Embedded System Design using IP Integrator

Introduction

This lab guides you through the process of using Vivado and IP Integrator to create a simple ARM Cortex-A9 based processor design targeting either the ZedBoard or the Zybo development board. You will use IP Integrator to create the hardware block diagram and SDK (Software Development Kit) to create an example application to verify the hardware functionality.

Objectives

After completing this lab, you will be able to:

- Create a Vivado project for a Zynq system
- Use the IP Integrator to create a hardware system
- Use IP Catalog to use AXI GPIO peripheral to extend the design
- Use SDK to create a standard memory test project
- Run the test application on the board

Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

Design Description

The purpose of the lab exercise is to walk you through a complete hardware and software processor system design. The following diagram represents the completed design (**Figure 1**).

In this lab, you will use IP Integrator to create a processing system based design consisting of the following:

- ARM Cortex A9 core (PS)
- UART for serial communication
- DDR3 controller for external DDR3_SDRAM memory
- AXI Interconnect block
- Two instances of GPIO peripheral to connect push-buttons and slide switches



Figure 1. Processor Design of this Lab



General Flow for this Lab



Create a Vivado Project

Step 1

1-1. Launch Vivado and create an empty project either targeting the ZedBoard (having xc7z020clg484-1 device) or Zybo (having xc7z010clg400-1 device) and using the Verilog language.

References to <2014_2_zynq_labs> means c:\xup\sys_design\2014_2_zynq_labs and <2014_2_zynq_sources> means c:\xup\sys_design\2014_2_zynq_sources directories.

- 1-1-1. Open Vivado by selecting Start > All Programs > Xilinx Design Tools > Vivado 2014.2 > Vivado 2014.2
- 1-1-2. Click Create New Project to start the wizard. You will see the Create a New Vivado Project dialog box. Click Next.
- 1-1-3. Click the Browse button of the *Project Location* field of the **New Project** form, browse to <2014_2_zynq_labs>, and click Select.
- 1-1-4. Enter **lab3** in the *Project Name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.
- 1-1-5. Select RTL Project in the Project Type form, and click Next.
- 1-1-6. Select Verilog as the *Target language* and as the *Simulator language* in the *Add Sources* form, and click Next.
- 1-1-7. Click Next two more times to skip Adding Existing IP and Add Constraints
- 1-1-8. In the Default Part form, select Boards specify, and either select ZedBoard or Zybo. Click Next.
- 1-1-9. Check the *Project Summary* and click **Finish** to create an empty Vivado project.



Create the System Using the IP Integrator

Step 2

2-1. Use the IP Integrator to create a new Block Design, and generate the ARM Cortex-A9 processor based hardware system.

- 2-1-1. In the *Flow Navigator* pane, click **Create Block Design** under IP Integrator
- 2-1-2. Enter system for the design name and click OK.

🚴 Create Bloc	k Design
Please s	pecify name of block design
<u>D</u> esign name:	system
Directory:	So <local project="" to=""></local>
	OK Cancel

Figure 2. Create New Block Diagram

2-1-3. IP from the catalog can be added in different ways. Click on <u>Add IP</u> in the message at the top of the *Diagram* panel, or click the *Add IP icon* in the block diagram side bar, press Ctrl + I, or right-click anywhere in the Diagram workspace and select Add IP

og.

1	Diag	jra	m	×			
₹	🌲 s	yst	em				
•	()	тł	nis d	lesign is emp	ty. To get started	, <u>Add IP</u>	from the catal
•							
٩,		[6	Properties		Ctrl+E	1
R			×	Delete		Delete	
ŵ			ħ	Сору		Ctrl+C	
¢			th,	Paste		Ctrl+V	
٩			θ,	Search		Ctrl+F	
₽			R	Select All		Ctrl+A	_
•		l	₽	Add IP		Ctrl+I	
			6	IP Settings.			
				Validate De	sign	F6	_
				Create Hier	archy		
				Create Com	iment		
Gi				Create Port		Ctrl+K	
				Create Inte	face Port	Ctrl+L	
			Ø	Regenerate	Layout		
			1	Save as PD	F File		

Figure 3. Add IP to Block Diagram



2-1-4. Once the IP Catalog is open, type "zy" into the Search bar, find and double click on **ZYNQ7 Processing System** entry, or click on the entry and hit the Enter key to add it to the design.

