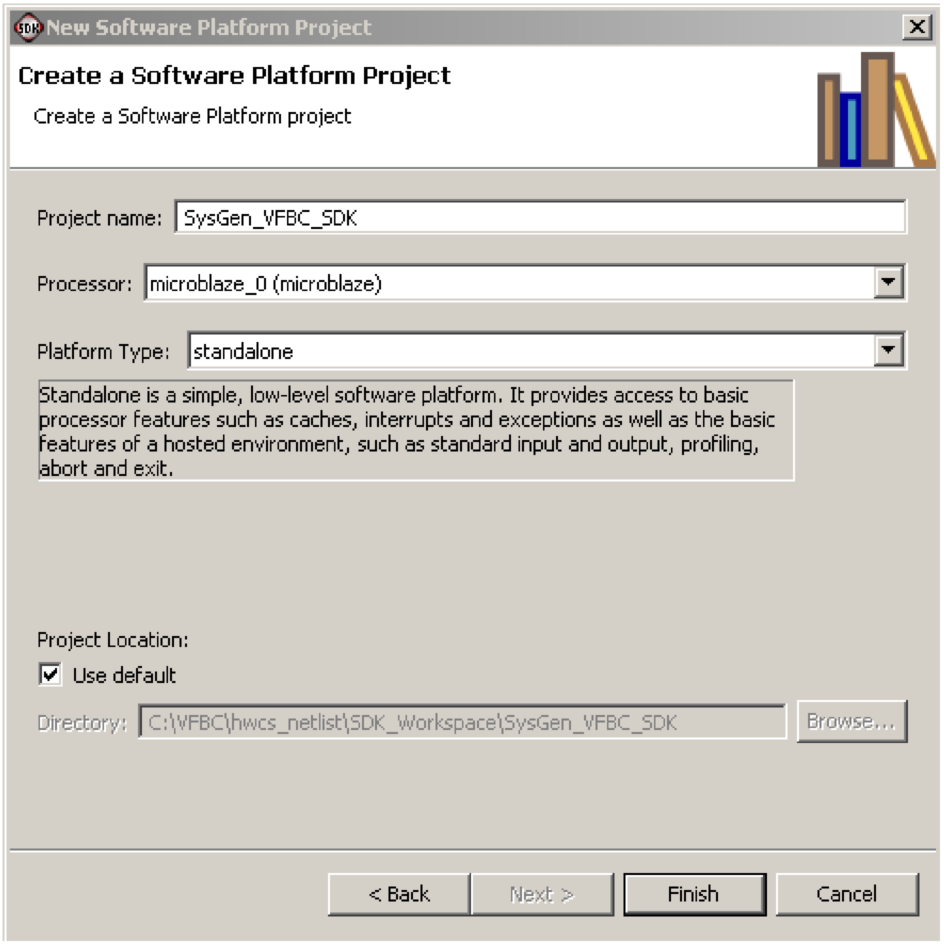
3. Right-click on the **system.xml SDK project** → **New** → **Project**.



4. Select **Software Platform** (Wizard).


New Project
Select a wizard
Xilinx Software Platform Wizard
Wizards:
Board Support Package
Managed Make C++ Application Project
Managed Make C Application Project
Software Platform
C
C++
CVS
Simple
Xilinx

5. Enter the project name and click **Finish**.



New Software Platform Project

Create a Software Platform Project

Create a Software Platform project

Project name: SysGen_VFBC_SDK

Processor: microblaze_0 (microblaze)

