NTU GIEE, MULTIMEDIA SYSTEM-ON-CHIP DESIGN

Lab 1 - Zynq RTL Design Flow

Pin-Hung Kuo

May 11, 2018

1 INTRODUCTION

In this lab, we are going to build a simple Zynq system on ZedBoard. This system works as following: the CPU sends an 8-bits integer value to our IP, then this IP shows the value by 8 PL LEDs on the board. During the lab, you will learn how to build a Zynq block design, how the CPU of Zynq to communicate with peripherals, and how to build an IP with RTL flow. The following steps will lead you to build such a system.

2 CREATING IP IN RTL





Open Vivado, select Create New Project as Fig.2.1.

Click **Next** in *New Project* dialogue. At the Project Name dialogue, enter **Lab1** as the **Project name** and **C:/MSOC**(the directory where you unzip the lab file) as **Project location**. Make sure that the **Create project subdirectory** is ticked. Please refer to Fig.2.2 for the above step. Click **Next**. Before next step, please copy the **source** folder in the lab file to **C:/MSOC**/.

🝌 New Project						×
Project Name						
Enter a name for y	our project and specify a dire	ctory where the pro	ject data files will b	e stored.		*
Project name:	Lab					\otimes
Project location:	C:/MSOC					
✓ Create proje	ect subdirectory					
Project will be c	reated at: C:/MSOC/Lab1					
?			< Back	<u>N</u> ext >	<u>F</u> inish	Cancel

Figure 2.2

In **Project Type**, select **RTL Project** and do not specify sources as shown in Fig.2.3.

🝌 Nev	v Project	×
Proje Specify	tect Type by the type of project to create.	4
۲	<u>R</u> TL Project You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.	
	Do not specify sources at this time	
	Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.	
	Do not specify sources at this time	
0	J/O Planning Project Do not specify design sources. You will be able to view part/package resources.	
0	Imported Project Create a Vivado project from a Synplify, XST or ISE Project File.	
0	Example Project Create a new Vivado project from a predefined template.	
?	< Back Einish	Cancel

Figure 2.3

Select **Verilog** as **Target language** in the *Add Sources* dialogue as in Fig.2.4. In *Add Existing IP(optional)* dialogue, we can add existing IP here. As we do not have that, click **Next**.

The *Add Constraints(optional)* dialogue will open. This is the stage were any physical or timing constraints files could be added to the project. As we do not have that, click **Next**.

The *Default Part* dialogue will open. Select **Boards** from the *Select dialogue* and **ZedBoard Zynq Evaluation and Development Kit** as Fig.2.5.

Target language: Verilog 👻 Simulator language: Mixed 👻

Figure 2.4

New Project			×
Default Part Choose a default Xlinx part or board for your project. This can be c	hanged later.		4
Parts Boards			
Reset All Filters Vendor: All Value Name: All			
Search: Q- V			
DisplayName	Preview	Vendor	File
ZedBoard Zyng Evaluation and Development Kit Add Daughter Card Connections		em.avnet.com	1.4
Artix-7 AC701 Evaluation Platform Add Daughter Card Connections		xilinx.com	1.4∨ →
$(\mathbf{\hat{o}})$	< Back Next >	<u>F</u> inish	Cancel

Figure 2.5

Click **Finish** to create the project.

From the menu bard, select Tools>Create and Package New IP..., as shown in Fig.2.6

Tools	Rep <u>o</u> rts	Window	Layout	View	<u>H</u> elp
	Create and	Pac <u>k</u> age Ne	w IP		
	Create Inter	face Definition	on		
	Enable Part	ial Reconfig	uration		
1	Run Tcl Scr	ipt			
	Property Ed	itor			Ctrl+J
	Associate E	L <u>F</u> Files			
te	Generate M	emory Config	guration File	ə	
	Compile Sir	nulation Lib	aries		
	Xilinx Tcl Sto	ore			
a	Custom Co	mmands			•
2	Language <u>1</u>	emplates			
•	Settings				

Figure 2.6

The Create and Package New IP dialogue will launch, Click Next.

Then, select Create new AXI4 peripheral here and click Next 2.7.