<u>S</u> earch: Q- zyn	(2 matches)
Name 1	VLNV
 ZYNQ7 Processing System ZYNQ7 Processing System BFM 	xilinx.com:ip:processing_system7:5.4 xilinx.com:ip:processing_system7_bf
Select and press ENTER or drag and d	Irop, ESC to cancel

Figure 4. Add Zynq block to the design

2-1-5. Notice the message at the top of the Diagram window that *Designer Assistance* available. Click on **Run Block Automation** and select /processing_system7_0



Figure 5. Designer Assistance message

2-1-6. Click OK when prompted to run automation.

🚴 Run Bloc	k Automat	ion				X
Zynq7 FIXED	' block autor)_IO, Trigge	nation appli r and DDR ii	lies current board interfaces	preset and genera	te external connections	s for
NOTE wish t	NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration					
Instar	nce: /proces	sing_system	m7_0			
Make Interfa	ce External:	FIXED_IO,	DDR			
Apply Board	Preset:	 Image: A start of the start of				
<u>C</u> ross Trigge	r In:	Disable	•			
Cross <u>T</u> rigge	r Out:	Disable	•			
					ОК	Cancel

Figure 6. Run Block Automation

Once Block Automation has been complete, notice that ports have been automatically added for the DDR and Fixed IO, and some additional ports are now visible. A default configuration for the Zynq depending on the target board has been applied which will now be modified.





Figure 7. Zynq Block with DDR and Fixed IO ports

- **2-1-7.** In the block diagram, double click on the *Zynq* block to open the *Customization* window for the Zynq processing system.
- **2-1-8.** A block diagram of the Zynq should now be open, showing various configurable blocks of the Processing System.

At this stage, the designer can click on various configurable blocks (highlighted in green) and change the system configuration.



Figure 8: Zynq System Configuration View



2-2. Configure I/O Peripherals block to use UART 1 only

Only the UART is required for this lab, so all other peripherals will be deselected.

- **2-2-1.** Click on one of the peripherals (in green) in the *IOP Peripherals* block, or select the *MIO Configuration* tab in the Page Navigator on left to open the configuration form
- **2-2-2.** Expand I/O peripherals if necessary, and deselect all the I/O peripherals except UART 1. i.e. Remove: ENET 0 USB 0

SD 0

Expand **GPIO** to deselect *GPIO MIO* Expand **Memory Interfaces** to deselect *Quad SPI Flash* Expand **Application Processor Unit** to deselect *Timer 0*.

Page Navigator «	MIC) Configuration				
Zynq Block Design	+	Bank 0 IO Voltage LVCMOS 3.3	3V -	Bank 1 IO V	oltage LVCMOS 1.8V	•
PS-PL Configuration		Search: Q-				
Peripheral I/O Pins	a	Peripheral	Ю	Signal	ІО Туре	:
MIO Configuration		Memory Interfaces				
Clock Configuration		SRAM/NOR Flash				
DDR Configuration		NAND Flash				
SMC Timing Calculation		I/O Peripherals				
Interrupto		🕀 🔲 ENET 0				
Interrupts		⊕. ENET 1				
		🔲 USB 0				
		🔲 USB 1				
		🖶 🔲 SD 0				
	-	⊞ 🔲 SD 1				
		UART 0				
		⊕ 🔽 UART 1	MIO 48 49 🔹			
		⊞ [] I2C 1				
		🕀 🔲 SPI 0				
		🕀 🔲 SPI 1				
		🕀 🔲 CAN 0				
		🕀 🔲 CAN 1				
		GPIO				
		GPIO MIO				
		EMIO GPIO (Wi				
		🕀 Resets				
		Application Processor Unit				
		Timer 0				
		Timer 1				

Figure 9. Selecting only UART 1

2-2-3. Click on the *Clock Configuration*, expand *PL Fabric Clocks*, and observe that **FCLK_CLK0** is enabled with **100** MHz frequency.



2-2-4. Click **OK**.

We left the rest of the configuration as is since we want to add two GPIO peripherals in the PL section which will be connected through the GP0 master interface, using FCLK_CLK0 as the clock source and FCLK_RESET0_N as the reset control signal.



Figure 10. Updated Zynq Block

Add Two Instances of GPIO

Step 3

- 3-1. Add two instances of the GPIO Peripheral from the IP catalog to the processor system.
- **3-1-1.** Click the Add IP icon IP and search for "gp".

