Lab 3 - PowerPC Processor

Adding Custom IP to an Embedded System Lab:



Creating and Adding Custom IP to an Embedded System Lab: PowerPC Processor

Introduction

This lab guides you through the process of creating and adding a custom OPB peripheral to a processor system by using the Create and Import Peripheral Wizard.

Objectives

After completing this lab, you will be able to:

- Create an IP
- Add the custom IP to a real system, develop an application, and generate a bitstream
- Download the bitstream and verify the functionality in an actual hardware

Procedure

The purpose of this lab exercise is to complete the hardware design started in Lab 1 and extended in Lab 2. Lab 1 included the PPC, PLB bus, JTAG_PPC, proc_Sys_Reset, DCM, PLB2OPB, RS232_Uart_1, PLB RAM controller, and PLB BRAM components. Lab 2 added the remaining IP, except for an MYIP instance for the LED, to extend the hardware design.



In this lab, you will use the Create and Import Peripheral Wizard of Xilinx Platform Studio (XPS) to create a user peripheral from an HDL module, add an instance of the imported peripheral, and modify the system.ucf file to provide an interface to the on-board LED module.



Figure 3-1. Completed Design

This lab comprises several steps involving the creation of OPB custom IP (a simple 4-bit output to drive LEDs) and addition of a custom OPB peripheral. Although the change to the hardware is simple, this lab illustrates the integration of a user peripheral through the Create and Import Peripheral Wizard. This lab also illustrates the use of an existing peripheral to provide the OPB bus interface.

Below each general instruction for a given procedure, you will find accompanying step-by-step directions and illustrated figures providing more detail for performing the general instruction. If you feel confident about a specific instruction, feel free to skip the step-by-step directions and move on to the next general instruction in the procedure.



Creating a Custom IP

Step 1

- Create a *lab3* folder and copy the contents of the *lab2* folder into the *lab3* folder if you wish to continue with the design you created in the previous lab. Launch the **Create Import Peripheral** wizard. Name the peripheral as **my_led** and let it be for OPB bus.
 - If you wish to continue using the design that you created in Lab 2, create a *lab3* folder in the *C:\xup\embedded\ppc\labs* directory and copy the contents from *lab2* to *lab3*
 - Open XPS by clicking Start → Programs → Xilinx Platform Studio 7.1i → Create-Import Peripheral
 - Click Next to continue
 - In the Select Flow panel, select Create templates for a new peripheral and click Next
 - In the Repository or Project panel, select To an existing XPS project, browse to C:\xup\embedded\ppc\labs\lab3 and click Next

Reposit	ory or Project			Sec. 1
Indicat	e where you want to store the new peripheral.			
A new periphe	aral can be stored in an EDK repository, or in an XI	PS project	. When stored in a	an
DK repositor	y the peripheral can be accessed by multiple XPS	projects.		
	IK user repositoru. (Apu directoru outside of uour E	DK install:	ation nath)	
Density	w Any directory bacside or your c		auon paunj	
Heposito	φ	<u> </u>	blowse	
To an XP	'S project			
 To an XF Project 	S project C:\xup\embedded\labs\lab3\system.xmp	•	Browse	
To an XF Project	S project C:\xupl\embedded\labs\lab3\system.xmp		Browse	
To an XF Project Peripheral wi	S project C:\xup\embedded\labs\lab3\system.xmp	<u>.</u>	Browse	
 To an XF Project Peripheral wi C:\xup\e 	S project C:\xup\embedded\labs\lab3\system.xmp II be placed under: mbedded\labs\lab3\pcores		Browse	
To an XF Project Peripheral wi C:\xup\e	S project C:\xup\embedded\labs\lab3\system.xmp II be placed under: mbedded\labs\lab3\pcores	•	Browse	
To an XF Project Peripheral wi C:\xup\e	S project C:\xupl\embedded\labs\lab3\system.xmp II be placed under: mbedded\labs\lab3\pcores		Browse	

Figure 3-2. Repository or Project Dialog Box

• In the Name and Version panel, enter my_led as the peripheral name, accept default versions, and click Next



Create Peripheral - Step 1
Name and Version Indicate the name and version of your peripheral.
Enter the name of your peripheral. This name will be used as the top level HDL module. Name: my_led
Version: 1.00.a
Major Revision Minor Revision Hardware/Software Compatibility Revision
Logical library name: my_led_v1_00_a All HDL files (either created by you or generated by this tool) used to implement this peripheral must be compiled into the logical library named above. Any other logical libraries referred to in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK repositories indicated in the XPS project settings.
More Info

Figure 3-3. Name and Version Dialog Box

• In the Bus Interfaces panel, select On-chip Peripheral Bus (OPB), and click Next