Platform Type: standalone

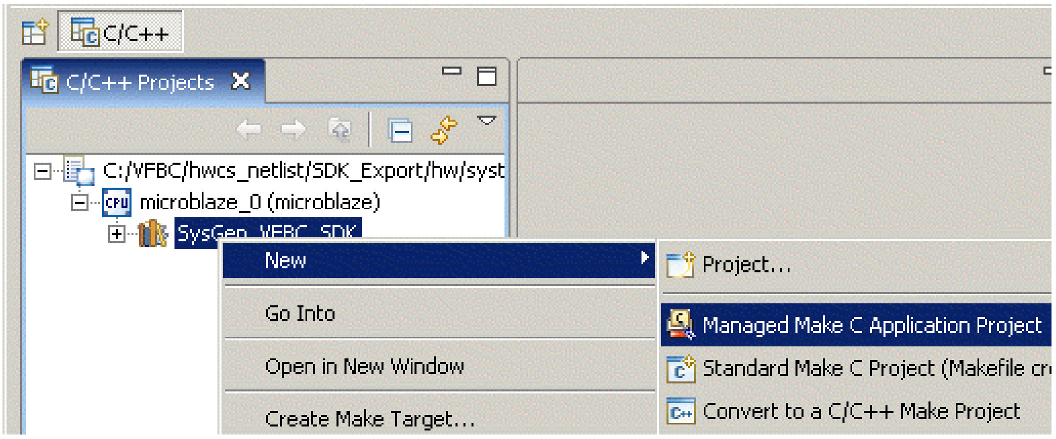
Standalone is a simple, low-level software platform. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Project Location:

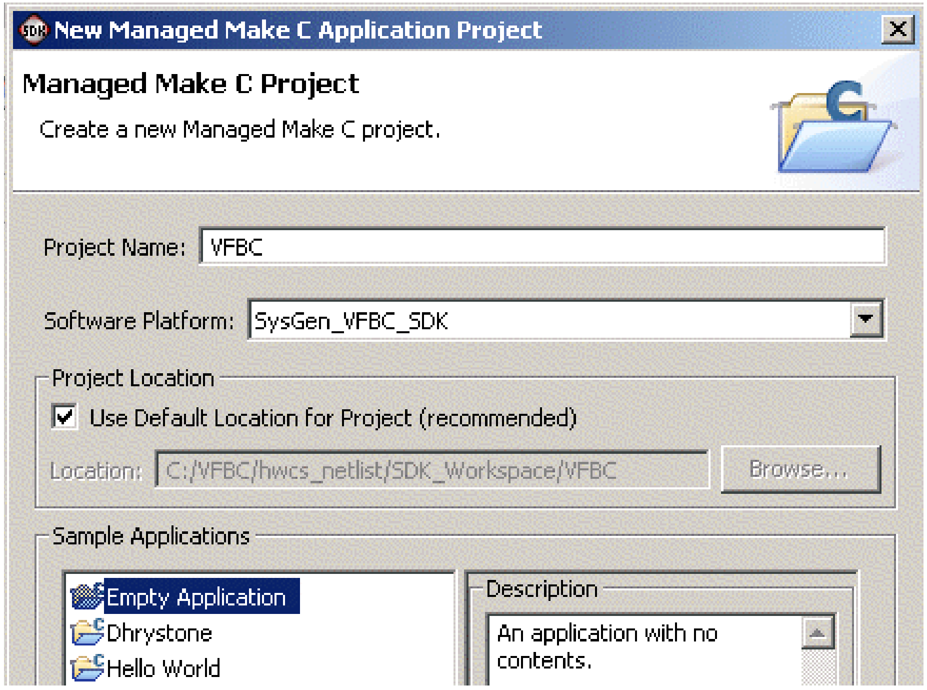
☑ Use default

Directory: C:\VFBC\hwcs_netlist\SDK_Workspace\SysGen_VFBC_SDK

6. Right-click on **SysGen_VFBC** → **New** → **Manage Make C Application Project**


C/C++
C/C++ Projects
C:/VFBC/hwcs_netlist/SDK_Export/hw/syst
microblaze_0 (microblaze)
SysGen_VFBC_SDK
New
Go Into
Open in New Window
Create Make Target...
Project...
Managed Make C Application Project
Standard Make C Project (Makefile cr
Convert to a C/C++ Make Project

7. Enter the Software Platform Project name, select **Empty Application** with no template, and click **Finish**.



Project Name: VFBC

Software Platform: SysGen_VFBC_SDK

Project Location

[✓] Use Default Location for Project (recommended)

Location: C:/VFBC/hwcs_netlist/SDK_Workspace/VFBC

An application with no contents.