Search: Q- gp	(7 matches)
Name 1	VLNV
 GGPP LTE Channel Estimator GGPP LTE MIMO Decoder GGPP LTE MIMO Encoder GGPP LTE Turbo Encoder GGPP Mixed Mode Turbo Decoder GGPP Turbo Encoder AVECTIO 	xilinx.com:ip:lte_3gpp_channel_esti xilinx.com:ip:lte_3gpp_mimo_decode xilinx.com:ip:lte_3gpp_mimo_encode xilinx.com:ip:tcc_encoder_3gpplte:4.0 xilinx.com:ip:tcc_decoder_3gppmm:2.0 xilinx.com:ip:tcc_encoder_3gpp:5.0
Select and press ENTER or drag and dr	rop, ESC to cancel

Figure 11. Add GPIO IP

3-1-2. Double-click the **AXI GPIO** to add the core to the design. The core will be added to the design and the block diagram will be updated.



Figure 12. Zynq system with AXI GPIO added

3-1-3. Click on the AXI GPIO block to select it, and in the properties tab,

ZedBoard: Change the name to sw_8bit



Block Properties	$-\Box \triangleleft \times$
$\leftarrow \rightarrow \bigotimes \Bbbk$	
🗣 axi_gpio_0	
Name: sw_8bit Parent name: system	
General Properties	

Figure 13: Change AXI GPIO default name

Zybo: Change the name to sw_4bit

Block Properties	_ 🗆 🖻 ×
← → 🔯 📐	
🗣 sw_4bit	
Name: sw_4bit Parent name: system	
General Properties	

Figure 13. Change AXI GPIO default name

3-1-4. Double click on the AXI GPIO block to open the customization window.

If a board specify was selected during the project creation, and board files were available for the Zybo, Vivado has knowledge of available resources on the board.

3-1-5. Click on Generate Board Based IO Constraints, and under Board Interface, for GPIO.

ZedBoard: Click on Custom to view the dropdown menu options, and select sws 8bits

🖵 Re-customize IP			×
AXI GPIO (2.0)		ſ	R
🍘 Documentation 📄 IP Location			
Show disabled ports	Component Name Zynq_axi_gpio_1 Board IP Configuration Generate Board based IO Constrait Associate IP interface with EM.AVNET IP Interface SPIO SpIO2 Enable Interrupt	_0 nts .COM:ZYNQ:ZED:D Board interface Board interface Custom Custom btns 5bits leds 6bits avvs 6bits	
		OK Cano	el

Figure 14: Configuring GPIO instance for switches on ZedBoard



Zybo: Click on Custom to view the dropdown menu options, and select sws 4bits

income transferre mathema in	a linearthe Astronom	X
Component Name system_	axi_gpio_0_0	
Generate Board based IC Associate IP interface with IP Interface	0 Constraints DIGILENTINC.COM:ZYBO:PART0:1.0 Boar Board Interface	d interface
GPIO	Custom	•
GPIO2	Custom btns 4bits	
	Component Name System_ Board IP Configuration Generate Board based IC Associate IP interface with IP Interface GPIO GPIO2	Component Name system_axi_gpio_0_0 Board IP Configuration Generate Board based IO Constraints Associate IP interface with DIGILENTINC.COM:ZYB0:PART0:1.0 Board IP Interface Board Interface GPI0 Custom GPI02 Custom Custo

Figure 14. Configuring GPIO instance for switches on Zybo

3-1-6. Click the *IP Configuration* tab. Notice the set GPIO Width.

Notice that the peripheral can be configured for two channels, but, since we want to use only one channel without interrupt, leave the *GPIO Supports Interrupts* and *Enable Channel 2 unchecked*.

ZedBoard: The GPIO width is set to 8.

Board IP Configura	tion	
GPIO		
✓ All Inputs		
All Outputs		
GPIO Width	8	[132]
Default Output Value	0x0000000	[0x0000000,0xFFFFFF]
Default Tri State Value	0xFFFFFFF	[0x0000000,0xFFFFFF]
Enable Dual Channel		

Figure 15. Configuring GPIO instance for ZedBoard

Zybo: The GPIO width is set to 4.