Create Peripheral - Ste	p 2			
Bus Interface Indicate the bus interl	ace supported by you	ır peripheral.		and the second
To which bus will this periph	eral be attached?			
 On-chip Peripheral Bu Processor Local Bus 	s (OPB) PLB)			
Fast Simplex Link (FS	L)			
NOTE: Other bus interfaces	are not supported in I	this release.		
More Info	< <u>B</u> ack	<u>N</u> ext >	Finish	Cancel

Figure 3-4. Bus Interface Dialog Box

BFM Simulation Lab





Continuing with the wizard, select **RST/MIR** and **User Logic S/W Register** support. Select only **one** software accessible register of **32**-bit width. Generate template driver files. Browse to the **C:\xup\embedded\ppc\labs\lab3** directory and answer the questions at the end of this step

• In the IPIF Services panel, select S/W Reset and Module Information register (RST/MIR) and User Logic S/W Register Support

IPIF Services Indicate the IPIF services required by your perip	heral.
Your peripheral will be connected to the OPB bus thro Besides standard functions like address decoding, this services. Using these services may significantly simplif	ugh the OPB IP interface (IPIF) module. s module also offers other commonly used y the implementation of your peripheral.
Slave Attachment Burst Byte Steering RST Write FIFO Read FIFO	 S/W Reset and Module Information register (RST/MIR) Burst Transaction Support DMA FIFO User Logic Interrupt Support User Logic S/W Register Support User Logic Master Support
	User Logic Address Hange Support

Figure 3-5. IPIF Services Dialog Box

- Click Next
- In the User S/W Register panel, click Next to accept the default values as will have only one register that will control the LEDs



Create Peripheral - Step 4 🛛 🔀
User S/W Register Configure the software accessible registers in your peripheral.
The software accessible registers will be implemented in the user-logic module of your core. These registers are addressable on the byte, half-word or word boundaries. The following fields determine the characteristics of the registers.
Number of software accessible registers:
Data width of each register: 32
More Info < <u>B</u> ack <u>N</u> ext > Finish Cancel

Figure 3-6. User S/W Register Dialog Box

• Scroll through the **IP Interconnect (IPIC)** panel, which displays the default IPIC signal which are available for the user logic based on the previous selection, click **Next**

IP Interconnect (I	PIC)	1
Select the interface the IPIF.	between the logic to be implemente	ed in your peripheral and
our peripheral is connect terfaces to the IPIF throu f the ports are always pre- inctionality required by yo	ed to the bus through a suitable IPI igh a set of signals called the IP inte sent. You can choose to include th jur peripheral.	F module. Your peripheral erconnect (IPIC) interface. Some e others based on the
	Note: all IPIC ports are active high	h
OPB or PLB bus	□IP2Bus_Clk ▲ ✓Bus2IP_Clk	Port Description
IPIF	Bus2IP_Reset Bus2IP_Freeze Bus2IP_Addr	
IPIC	✓Bus2IP_Data	
User Logic	Bus2IP_BE	<u> </u>
- N. 177		
	10 C	1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.

Figure 3-7. IP Interconnect (IPIC) Dialog Box



• In the (OPTIONAL) Peripheral Simulation Support panel, uncheck Generate BFM simulation platform inorder not to generate the BFM simulation associated files and directories, and click Next

The EDK provides a BFM simulation platform to he want this tool to generate the appropriate HDL and for the target bus.	lp you simulate your peripheral. Indicate if you d Bus Functional Language (BFL) stimulus file
OPB Device (masker) OPB Monitor OPB Monitor Mittany United Vice of the set BFM Synch Bus	Generate BFM simulation platform for ModelSim NOTE: This feature requires that you have accepted the associated IBM license agreement and installed the BFM toolkit. The link below shows how: BFM Toolkit Installation Instructions

Figure 3-8. Peripheral Simulation Support Dialog Box

In the (OPTIONAL) Peripheral Implementation Options panel, uncheck Generate ISE and XST project files to help you not to implement the peripheral using XST flow and check Generate template driver files to help you to implement software interface

Generate optional files for	r hardware/software implementation
Upon completion, this tool will cr you requested. A stub 'user_logii implementation of this module us interface files (mpd/pao) for the peripheral to a processor system	eate synthesizable HDL files that implement the IPIF services c' module will be created. You will need to complete the ing standard HDL design flows. The tool will also generate EDK synthesizable templates, so that you can hook up the generated
Peripheral (VHDL)	NOTE: Should the peripheral interface (ports/parameters) or file list change, you will need to regenerate the EDK interface files using the import functionality of this tool.
IPIF (VHDL)	Generate stub 'user_logic' template in Verilog instead of VHDL.
	Generate ISE and XST project files to help you implement the peripheral using XST flow.