🙏 Create and	d Package New IP	×
Create Pe Please selec	ripheral, Package IP or Package a Block Design ctone of the following tasks.	4
Packag	ing Options	
	Package your current project Use the project as the source for creating a new IP Definition.	
	Package a block design from the current project Choose a block design as the source for creating a new IP Definition.	
	Package a specified directory Choose a directory as the source for creating a new IP Definition.	
Create	AXI4 Peripheral	
۲	Create a new AX4 peripheral Create an AX4 IP, driver, software test application, IP Integrator AX4 VIP simulation and debug demonstration design.	
?	< Back Next > Finish C	ancel

Figure 2.7

Enter **led_controller** as the **Name 2.8**.

🝌 Create and Packa	ge New IP				×
Peripheral Deta Specifyname, versio	is n and description for the new peripheral				4
Name:	led_controller				\otimes
Version:	1.0				\otimes
Displayname:	led_controller_v1.0				\otimes
Description:	My new AXI IP				\otimes
IP location:	C:/MSOC/Lab1//ip_repo				S
Overwrite ex	sting				
?		< <u>B</u> ack	<u>N</u> ext >	<u>F</u> inish	Cancel

Figure 2.8

Click Next.

The *Add Interface* dialogue allows you to specify the AXI4 interface(s) that will be present in your custom peripheral. Here you can specify:

- Number of interfaces
- Interface type (AXI-Lite, AXI-Stream or AXI-Full)
- Interface mode(slave or master)
- Interface data width

Features specific to individual interface types will also be available when the corresponding type is selected.

As our peripheral is a simple controller for the LEDs, which only requires single values to be transferred to it, an AXI-Lite slave interface is sufficient. Only one memory mapped register is required for our simple controller, but as the minimum number that can be specified in the dialogue is 4, we will chose that. Specify the *Add Interface* dialogue as shown in Fig.2.9.

🍌 Create and Package New IP				×
Add Interfaces Add AV4 interfaces supported by your periph	eral			A
Enable Interrupt Support	+ -	Name	S00_AXI	\otimes
	lnterfaces	Interface Type	Lite	~
	S00_AXI	Interface Mode	Slave	~
		Data Width (Bits)	32	~
	<	Memory Size (Bytes)	64	~
SOO_AXI	>	Number of Registers	4	[4512]
led_controller_v1.0				寄存器的救量
(\mathbf{s})		< <u>B</u> ack	<u>N</u> ext > <u>F</u> ini	sh Cancel

Figure 2.9

🙏 Create and Package New	IP	×
	Create Peripheral	
HLx Editions	Peripheral Generation Summary	<u>^</u>
	1. IP (xilinx.com:user:led_controller:1.0) with 1 interface(s)	
	2. Driver(v1_00_a) and testapp more info	
	3. AXI4 VIP Simulation demonstration design more info	
	4. AXI4 Debug Hardware Simulation demonstration design more info	
	Perinheral created will be available in the catalog -	
	C:/MSOC/Lab1/./jp repo	
	Next Steps:	
	Add IP to the repository	
	Edit IP	
	Verde Barde hand ID up for AV44 MD	
	Verify Peripreta iP using XX4 ViP	
E XILINX	Click Finish to continue	~
ALL PROGRAMMABLE.		
0		
?	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish	Cancel

Figure 2.10

Review the information in the *Create Peripheral* dialogue, which details the output files which will be created. Select the option to **Edit IP** 2.10. This will create the IP peripheral files and create a new Vivado project where the functionality of the peripheral can be modified in the source HDL code, and the packaged. Click **Finish** to close the *Wizard* and create the peripheral template.

A new Vivado project, named **edit_led_controller_v1_0**, will open.

In the *Sources* pane, you should see two HDL source files (you may need to expand the file selection):



Figure 2.11

The two source files are:

- **led_controller_v1_0.v** This file instantiates all AXI-Lite interfaces. In this case, only on interface is present.
- **led_controller_v1_0_S00_AXI.v** This file contains the AXI4-Lite interface functionality which handles the interactions between the peripheral in the PL and the software running on the PS.