Board IP Configura	tion		
GPIO			
✓ All Inputs			
All Outputs			
GPIO Width	4	0	[132]
Default Output Value	0x00000000		[0x00000000,0xFFFFFFF]
Default Tri State Value	0xFFFFFFF		[0x00000000.0xFFFFFFF]

Figure 15. Configuring GPIO instance for Zybo



- 3-1-7. Click OK to save and close the customization window
- **3-1-8.** Notice that *Design assistance* is available. Click on **Run Connection Automation**, and select /sw_8bit/S_AXI for the ZedBoard or /sw_4bit/S_AXI for the Zybo.
- 3-1-9. Click OK when prompted to automatically connect the master and slave interfaces

🚴 Run Con	nection Automation	X
Conne	ect Slave interface (/sw_8bit/S_AXI) to a selected Master address space.	
Master:	/processing_system7_0/M_AXI_GP0	
Clock Conne	ction (for unconnected clks) : Auto	-
	ОКСС	ancel

Figure 16. Run Connection Automation for ZedBoard

A Run Connection Automation		X
Connect Slave interface (/sw_4bit	/S_AXI) to a selected Master address space.	
Master:	/processing_system7_0/M_AXI_GP0	
<u>Clock Connection (for unconnected clks)</u>	Auto	-
	ОК	Cancel

Figure 16. Run Connection Automation for Zybo

3-1-10. Notice two additional blocks, *Proc Sys Reset*, and *AXI Interconnect* have automatically been added to the design. (The blocks can be dragged to be rearranged, or the design can be redrawn.)









Figure 17. Design with sw_4bit automatically connected for Zybo

- **3-1-11.** Add another instance of the *GPIO* peripheral (**Add IP**), and using the board flow, configure it to connect to either the **btns 5bit** on ZedBoard or the **btns 4bits** on Zybo.
- **3-1-12.** Change the name of the block either to **btns_5bit** for ZedBoard or to **btns_4bit** for Zybo (Click on the block to select it, and change the name in the properties view)

At this point connection automation could be run, or the block could be connected manually. This time the block will be connected manually.

3-1-13. Double click on the AXI Interconnect and change the *Number of Master* Interfaces to **2** and click **OK**



🗜 Re-customize IP		X				
AXI Interconnect (2.1)						
🖗 Documentation 🚞 IP Location						
Component Name system_processing_system?	n7_0_axi_periph_0					
Top Level Settings Slave Interfaces	Master Interfaces	_				
Number of Slave Interfaces	1 -					
Number of Master Interfaces	2 🔹					
Interconnect Optimization Strategy	Custom 👻					
Interconnect Optimization Strategy Custom AXI Interconnect includes IP Integrator automatic converter insertion and configuration. When the endpoint IPs attached to the interfaces of the AXI Interconnect differ in width, clock or protocol, a converter IP will automatically be added inside the interconnect. If a converter IP is inserted, IP integrator's parameter propagation automatically configures the converter to match the design. To see which conversion IPs have been inserted, use the IP integrator 'expand hierarchy' buttons to explore inside the AXI Interconnect hierarhcy. NOTE:Addressing information for AXI Interconnect is specified in the IP Integrator address editor. Enable Advanced Configuration Options						
	OK Cance					

Figure 18. Add slave port to AXI Interconnect

3-1-14. ZedBoard: Click on the *s_axi* port of the new *btns_5bit* block, and drag the pointer towards the AXI Interconnect block. The message *Found 1* interface should appear, and a green tick should appear beside the *M01_AXI* port on the AXI Interconnect indicating this is a valid port to connect to. Drag the pointer to this port and release the mouse button to make the connection.



Figure 19: Connect the interfaces

Zybo: Click on the *s_axi* port of the new *btns_4bit* block, and drag the pointer towards the AXI Interconnect block. The message *Found 1* interface should appear, and a green tick should appear beside the *M01_AXI* port on the AXI Interconnect indicating this is a valid port to connect to. Drag the pointer to this port and release the mouse button to make the connection.







3-1-15. In a similar way, connect the following ports: AXI Interconnect M01_ACLK -> Zynq7 Processing System FCLK_CLK0 AXI Interconnect M01_ARESETN -> Proc Sys Reset peripheral aresetn

ZedBoard:

btns_5bit s_axi_aclk -> Zynq7 Processing System FCLK_CLK0
btns_5bit s_axi_aresetn -> Proc Sys Reset peripheral_aresetn

Zybo:

btns_4bit s_axi_aclk -> Zynq7 Processing System FCLK_CLK0 btns_4bit s_axi_aresetn -> Proc Sys Reset peripheral_aresetn

The block diagram should look similar to this:



Figure 20. Block diagram view after adding the peripherals for ZedBoard





Figure 20. Block diagram view after adding the peripherals for Zybo

3-1-16. Click on the Address Editor, and expand **processing_system7_0 > Data > Unmapped Slaves.**

ZedBoard: Notice that sw_8bit has been automatically assigned an address, but **btns_5bit** has not. Right click on *btns_5bit* and select **Assign Address.** Note that both peripherals are assigned in the address range of 0x40000000 to 0x7FFFFFFF (GP0 range).