Figure 3-9. Peripheral Implementation Options Dialog Box

BFM Simulation Lab

www.xilinx.com/univ xup@xilinx.com



Create and Import Periphera	l Wizard	×
	Congratulations!	
NA C	When you click Finish, HDL files representing your core will be generated. You will have to implement the functionality of your peripheral in user_logic.vhd.	
	IMPORTANT: If you make any changes to the generated port and parameter interfaces, or add new files you will need to regenerate the EDK interface files by using this tool in the Import mode.	
	Thank you for using Create and Import Peripheral Wisard! Peripheral summary	
	top name : my_led version : 1.00.a type : OPB slave features : slave attachement	
More Info	<u>≺B</u> ack <u>N</u> ext > Finish Cance	

• Click Next, and you will see the summary information panel

Figure 3-10. Congratulations Dialog Box

3 Click **Finish** to close the wizard



Add LED port in my_led_v2_1_0.mpd file generated by the wizard in C:\xup\embedded\ppc\labs\lab3\pcores\my_led_v1_00_a\data directory

- Using Windows Explorer, browse to C:\xup\embedded\ppc\labs\lab3\pcores\my_led_v1_00_a\data directory
- Open my_led_v2_1_0.mpd file using an editor
- Add the following line before the **OPB_Clk** port under the **Ports** section

PORT LED = " ", DIR = O, VEC = [0:3]

This is necessary for the port to appear in Add/Edit Cores... (Dialog)

4 Save the file and close





Open the my_led.vhd and user_logic.vhd files in the Text Editor window from C:\XUP\embedded\ppc\labs\lab3\pcores\my_led_v1_00_a\hdl\vhdl directory. Add necessary declarations and logic in my_led.vhd and user_logic.vhd files.

- Browse to C:\xup\embedded\labs\lab3\pcores\my_led_v1_00_a\hdl\vhdl directory.
- Right click on **my_led.vhd** file and open it with **text editor** program.
- Add user port LED under USER ports added here token

1	109📫	entity my	led is
	110	gene	ric
	111	(
	112		ADD USER GENERICS BELOW THIS LINE
	113		USER generics added here
	114		ADD USER GENERICS ABOVE THIS LINE
	115		
	116		DO NOT EDIT BELOW THIS LINE
	117		Bus protocol parameters, do not add to or delete
	118		C_BASEADDR : std_logic_vector := X"00000000";
	119		C_HIGHADDR : std_logic_vector := X"0000FFFF";
	120		C_OPB_AWIDTH : integer := 32;
	121		C_OPB_DWIDTH : integer := 32;
	122		C_USER_ID_CODE : integer := 3;
	123		C_FAMILY : string := "virtex2p"
	124		DO NOT EDIT ABOVE THIS LINE
	125);	
	126	port	
	127	(
	128		ADD USER PORTS BELOW THIS LINE
	129	-	USER ports added here
	130		LED : out std_logic_vector (O to 3);
	131	L	
	132		ADD HISER PORTS ABOVE THIS LINE

Figure 3-11. Add the User Port LED

• Search for next --USER and add port mapping statement

393	
394	instantiate the User Logic
395	
396	USER_LOGIC_I : entity my_led_v1_00_a.user_logic
397	generic map
398	(
399	MAP USER GENERICS BELOW THIS LINE
400	USER generics mapped here
401	MAP USER GENERICS ABOVE THIS LINE
402	
403	C_DWIDTH => USER_DWIDTH,
404	C_NUM_CE => USER_NUM_CE
405)
406	port map
407	(
408	MAP USER PORTS BELOW THIS LINE
409	USER ports mapped here
410	LED => LED,
411	MAP USER PORTS ABOVE THIS LINE

Figure 3-12. Add Port Mapping Statement



• Open user_logic.vhd file from *vhdl* directory and add LED port definition in the USER Ports area

92⇔ entit	ty us	er_logic is
93	gene	ric
94	(
95		ADD USER GENERICS BELOW THIS LINE
96		USER generics added here
97		ADD USER GENERICS ABOVE THIS LINE
98		
99		DO NOT EDIT BELOW THIS LINE
100		Bus protocol parameters, do not add to or delete
101		C_DWIDTH : integer := 32;
102		C_NUM_CE : integer := 1
103		DO NOT EDIT ABOVE THIS LINE
104);	
105	port	
106	(
107		ADD USER PORTS BELOW THIS LINE
108	-	USER ports added here
109		LED : out std_logic_vector (O to 3);
110		ADD USER PORTS ABOVE THIS LINE

Figure 3-13. Add the LED Port Definition

• Search for next --USER and add the internal signal declaration for the user logic

123	IP2Bus Error : out std logic;
124	IP2Bus_ToutSup : out std logic
125	DO NOT EDIT ABOVE THIS LINE
126);
127	end entity user_logic;
128	_
129	
130	Architecture section
131	
132	
133	architecture IMP of user_logic is
134	
135	USER signal declarations added here, as needed for user logic
136	signal LED_i : std logic vector (0 to 3);
137	·································
138	Signals for user logic slave model s/w accessible register example
139	
140	<pre>signal slv_reg0 : std_logic_vector(0 to C_DWIDTH-1);</pre>
141	<pre>signal slv_reg_write_select : std_logic_vector(0 to 0);</pre>