The IP Packager pane will also be open in the Workspace:

Project Summary × Package IP - led	d_controller ×		ΟĽ
Packaging Steps	Identification		
✓ Identification	Vendor:	xilinx.com	\otimes
✓ Compatibility	Library:	user	\otimes
✓ File Groups	Name:	led_controller	\otimes
 Customization Parameters 	Version:	1.0	\otimes
 Ports and Interfaces 	Display name:	led_controller_v1.0	\otimes
 Addressing and Memory 	Description:	My new AXI IP	\otimes
Customization GUI	Vendor display name:		
Basian and Baskaga	Company url:		
 Review and Package 	Root directory:	c/MSOC/ip_repoiled_controller_1.0	
	Categories	1	
	AXI_Peripheral	Y	

Figure 2.12

The information that we specified about our peripheral in previous steps will be visible.

We can now add the functionality to our **led_controller** peripheral. We will be adding a new output port the peripheral template to allow it to connect to the LED pins on the Zynq device, as well as assigning the value received from the Zynq PS to the new output port.

Open **led_controller_v1_0_S00_AXI.v** by double-click on it in the *Sources* pane. The file will be opened in the Workspace.

Add the declaration



as shown in Fig.2.13.



Figure 2.13

And connect this output port to register slv_reg0

```
assign LEDs_out = slv_reg0[7:0];
```

as Fig.<mark>2.14</mark>.

```
395
                     axi_rdata <= reg_data_out;
                                                 // register read data
396 白
                   end
397 🗀
              end
398 白
           end
399
400
           // Add user logic here
          assign LEDs_out = slv_reg0[7:0];
401 ;
           // User logic ends
402
403
           endmodule
404 🗀
405
```



Save the file by File>Save File or Ctrl+s.

Open **led_controller_v1_0.v**. We must once again create a new output port to the top-level source file, and map it to the equivalent port that we created in the AXI4-Lite interface file in the previous step.

As in led_controller_v1_0_S00_AXI.v, we add the declaration of LEDs_out again

```
output wire [7:0] LEDs_out,
```

as shown in Fig.2.15.

15)	
16	(
17		// Users to add ports here
18		output wire [7:0] LEDs_out,
19 🖯		// User ports ends
20		// Do not modify the ports beyond this line
21		
22		
23 🏳		// Ports of Axi Slave Bus Interface S00_AXI

Figure 2.15

and connect to the port by

.LEDs_out(LEDs_out)

in the instance of **led_controller_v1_0.v**. Fig.<mark>2.16</mark> shows this step. **NOTE:** You should add a "," in the end of the port before LEDs_out.

64	.S_AXI_ARADDR(sOO_axi_araddr),	
65	.S_AXI_ARPROT(sOO_axi_arprot),	
66	.S_AXI_ARVALID(sOO_axi_arvalid),	
67	.S_AXI_ARREADY(sOO_axi_arready),	
68	.S_AXI_RDATA(sOO_axi_rdata),	
69	.S_AXI_RRESP(sOO_axi_rresp),	
70	.S_AXI_RVALID(sOO_axi_rvalid),	
71	.S_AXI_RREADY(sOO_axi_rready);	
72	.LEDs_out(LEDs_out)	
73);	
74		
75 🖯	// Add user logic here	
76		
A		

Figure 2.16

Return to **IP Packager** by selecting the **Package IP - led_controller**. IP Packager will detect the changes to the source files, and the areas which need refreshed will be highlighted

with the following icon: 🖉. You should see that the following three areas of interest need

File Groups

refreshed:

- Customization Parameters
- Ports and Interfaces

Select **Customization Parameters** in the *Packager* pane. You should see the following information message as the top of the pane:

Merge changes from Customization Parameters Wizard

Click **Merge changes from Customization Parameters Wizard** to update the IP Packager information to the changes made in the HDL source files. Perform the similar process to **File**

Groups to eliminate all *icon* except for **Review and Package**.

To verify that IP Packager has updated the Ports and Interfaces area, we will open it and check.

Select Ports and Interfaces from the IP Packager pane.

You should notice that the LEDs_out port that we added to the source files has been added to the IP Ports pane and has a length of 8:

The final step in creating our new IP peripheral is to package the IP. Select **Review and Package** from the **IP Packager** pane.