P	Diagram 🗙 🔣 Address Editor 🗙					
٩	Cell	Interf	ace Pin Base Na	me Offset Address	Range	High Address
\mathbf{Z}	processing_system7_0					
	🖻 🔠 Data (32 address bits : 4G)					
-	🚥 sw_8bit	S_AXI	Reg	0x41200000	64K 🤜	0x4120FFFF
H	🔤 🚥 btns_5bit	S_AXI	Reg	0x41210000	64K 🤜	0x4121FFFF

Figure 21. Peripheral Memory Map for ZedBoard

Zybo: Notice that sw_4bit has been automatically assigned an address, but **btns_4bit** has not. Right click on *btns_4bit* and select **Assign Address.** Note that both peripherals are assigned in the address range of 0x40000000 to 0x7FFFFFF (GP0 range).

30	Diagram 🗙 🔣 Address Editor 🗙					
٩	Cell	Interface Pin	Base Name	Offset Address	Range	High Address
Z	Image: processing_system7_0					
	🖃 🖽 Data (32 address bits : 4G)					
-	🚥 sw_4bit	S_AXI	Reg	0x41200000	64K 🔹	0x4120FFFF
矖	🚥 btns_4bit	S_AXI	Reg	0x41210000	64K 🔹	0x4121FFFF
			_			

Figure 21. Peripherals Memory Map for Zybo

Make GPIO Peripheral Connections External

Step 4

4-1. The push button and dip switch instances will be connected to the corresponding pins on the board. This can be done manually, or using Designer Assistance. The location constraints are automatically applied by the tools as the information for the target board is already known. Normally, one would consult the board's user manual for this information.



In the Diagram view, notice that *Designer Assistance* is available. This will be ignored for now, and a port will be manually created and connected for the switch instance. Designer Assistance will be used to connect the buttons peripheral.

- **4-1-1.** Right-Click on the gpio port of the *sw_8bit* instance for ZedBoard or the *sw_4bit* for Zybo, and select **Make External** to create the external port. This will create the external port named *gpio* and connected to the peripheral.
- **4-1-2.** Select the *gpio* port and change the name to **sw_8bit** for ZedBoard or **sw_4bit** for Zybo in its properties form.

The width of the interface will be automatically determined by the upstream block.

4-1-3. Connection automation will be used to create a port for the buttons.

ZedBoard: Add the port for the *btns_5bit* component automatically, by clicking on *Run Connection Automation*, and selecting */btns_5bit/GPIO*. In the *Select Board Interface* drop down menu, select **btns_5bits**, and click **OK** to create and connect the external port.

Zybo: Add the port for the *btns_4bit* component automatically, by clicking on *Run Connection Automation*, and selecting */btns_4bit/GPIO*. In the *Select Board Interface* drop down menu, select **btns_4bits**, and click **OK** to create and connect the external port.

4-1-4. Run Design Validation (**Tools -> Validate Design**) and verify there are no errors. The design should now look similar to the diagram below:







Figure 22. Completed design for Zybo



- **4-1-5.** In the Flow Navigator window, click on the *Generate Block Design* to generate the relevant files of each IP in the design. Click **Generate**.
- 4-1-6. In the *sources* view, Right Click on the block diagram file, **system.bd**, and select **Create HDL** Wrapper to create the HDL wrapper file. When prompted, select **Let Vivado manage wrapper** and auto-update, click **OK**.
- 4-1-7. In the Flow Navigator, click **Run Synthesis**. (Click **Save** when prompted) and when synthesis completes, select **Open Synthesized Design** and click **OK**
- 4-1-8. In the shortcut Bar, select I/O Planning from the Layout dropdown menu

😬 Default Layout	
😬 Default Layout	
💾 I/O Planning	
😬 Clock Planning	
😬 Floorplanning	

Figure 23. Switch to the IO planning view

4-1-9. ZedBoard: In the I/O ports tab, expand *GPIO_42989 > BTNs_5bit_tri_i*, and notice pins have already been assigned to this peripheral. The pin information was included in the board files, and automatically assigned when the IP was connected to the port. The *sw_8bit_tri_i* have also been automatically assigned pin locations, along with the other Fixed ports in the design.