Search for -USER logic implementation and add the following code



Figure 3-15. Add Code

• Save changes and close the **my_led-imp**

Importing and Adding Custom IP to the Project Step 2

Browse to C:\xup\embedded\ppc\labs\lab3 and open system.xmp. Using Tools → Create/Import Peripheral, import the created my_led custom IP to the project

- Browse to C:\xup\embedded\ppc\labs\lab3 and double-click on system.xmp. Open up the Create and Import Peripheral Wizard by selecting Tools → Create/Import Peripheral... from XPS menu and click Next to continue
- In the Select Flow panel, select the Import existing peripheral mode and click Next



Indicate if you wa	nt to create a new peripheral or import an existing peripheral.
is tool will help you cr isting peripheral into a uctures required by El	eate templates for a new EDK compliant peripheral, or help you import an n XPS project or EDK repository. The interface files and directory DK will be generated.
-	Select Flow
reate Templates	Create templates for a new peripheral
	52 • Import existing peripheral
mplement/Verify	Flow Description
Import to XPS	This tool will help you import a fully implemented peripheral into a XPS project or EDK repository. Such peripherals need to have ports and parameters that conform to the conventions required by EDK.

Figure 3-16. Create/Import User Peripheral Dialog Box

• In the **Repository or Project** panel, choose to import To an existing XPS project and select your current project from the drop down list c:\xup\embedded\ppc\labs\lab3\system.xmp and click Next



reate and Impo	rt Peripheral \	Wizard			×
Repository Indicate wh	v or Project here you want to	store the new p	eripheral.		1
A new peripheral EDK repository th	can be stored in e peripheral can	an EDK repositi be accessed by	ory, or in an XPS p v multiple XPS pro	project. When stored jects.	d in an
C To an EDK us Benository	ser repository (A	ny directory outs	ide of your EDK in	nstallation path)	ì
• To an existing	3XPS project		-		1
Project	C:\training\embe	edded\labs\lab3	\system.xmp	Browse]
Peripheral will be C:\training\er	written under: mbedded\labs\la	ab3\pcores			
More Info		< Back	Next >	Finish	Cancel

Figure 3-17. Repository or Project Dialog Box

In the Core Name and Version panel, select my_led from the drop down list box, check Use Version and accept 1.00.a, and click Next

perip	heral version	naming s	cheme.		ste il you a	re using t	NELDK	
Enter th	e name of th	e top VHD	L entity or	Verilog m	odule of y	our periph	ieral.	
Nam	e my_led			•				
Logica All th the p the c settir librari	al library nam e files for this eripheral refe urrent projec igs. Since all es other that	e: my_led_ s periphera ars to othe t or in the design filk n given ab	_v1_00_a _v1_00_a r logical lib repositorie es are com over may c d.	a biled into t raries, the s access piled in th ause nar	he logical l ay are assu ble through e same dir he space c	ibrary nai med to b n the curr ectory, u: onflicts, 1	ned above e available ent project sing logical .ogical libra	. If in aries

Figure 3-18. Core Name and Version Dialog Box



• In the Source File Types panel, check HDL Source Files(*.vhd, *.v), and click Next

Import Peripheral - Step 2	
Source File Types Indicate the types of files that make up your peripheral.	- Children
Indicate the types of files that make up your peripheral. HDL Source Files (*.vhd, *.v) Netlist Files (*.edn, *.edf, *.ngo, *.ngc) Documentation Files (*.doc, *.txt, *.pdf, *)	
More Info KBack Next > Finish	Cancel

Figure 3-19. Source File Types Dialog Box

In the HDL Source Files panel, select Use existing Peripheral Analysis Order file (*.pao) as the way to locate the HDL source files, specify the PAO file generated in the create mode by using the Browse button and browsing to

c:\xup\embedded\ppc\labs\lab3\pcores\my_led_v1_00_a\data\my_led_v2_1_0.pao, and click Next

what HDL is y	our peripheral implemented?	VHDL	-
Use data (*.n	npd) collected during a previous	invocation of this t	tool
[Browse
C Use an X This tool (from the a	ST project file (*.prj) will input the HDL file-set and the ippropriate lines in the project file	 logical libraries the 	ey are compiled into Browse
C Use an X This tool of from the a	ST project file (*.prj) will input the HDL file-set and the appropriate lines in the project file	: logical libraries the a.	ey are compiled into
C Use an X This tool y from the a	ST project file (*.prj) will input the HDL file-set and the appropriate lines in the project file ng Peripheral Analysis Order file g\embedded\labs\lab3\MyProc	s logical libraries th s. (*.pao) :essorIPLib\pcores	ey are compiled into Browse