The SDK design cockpit should look similar to that shown in Figure 14.

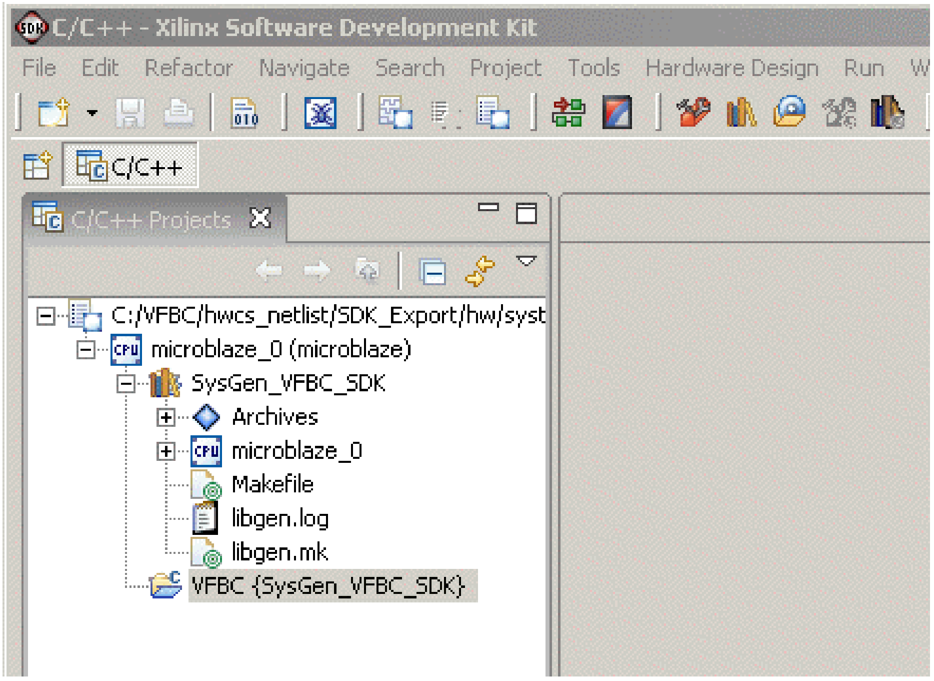


Figure 14: **SDK Design Cockpit (example)**

8. The last step is to either create a new C code source file or add an existing one to the project. In this case, just add the existing one from C:\VFBC\C-code\vfbc.c.

   The easiest way to add a C code source file to the VFBC (SysGen_VFBC) application project is to simple Copy & Paste or Drag & Drop the file into the project. Once the file is added, the project will be built and compiled automatically.

How to Iterate the Design between System Generator and SDK

By default, the ELF file is named after the application project name – **VFBC**.elf in this case. It is also located under the folder of the application project location.

Design iteration using the SDK is similar to that performed in XPS, but with more advanced features and functionalities. Once the C code is modified and saved, the software application will be rebuilt and recompiled automatically. This, in turn, generates a new ELF file that will then be used by System Generator to recompile and update the bitstream to the target board.

SW Iteration

1. SDK – Modify C code and make sure the software project is recompiled successfully.
2. SysGen – Click Compile and Update bitstream button.
3. SysGen – Simulate the design.

HW Iteration

1. SDK – Add new peripherals or modify existing ones.
2. SysGen – Re-import an XPS project into the SysGen design.
3. SysGen – Re-generate a HWCS block.
4. SDK – Modify C code accordingly.
5. SysGen – Re-simulate the design.

Note: Making HW changes requires a new design implementation through Place & Route.

Revision History

Date	Version	Description of Revisions
06/01/09	1.0	Initial Xilinx release.

Notice of Disclaimer