I/O	Ports								
9	Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Site	Fixed	Bank	I/O Std
\mathbf{Z}	🖃 🐼 All ports (143)								
	DDR_1497 (71)	In/Out					1		(Multiple)*
E	FIXED_IO_1497 (59)	In/Out					-		(Multiple)*
8	🖨 🕼 GPIO_42989 (5)	Input					1	34	LVCMOS25*
C	🗐 👺 btns_5bits_tri_i (5)	Input					-	34	LVCMOS25*
Þ	btns_5bits_tri_i[4]	Input	btns_5bits_tri			T18	1	34	LVCMOS25*
	btns_5bits_tri_i[3]	Input	btns_5bits_tri			R18	-	34	LVCMOS25*
凶	btns_5bits_tri_i[2]	Input	btns_5bits_tri			N15	1	34	LVCMOS25*
\checkmark	btns_5bits_tri_i[1]	Input	btns_5bits_tri			R16	-	34	LVCMOS25*
	btns_5bits_tri_i[0]	Input	btns_5bits_tri			P16	1	34	LVCMOS25*
	Scalar ports (0)								
	🕀 🧓 GPIO_61115 (8)	Input					-	(Multiple)	LVCMOS25*
	Calar ports (0)								
	•				111				
	🗏 Tcl Console 🛛 🗩 Messages 🛛 💐	Log 📄 Rep	oorts 🚯 Design	Runs 👂 Package Pin	s D I/O Por	ts			

Figure 24. Check the IP port pin constraints

Zybo: In the I/O ports tab, expand *GPIO_23532 > btns_4bit_tri_i*, and notice pins have already been assigned to this peripheral. The pin information was included in the board files, and automatically assigned when the IP was connected to the port. The *sw_4bit_tri_i* have also been automatically assigned pin locations, along with the other Fixed ports in the design.



I/O F	Ports								
9	Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Site	Fixed	Bank	I/O Std
	All ports (138)								
	🖽 🕝 DDR_1497 (71)	In/Out					~		(Multiple)*
83-	🕀 🔂 FIXED IO 1497 (59)	In/Out					1		(Multiple)*
8	🖨 🛱 GPIO_23532 (4)	Input					-	34	LVCMOS33*
G	🖻 🤒 btns_4bits_tri_i (4)	Input					1	34	LVCMOS33*
ben	btns_4bits_tri_i[3]	Input	btns_4bits_tri			Y16	1	34	LVCMOS33*
	btns_4bits_tri_i[2]	Input	btns 4bits tri			V16	1	34	LVCMOS33*
2	btns_4bits_tri_i[1]	Input	btns_4bits_tri			P16	-	34	LVCMOS33*
\checkmark	btns_4bits_tri_i[0]	Input	btns_4bits_tri			R18	1	34	LVCMOS33*
	Scalar ports (0)								
	🖨 🙀 GPIO_48824 (4)	Input					1	(Multiple)	LVCMOS33*
	🗄 🤒 sw_4bit_tri_i (4)	Input					1	(Multiple)	LVCMOS33*
	Scalar ports (0)								
	Scalar ports (0)								
							_		

Figure 24. Check the IP port pin constraints

Generate the Bitstream and Export to SDK

Step 5

5-1. Generate bitstream, export the hardware, and start SDK by.

- **5-1-1.** Click on **Generate Bitstream**, and click **Yes**, if prompted, to launch Implementation (Click **Yes** if prompted to save the design).
- **5-1-2.** Select **Open Implemented Design** option when the bitstream generation process is complete and click **OK**. (Click **Yes**, if prompted, to close the synthesized design) You may see a Critical Messages window pop-up. Click **OK** to ignore it.

You should have the block design open before you export the design and launch SDK. If it is closed then open the block design by clicking on the Open Block Design under the IP Integrator sub-menu of the Flow Navigator pane.

5-1-3. Select File > Export > Export Hardware

5-1-4. Make sure that **Include Bitstream** option is selected and click **OK**, leaving the target directory set to local project directory.



Figure 25. Exporting hardware



The lab3.sdk directory will be created under the lab3 project directory. The system_wrapper.hdf file will be placed under the lab3.sdk directory.

5-1-5. Select File > Launch SDK

5-1-6. Click **OK**.

SDK should now be open. A Hardware platform project should have been created, and the *system_wrapper_hw_platform_0* folder should exist in the Project Explorer panel. The system.hdf file for the Hardware platform should be open in the preview pane. Double click system.hdf to open it if it is not.