In the *After Packaging* panel, click **edit packaging settings** at the bottom:

```
After Packaging

    Create archive of IP - C:/MSOC/ip_repo/led_controller_1.0/xilinx.com_user_led_controller_1.0.zip
    edit
    Project will be removed after completion
    edit packaging settings
```

Figure 2.17

In the *Automatic Behaviour* panel, enable the option to *Create archive of IP, Close IP Pack-ager window* and *Add IP to the IP Catalog of the Current Project*. You may *Delete project after packaging* if you wish (but the Verilog files can still be found even you select this option)

Project Settings	IP > Packager Specify settings related to IP Packager.		1
General			
Simulation	Default Values		
Elaboration	The following volues will be sutemptically and	lind offer finishing	
Synthesis	the IP Packager Wizard.		
Implementation	Vendor: vilinx com		
Bitstream			
√ IP	Library: user	\otimes	
Repository	Category: /UserIP	\otimes	
Packager			
and Sattings	IP location:/ip_repo	8	
Project	Automatic Robavior		
IP Defaults			
Source File	After Packaging		
Display	Create archive of IP		
WebTalk			
Help	Add IP to the IP Catalog of the current p	project	
> Text Editor	Close IP Packager window		
3rd Party Simulators	Include Source project archive		
> Colors	;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;		
Selection Rules	Edit IP in IP Packager		
Shortcuts	Delete project after packaging		
> Strategies			
> Window Behavior	File Extensions to Filter on Add Directory		
	Create a list of file extensions that will be automat	ically filtered when	

Figure 2.18

Click **OK** to apply the setting. Review the information provided in the *Review and Package* window, and click *Re-Package IP*. A dialogue box will appear asking if you want to close the project, click **Yes**. The changes made to the IP peripheral will be included in the repackaged IP, and the Vivado project will close.

3 CREATE SYSTEM BLOCK

We will return to our original Vivado project Lab1 after we finish the IP creation.

To start, we will create a new Block Design and add the IP peripheral which we just created to the design. In the *Flow Navigator* window, select **Create Block Design** from the *IP Integrator* section. Enter **lab_1** in the *Design name* box, and click **OK** to create the blank design. Now, we want to add IP in the *Vivado IP Integrator Diagram canvas*. This can be achieved by 3 methods:

• right click anywhere and select Add IP as Fig.3.1

- Hot key Ctrl+i
- The tool bar at the left of the canvas as Fig.3.2

	Properties	Ctrl+E
×	Delete	Delete
	Сору	Ctrl+C
1.00	Paste	Ctrl+V
Q,	Search	Ctrl+F
Vite	Select All	Ctrl+A
+	Add IP	Ctrl+I
	Add Module	
	Pinning	Þ
	IP Settings	
2	Validate Design	F6
	Create Hierarchy	
	Create Comment	
	Create Port	Ctrl+K
	Create Interface Port	Ctrl+L
C	Regenerate Layout	
	Save as PDF File	



Figure 3.2: Tool bar

Enter **led** in the *Search* box, and double-click **led_controller_v1.0** to add an instance of the LED controller IP to the design. Fig.3.3 shows the instance of our IP. To enable the peripheral to connect to the LEDs on the ZedBoard, we must make the LEDs_out port external. This allows the output port to be connected to specific physical pins on the Zynq device, which are connected to the LEDs. Right click on the port **LEDs_out[7:0]** and select **Make External**.



Figure 3.3: led_controller block

The block design should new resemble Fig.3.4.



Figure 3.4: led_controller block with external port

The next step is to add a Zynq Processing System block, which stands for the PS part on the ZedBoard. Add an instance of **Zynq7 Processing System** by the similar way as we add the IP led_controller_v1.0. Type **zynq** in the search box to add the **Zynq7 Processing System**. The *Designer Assistance* message at the top of the canvas will appear:

* Designer Assistance available. Run Block Automation Run Connection Automation

Click **Run Block Automation**. An information message will appear. Ensure that **Apply Board Preset** is selected, and click **OK**. This will make all necessary modifications to the Zynq processing system that relate to the board preset and make required external connections.

We must now connect the LED Controller to the Zynq Processing System. This step can also be carried out using Designer Assistance.

In the *Designer Assistance* message, click **Run Connection Automation**. An information message will appear, select **led_controller_0/S00_AXI** and click **OK**. This will add some additional blocks to the design which are require to connect the LED Controller to the Zynq Processing System.

Our block design in now complete. Validate the design by selectin **Tools>Validate Design** from the *Menu Bar*, or select **Validate Design** in right-click menu, or just press keyboard **F6**. If we pass the validation, our block system is done.