Figure 3-20. HDL Source Files Dialog Box



- The HDL Analysis Information panel shows you all the dependent library files and HDL source files to compile your peripheral, as well as corresponding logical libraries those files will be compiled into. For this custom peripheral, wizard based on the PAO file automatically intuits all files needed. Click Next to continue
- In the **Bus Interfaces** panel, check the **OPB Slave** (**SOPB**) bus interface
- Click **Next** until the **Identify Interrupt Signals** panel is reached, select No Interrupt, and click **Next** to continue

Import Peripheral - Step 8	×
Identify Interrupt Sign Identify the interrupt signal:	als s on your peripheral.
Indicate the attributes of the inter then clicking on the radio button: connect the interrupt ports of you	rrupt signals by checking the interrupt port name on the left and s to the right. EDK uses this information to automatically ur peripheral. Properties of interrupt port: Interrupt Sensitivity C Falling edge sensitive
	Relative Interrupt Priority O Low O Medium O High
No Interrupt	
More Info	Kext > Finish Cancel

Figure 3-20. Identify Interrupt Signals Dialog Box

• Click Next until you reach the last page and click Finish to close wizard

You will see following message box. Click Yes

Xilinx Pla	tform Studio 🛛 🔀
?	You must close and reopen the project in order to utilize the imported peripheral. Do you want to close and reopen the project now?
	<u>Y</u> es <u>N</u> o

Figure 3-21. Confirmation Dialog Box





Using **Project** \rightarrow **Add/Edit Core...** (dialog) from **XPS**, add **my_led** to the project, make bus connections, generate address for the **my_led** instance, add necessary ports to the instance, name them appropriately and bring out the data port. Add following to the UCF file:

Net fpga_0_LEDs_4Bit_GPIO_d_out<0> LOC=AC4; Net fpga_0_LEDs_4Bit_GPIO_d_out<1> LOC=AC3; Net fpga_0_LEDs_4Bit_GPIO_d_out<2> LOC=AA6; Net fpga_0_LEDs_4Bit_GPIO_d_out<3> LOC=AA5;

- Click **Project** → Add/Edit Core... (dialog) from XPS.
- In the Peripherals tab, click the OPB radio button from the Bus group, click the Custom IP radio button from the Component Filter group, then highlight the custom peripheral my_led from the list box, and click << Add button to add it to the hardware system</p>
- Switch to the **Bus Connections** tab, attach **my_led_0** to OPB bus as a salve device by clicking on the cell that corresponds to **my_led_0 sopb** row and **opb** column
- Switch to the Addresses tab, select size as 512 from drop down box for my_led_0 and click the Generate Addresses button to let XPS automatically assign addresses for all the peripherals for you, including the my_led custom peripheral
- Switch to the **Ports** tab and perform the following
 - Select my_led_0 from the Ports Filter combo box
 - Select the LED and OPB_Clk ports displayed under my_led_0 from the right list box
 - Click << Add to add them to the Internal Ports Connections table
 - Locate **my_led_0**'s **OPB_Clk** port in the table, modify its Net Name to **sys_clk_s** by selecting from its drop down list box
 - Locate my_led_0's LED port in the table, modify its Net Name to fpga_0_LEDs_4Bit_GPIO_d_out by typing in the Net Name field
 - Select my_led_0's LED port click on Make External
 - In the **External Ports Connections** table, select **my_led_0_LED** and type **[0:3]** in the corresponding **Range** field
- Click **OK** to close the **Add/Edit Hardware Platform Specifications** dialog, open up the **system.mhs** file and verify following snippets

PORT fpga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX, DIR = IN

PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX, DIR = OUT

PORT sys_clk_pin = dcm_clk_s, DIR = IN

BFM Simulation Lab



PORT sys_rst_pin = sys_rst_s, DIR = IN

PORT PUSH = PUSH, VEC = [0:4], DIR = I

PORT DIP = DIP, VEC = [0:3], DIR = I

PORT fpga_0_LEDs_4Bit_GPIO_d_out = fpga_0_LEDs_4Bit_GPIO_d_out, VEC = [0:3], DIR = O

• Add following pin assignments in the UCF file

Net fpga_0_LEDs_4Bit_GPIO_d_out<0>LOC=AC4; Net fpga_0_LEDs_4Bit_GPIO_d_out<1>LOC=AC3; Net fpga_0_LEDs_4Bit_GPIO_d_out<2>LOC=AA6; Net fpga_0_LEDs_4Bit_GPIO_d_out<3>LOC=AA5;

Save and close the UCF file

Develop Application and Verify the Design in Hardware Step 5

Run LibGen. Edit TestApp.c source file in the TestApp software project to display its settings on the LEDs. Compile the program successfully.