Basic information about the hardware configuration of the project can be found in the .hdf file, along with the Address maps for the PS systems, and the IPs used information.



Figure 26. SDK C/C++ development view for ZedBoard



ခြဲ Project Explorer 🛛 🦳 🗖	system.hdf 🛛
 □ ≤ ▼ ▲ system_wrapper_hw_platform_ ▲ ps7_init.c 	system_wrapper_hw_platform_0 Hardware Platform Specification
ps7_init.h	Design Information
 ps7_init.ntml ps7_init.tcl system.hdf 	Target FPGA Device: 7z010 Created With: Vivado 2014.2 Created On: Mon Jul 21 13:10:24 2014
	Address Map for processor ps7_cortexa9_0
	btns_4bit 0x41210000 0x4121ffff
	ps7_afi_1 0xf8009000 0xf8009fff
	ps7_afi_2 0xf800a000 0xf800afff
	ps7_coresight_comp_0 0xf8800000 0xf88fffff
	ps7_ddr_0 0x00100000 0x1fffffff ps7_ddrc_0 0xf8006000 0xf8006fff
	ps7_dev_cfg_0 0xf8007000 0xf80070ff
	ps/_dma_ns 0xf8004000 0xf8004fff ps7_dma_s 0xf8003000 0xf8003fff
	ps7 globaltimer 0 0xf8f00200 0xf8f002ff

Figure 26. SDK C/C++ development view for Zybo

Generate TestApp Application in SDK

Step 6

- 6-1. Generate software platform project with default settings and default software project name.
- 6-1-1. From the File menu select File > New > Board Support Package
- 6-1-2. Click Finish with the *standalone* OS selected.
- 6-1-3. Click OK to generate the board support package named standalone_bsp_0.

This will create a new Board Support Package Project

The library generator will run in the background and will create the xparameters.h file in the <2014_2_zynq_labs>\lab3\lab3.sdk\standalone_bsp_0\ps7_cortexa9_0\include directory

- 6-1-4. From the File menu select File > New > Application Project
- 6-1-5. Name the project **TestApp** and in the *Board Support Package* section, select *Use existing* to select **standalone_bsp_0**, and click **Next**



80K New Project					
Application Project					
Create a managed make a	application projec	:t.			
Project name: TestApp					
Use default location					
Location: C:\xup\sys_desi	gn\labs\lab3\lab	3.sdk\TestApp		Browse	
Choose file syste	em: default 💌				
Target Hardware					
Hardware Platform: sys	tem_wrapper_hw_	platform_0		▼ New	
Processor: ps7	_cortexa9_0			•	
Taxaat Cafeyaaa					
larget Software					
Language:	000000				
OS Platform:	standalone			-	
Board Support Package:	Create New	TestApp_bsp			
	Ose existing	standalone_bsp_0		-	
L					
(?)	< Back	Next >	Finish	Cancel	

Figure 27. Application project settings

6-1-6. Select Empty Application and click Finish

This will create a new application project directory, named TestApp, under lab3.sdk directory.

- 6-1-7. Expand TestApp in the project view, and right-click on the src folder, and select Import
- 6-1-8. Expand General category and double-click on File System
- 6-1-9. Browse to <2014_2_zynq_sources>\lab3_zed for ZedBoard or <2014_2_zynq_ sources>\lab3_zybo for Zybo.
- 6-1-10. Select lab3.c and click Finish.

A snippet of the source code is shown in figure below.



```
#include "xparameters.h"
#include "xgpio.h"
//-----
int main (void)
{
   XGpio dip, push;
    int i, psb_check, dip_check;
   xil_printf("-- Start of the Program --\r\n");
   XGpio Initialize(&dip, XPAR SW 8BIT DEVICE ID);
   XGpio_SetDataDirection(&dip, 1, 0xfffffff);
    XGpio_Initialize(&push, XPAR_BTNS_5BIT_DEVICE_ID);
   XGpio_SetDataDirection(&push, 1, 0xfffffff);
    while (1)
    {
     psb_check = XGpio_DiscreteRead(&push, 1);
     xil_printf("Push Buttons Status %x\r\n", psb_check);
     dip_check = XGpio_DiscreteRead(&dip, 1);
     xil_printf("DIP Switch Status %x\r\n", dip_check);
     for (i=0; i<9999999; i++);</pre>
    }
}
```