- In XPS, select **Options** \rightarrow **Project Options** and then select the **Hierarchy and Flow** tab
- Under Implementation Tool Flow, select the XPS (Xflow) option and click OK to accept the settings
- Click Tools → Generate Libraries and BSPs to run the library generator
- In the Application tab, double-click on TestApp_Memory.c under the Sources of TestApp_Memory software project
- Edit the **TestApp_Memory.c** file to match the following

#include "xparameters.h"
#include "xgpio.h"
#include "xutil.h" 02 03 04 05 06 07 08 09 10 11 12 13 14 #include 'my_led.h" int main (void) -{· XGpio dip_push; int i, psb_check, dip_check; ...print("--.Start.of.the.program.--.\r\n");XGpio_Initialize(&dip_push,XPAR_DIP_PUSH_DEVICE_ID);XGpio_SetDataDirection(&dip_push,1,0xffffffff); 16 17 18 20 21 22 23 24 25 26 27 28 29 30 31 ...XGpio_Initialize(&dip_push,XPAR_DIP_PUSH_DEVICE_ID); *XGpio_SetDataDirection(&dip_push,2,0xffffffff); >> while(1){ » psb_check = XGpio_DiscreteRead(&dip_push,1); xil_printf("Push Buttons Status : *%x \n\r",psb_check); dip_check = XGpio_DiscreteRead(&dip_push,2); >> » » » » >>xil_printf("Dip.Switch.Status.:.%x\n\r",dip_check) >> MY LED mWriteReg(XPAR MY LED U BASEADDR, U, dip check); » >> sleep(1); 3 print("-- End of the program -- \r\n"); return(0) 32 3

- Compile the program successfully
- Click Tools → Update Bitstream to generate the bit file

This may take over 10 minutes.

- Download the bit file on the board
- Change dip switch settings and see the corresponding LED turning ON and OFF, verifying the functionality

Conclusion

This lab led you through the creating a custom IP. Once created, it guided you to import and add the custom IP into a system. Further, you developed an application using the custom IP driver and verified its functionality in hardware. Thus you were able to see the ease of creating/importing a custom IP.

BFM Simulation Lab



Completed MHS File

Created by Base System Builder Wizard for Xilinx EDK 7.1.1 Build EDK_H.11.3 # Wed Jun 08 23:27:28 2005 # Target Board: Xilinx XUP Virtex-II Pro Development System Rev A # Family: virtex2p # Device: xc2vp30 # Package: ff896 # Speed Grade: -6 # Processor: PPC 405 # Processor clock frequency: 300.000000 MHz # Bus clock frequency: 100.000000 MHz # Debug interface: FPGA JTAG # On Chip Memory: 64 KB

PARAMETER VERSION = 2.1.0

PORT fpga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX, DIR = INPUT PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX, DIR = OUTPUT PORT sys_clk_pin = dcm_clk_s, DIR = INPUT PORT sys_rst_pin = sys_rst_s, DIR = INPUT PORT PUSH = PUSH, VEC = [0:4], DIR = I PORT DIP = DIP, VEC = [0:3], DIR = I PORT fpga_0_LEDs_4Bit_GPIO_d_out = fpga_0_LEDs_4Bit_GPIO_d_out, VEC = [0:3], DIR = O

```
BEGIN ppc405

PARAMETER INSTANCE = ppc405_0

PARAMETER HW_VER = 2.00.c

BUS_INTERFACE JTAGPPC = jtagppc_0_0

BUS_INTERFACE IPLB = plb

BUS_INTERFACE DPLB = plb

PORT C405RSTCHIPRESETREQ = C405RSTCHIPRESETREQ

PORT C405RSTCORERESETREQ = C405RSTCORERESETREQ

PORT C405RSTSYSRESETREQ = C405RSTSYSRESETREQ

PORT C405RSTSYSRESETREQ = C405RSTSYSRESETREQ

PORT RSTC405RESETCHIP = RSTC405RESETCHIP

PORT RSTC405RESETCORE = RSTC405RESETCORE

PORT RSTC405RESETSYS = RSTC405RESETSYS

PORT CPMC405CLOCK = proc_clk_s

PORT PLBCLK = sys_clk_s

END
```

BEGIN ppc405 PARAMETER INSTANCE = ppc405_1 PARAMETER HW_VER = 2.00.c BUS_INTERFACE JTAGPPC = jtagppc_0_1 END



BEGIN jtagppc_cntlr PARAMETER INSTANCE = jtagppc_0 PARAMETER HW_VER = 2.00.a BUS_INTERFACE JTAGPPC0 = jtagppc_0_0 BUS_INTERFACE JTAGPPC1 = jtagppc_0_1 END

BEGIN proc_sys_reset PARAMETER INSTANCE = reset_block PARAMETER HW_VER = 1.00.a PARAMETER C_EXT_RESET_HIGH = 0 PORT Chip_Reset_Req = C405RSTCHIPRESETREQ PORT Core_Reset_Req = C405RSTCORERESETREQ PORT System_Reset_Req = C405RSTSYSRESETREQ PORT Rstc405resetchip = RSTC405RESETCHIP PORT Rstc405resetcore = RSTC405RESETCORE PORT Rstc405resetsys = RSTC405RESETCORE PORT Rstc405resetsys = RSTC405RESETSYS PORT Dcm_locked = dcm_0_lock PORT Bus_Struct_Reset = sys_bus_reset PORT Slowest_sync_clk = sys_clk_s PORT Ext_Reset_In = sys_rst_s END

BEGIN plb_v34 PARAMETER INSTANCE = plb PARAMETER HW_VER = 1.02.a PARAMETER C_DCR_INTFCE = 0 PARAMETER C_EXT_RESET_HIGH = 1 PORT SYS_Rst = sys_bus_reset PORT PLB_Clk = sys_clk_s END

BEGIN opb_v20 PARAMETER INSTANCE = opb PARAMETER HW_VER = 1.10.c PARAMETER C_EXT_RESET_HIGH = 1 PORT SYS_Rst = sys_bus_reset PORT OPB_Clk = sys_clk_s END

BEGIN plb2opb_bridge PARAMETER INSTANCE = plb2opb PARAMETER HW_VER = 1.01.a PARAMETER C_DCR_INTFCE = 0 PARAMETER C_RNG0_BASEADDR = 0x40000000 PARAMETER C_RNG0_HIGHADDR = 0x7fffffff PARAMETER C_NUM_ADDR_RNG = 1 BUS_INTERFACE SPLB = plb BUS_INTERFACE MOPB = opb PORT OPB_Clk = sys_clk_s PORT PLB_Clk = sys_clk_s END

BFM Simulation Lab

BEGIN opb_uartlite PARAMETER INSTANCE = RS232_Uart_1 PARAMETER HW_VER = 1.00.b PARAMETER C_BAUDRATE = 115200 PARAMETER C_DATA_BITS = 8 PARAMETER C_ODD_PARITY = 0 PARAMETER C_USE_PARITY = 0 PARAMETER C_LUSE_PARITY = 0 PARAMETER C_RUSE_PARITY = 0 PARAMETER C_RUSE_PARITY

BEGIN plb_bram_if_cntlr PARAMETER INSTANCE = plb_bram_if_cntlr_1 PARAMETER HW_VER = 1.00.b PARAMETER c_plb_clk_period_ps = 10000 PARAMETER c_baseaddr = 0xffff0000 PARAMETER c_highaddr = 0xfffffff BUS_INTERFACE SPLB = plb BUS_INTERFACE PORTA = plb_bram1 PORT PLB_Clk = sys_clk_s END

BEGIN bram_block PARAMETER INSTANCE = plb_bram_if_cntlr_1_bram PARAMETER HW_VER = 1.00.a BUS_INTERFACE PORTA = plb_bram1 END

BEGIN dcm_module PARAMETER INSTANCE = dcm_0 PARAMETER HW_VER = 1.00.a PARAMETER C_CLK0_BUF = TRUE PARAMETER C_CLKFX_BUF = TRUE PARAMETER C_CLKFX_DIVIDE = 1 PARAMETER C_CLKFX_MULTIPLY = 3 PARAMETER C_CLKIN_PERIOD = 10.000000 PARAMETER C_CLK_FEEDBACK = 1X PARAMETER C_EXT_RESET_HIGH = 1 PORT LOCKED = dcm_0_lock PORT CLKIN = dcm_clk_s PORT RST = net_gnd PORT CLKFX = proc_clk_s PORT CLK0 = sys_clk_s PORT CLKFB = sys_clk_s END

BEGIN opb_gpio PARAMETER INSTANCE = dip_push PARAMETER HW_VER = 3.01.b PARAMETER C_GPIO_WIDTH = 32 PARAMETER C_ALL_INPUTS = 1 PARAMETER C_IS_BIDIR = 0 PARAMETER C_IS_DUAL = 1 PARAMETER C_ALL_INPUTS_2 = 1 PARAMETER C_SBIDIR_2 = 0 PARAMETER C_BASEADDR = 0x40000000 PARAMETER C_HIGHADDR = 0x40000000 PARAMETER C_HIGHADDR = 0x4000ffff BUS_INTERFACE SOPB = opb PORT GPIO_in = DIP PORT GPIO_in = PUSH PORT OPB_CIk = sys_clk_s END

BEGIN plb_bram_if_cntlr PARAMETER INSTANCE = plb_bram_if_cntlr_2 PARAMETER HW_VER = 1.00.b PARAMETER c_plb_clk_period_ps = 10000 PARAMETER c_baseaddr = 0x00000000 PARAMETER c_highaddr = 0x00003fff BUS_INTERFACE SPLB = plb BUS_INTERFACE PORTA = plb_bram2 PORT plb_clk = sys_clk_s END