Figure 28: Snippet of source code

```
system.hdf
            system.mss
                         🔁 lab3.c 🕺
   #include "xparameters.h"
   #include "xgpio.h"
   //-----
  int main (void)
   {
      XGpio dip, push;
       int i, psb_check, dip_check;
       xil_printf("-- Start of the Program --\r\n");
      XGpio Initialize(&dip, XPAR SW 4BIT DEVICE ID);
       XGpio_SetDataDirection(&dip, 1, 0xfffffff);
      XGpio_Initialize(&push, XPAR_BTNS_4BIT_DEVICE_ID);
       XGpio_SetDataDirection(&push, 1, 0xfffffff);
       while (1)
        psb_check = XGpio_DiscreteRead(&push, 1);
        xil_printf("Push Buttons Status %x\r\n", psb_check);
        dip_check = XGpio_DiscreteRead(&dip, 1);
        xil_printf("DIP Switch Status %x\r\n", dip_check);
        for (i=0; i<9999999; i++);</pre>
       }
   3
```

Figure 28. Snippet of source code



Test in Hardware

In case of Zybo, make sure that the JP7 is set to select USB power, so that one cable can be used for both programming and powering up the board.

In case of ZedBoard, you will need two micro-usb cables, one to program the device and another for serial communication.

- 7-1. Connect the board with necessary micro-usb cable(s) and power it ON. Establish the serial communication using SDK's Terminal tab.
- 7-1-1. Zybo only: Make sure that the JP7 is set to select USB power.
- 7-1-2. ZedBoard: Connect the Micro-USB cable to the UART port of the ZedBoard. Turn ON the power.

Zybo: Make sure that a micro-USB cable is connected to the JTAG PROG connector (next to the power supply connector). Turn ON the power.

- 7-1-3. Select the **Preminal** tab. If it is not visible then select **Window > Show view > Terminal**.
- **7-1-4.** Click on M and if required, select Serial as the connection type, select appropriate COM port (depends on your computer), and configure it with the parameters as shown. (These settings may have been saved from previous lab).



Terminal Settings	sok Terminal Settings				
View Settings: View Title: Termina	11				
Encoding: ISO-885	9-1 🔻				
Connection Type:					
Serial	-				
Settings:					
Port: CO	M5 👻				
Baud Rate: 115	200 👻				
Data Bits: 8	-				
Stop Bits: 1	•				
Parity: Nor	ne 🔻				
Flow Control: Nor	ne 🔻				
Timeout (sec): 5					
ОК	Cancel				

Figure 29. Setting up the Terminal tab

- 7-2. Program the FPGA by selecting Xilinx Tools > Program FPGA. Run the TestApp application and verify the functionality.
- 7-2-1. Select Xilinx Tools > Program FPGA
- **7-2-2.** Click **Program** to download the hardware bitstream. When FPGA is programmed, the DONE LED (blue color) will be lit.
- **7-2-3.** Select **TestApp** in *Project Explorer*, right-click and select **Run As > Launch on Hardware (GDB)** to download the application, execute ps7_init, and execute TestApp.elf.
- **7-2-4.** You should see the something similar to the following output on Terminal console.



DIP Switch Status 56 Push Buttons Status 0 DIP Switch Status 56

Figure 30. Program output on ZedBoard

Push Buttons Status 0 DIP Switch Status 6 Push Buttons Status 6 Push Buttons Status 6 Push Buttons Status 0 DIP Switch Status 6 Push Buttons Status 6 Push Buttons Status 6 Push Buttons Status 6 DIP Switch Status 6 DIP Switch Status 6

Figure 30. Program output on Zybo

- **7-2-5.** Select the *Console* tab and click on the *Terminate* button (\blacksquare) to stop the program.
- 7-2-6. Close SDK and Vivado programs by selecting File > Exit in each program.
- 7-2-7. Power OFF the board.

Conclusion

Vivado and the IP Integrator allow base embedded processor systems and applications to be generated very quickly. GPIO peripherals were added from the IP catalog and connected to the Processing System through the 32b Master GP0 interface. The peripherals were configured and external FPGA connections were established. Pin location constraints, since we used the board aware port names, were automatically applied to connect the peripherals to the push buttons and DIP switches of the target board. After the system has been defined, the hardware can be exported and SDK can be invoked from Vivado. Software development was done in SDK which provides several application templates including memory tests. You verified the operation of the hardware by downloading a test application, executing on the processor, and observing the output in the serial terminal window.