BEGIN bram_block PARAMETER INSTANCE = plb_bram_if_cntrl_2_bram PARAMETER HW_VER = 1.00.a BUS_INTERFACE PORTA = plb_bram2 END

BEGIN my_led PARAMETER INSTANCE = my_led_0 PARAMETER HW_VER = 1.00.a PARAMETER C_BASEADDR = 0x7d800000 PARAMETER C_HIGHADDR = 0x7d80ffff BUS_INTERFACE SOPB = opb PORT LED = fpga_0_LEDs_4Bit_GPIO_d_out PORT OPB_Clk = sys_clk_s END



Completed MSS File

PARAMETER VERSION = 2.2.0

BEGIN OS PARAMETER OS_NAME = standalone PARAMETER OS_VER = 1.00.a PARAMETER PROC_INSTANCE = ppc405_0 PARAMETER STDIN = RS232_Uart_1 PARAMETER STDOUT = RS232_Uart_1 END

BEGIN OS PARAMETER OS_NAME = standalone PARAMETER OS_VER = 1.00.a PARAMETER PROC_INSTANCE = ppc405_1 END

BEGIN PROCESSOR PARAMETER DRIVER_NAME = cpu_ppc405 PARAMETER DRIVER_VER = 1.00.a PARAMETER HW_INSTANCE = ppc405_0 PARAMETER COMPILER = powerpc-eabi-gcc PARAMETER ARCHIVER = powerpc-eabi-ar PARAMETER CORE_CLOCK_FREQ_HZ = 300000000 END

BEGIN PROCESSOR PARAMETER DRIVER_NAME = cpu_ppc405 PARAMETER DRIVER_VER = 1.00.a PARAMETER HW_INSTANCE = ppc405_1 PARAMETER COMPILER = powerpc-eabi-gcc PARAMETER ARCHIVER = powerpc-eabi-ar END

BEGIN DRIVER PARAMETER DRIVER_NAME = plbarb PARAMETER DRIVER_VER = 1.01.a PARAMETER HW_INSTANCE = plb END

BEGIN DRIVER PARAMETER DRIVER_NAME = opbarb PARAMETER DRIVER_VER = 1.02.a PARAMETER HW_INSTANCE = opb END

BEGIN DRIVER

BFM Simulation Lab

PARAMETER DRIVER_NAME = plb2opb PARAMETER DRIVER_VER = 1.00.a PARAMETER HW_INSTANCE = plb2opb END

BEGIN DRIVER PARAMETER DRIVER_NAME = uartlite PARAMETER DRIVER_VER = 1.00.b PARAMETER HW_INSTANCE = RS232_Uart_1 END

BEGIN DRIVER PARAMETER DRIVER_NAME = bram PARAMETER DRIVER_VER = 1.00.a PARAMETER HW_INSTANCE = plb_bram_if_cntlr_1 END

BEGIN DRIVER PARAMETER DRIVER_NAME = generic PARAMETER DRIVER_VER = 1.00.a PARAMETER HW_INSTANCE = dcm_0 END

BEGIN DRIVER PARAMETER DRIVER_NAME = gpio PARAMETER DRIVER_VER = 2.00.a PARAMETER HW_INSTANCE = dip_push END

BEGIN DRIVER PARAMETER DRIVER_NAME = bram PARAMETER DRIVER_VER = 1.00.a PARAMETER HW_INSTANCE = plb_bram_if_cntlr_2 END

BEGIN DRIVER PARAMETER DRIVER_NAME = my_led PARAMETER DRIVER_VER = 1.00.a PARAMETER HW_INSTANCE = my_led_0 END



Completed C File

#include "xparameters.h"
#include "xgpio.h"
#include "xutil.h"
#include "my_led.h"

XGpio dip_push; int i,psb_check, dip_check;

```
print("-- Start of the program --\r\n");
XGpio_Initialize(&dip_push,XPAR_DIP_PUSH_DEVICE_ID);
```

XGpio_SetDataDirection(&dip_push,1,0xffffffff);

```
XGpio_Initialize(&dip_push,XPAR_DIP_PUSH_DEVICE_ID);
XGpio_SetDataDirection(&dip_push,2,0xffffffff);
```

```
while(1)
{
        psb_check = XGpio_DiscreteRead(&dip_push,1);
        xil_printf("Push Buttons Status : %x\n\r",psb_check);
        dip_check = XGpio_DiscreteRead(&dip_push,2);
        xil_printf("Dip Switch Status : %x\n\r",dip_check);
        MY_LED_mWriteReg(XPAR_MY_LED_0_BASEADDR,0,dip_check);
        sleep(1);
}
```

```
print("-- End of the program --\r\n");
return 0;
```

}