Now we should generate the HDL files for the design.

In the *Sources* pane, find **lab_1(lab_1.bd**) under **Design Sources(1)**. Right-click on it and select **Create HDL Wrapper**. Select **Let Vivado manage wrapper and auto-update** as Fig.3.5 and click **OK**. This will create the top-level HLD file for the design.

🝌 Create HDL Wrapper	×
You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.	4
Options	
O Copy generated wrapper to allow user edits	
(e) Let Vivado manage wrapper and auto-update	
? ОК Саг	cel

Figure 3.5: Create HDL Wrapper

We must now connect the LEDs_out port of the design to the correct pins on the Zynq device. This is done through the specification of constraints in an XDC file.

In the *Flow Navigator* window, select **Add Sources** from the *Project Manager* section. The *Add Sources* dialogue will open. Select **Add or Create Constraints**, and click **Next**. Click the

below		
7		
	below	below

Figure 3.6: Add or Create Constraints Dialogue Window

The *Create Constraints File* dialogue will open. Select **XDC** as the *File type* and enter **led_constraints** as the *File name*. Click **OK**.

Click **Finish** to create the file and close the dialogue.

In the **Sources** tab, expand the **Constraints** entry and open the newly created XDC file by double-clicking on **led_constraints.xdc**. The file will open in the *Workspace*. Add the following constraints into the file. You can find the same constraints code in **c:/MSOC/source/Lab1/ led_constraints.xdc**.

```
set_property PACKAGE_PIN T22 [get_ports {LEDs_out_0[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[0]}]
set_property PACKAGE_PIN T21 [get_ports {LEDs_out_0[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[1]}]
set_property PACKAGE_PIN U22 [get_ports {LEDs_out_0[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[2]}]
set_property PACKAGE_PIN U21 [get_ports {LEDs_out_0[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[3]}]
set_property PACKAGE_PIN V22 [get_ports {LEDs_out_0[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[4]}]
set_property PACKAGE_PIN W22 [get_ports {LEDs_out_0[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[5]}]
set_property PACKAGE_PIN U19 [get_ports {LEDs_out_0[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[6]}]
set_property PACKAGE_PIN U14 [get_ports {LEDs_out_0[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs_out_0[7]}]
```

This constrains connect each individual bit of the LEDs_out port to a specific pin on the Zed-Board and specify the voltage of these pins.

Save the XDC file. Now, we have already finished our system architecture. The next step is to put our design on the ZedBoard. Select **Generate Bitsream** in *Flow Navigator*. If a dialogue appears prompting you to save your design, click **Save**. A dialogue window may requesting that you launch synthesis and implementation before starting the *Generate Bitstream* process. Click **Yes** if the request appears. It may take several minutes for synthesis, implementation and bitstream generation, you can take a break.

When bitstream generation is complete, a dialogue window will open to inform you. Select **Open Implemented Design**, and Click **OK**. You can observe how the FPGA resources are used by your design here.

Now, we just finished hardware part of our design. The next step is to write our software executed on the Zynq ARM Cortex-A9 CPU.

Select **File>Export>Export Hardware...** from the *Menu Bar*. The *Export Hardware for SDK* dialogue window will open. Ensure that the option to **Include bitstream** is selected, and click **OK**. Launch the SDK in Vivado by selecting **File>Launch SDK** from the *Menu Bard* and click **OK**.

The SDK will launch.

3.1 SOFTWARE APPLICATION

Once the SDK has launched, create a new **Application Project** by selectin **File** > **New** > **Application Project** from the *Menu Bar*. In the *New Project* dialogue, enter **Lab1** as shown in Fig.3.7. You can notice that at the bottom of *New Project* dialogue is **Board Support Pack-age(BSP)**. We choose **Create New** here to let SDK generate the necessary support package (or drivers) depending on our hardware in the previous steps. We can also generate a full BSP by **File** > **New** > **Board Support Package** before we create a new application project. Click **Next**> and select **Empty Application** in *Available Template* window as Fig.3.8. Click **Finish**.

New Project	—	<					
Application Project Create a managed make application project.							
Project name: Lab1							
Use default locatio	n						
Location: C:\MSOC\L	ab1\Lab1.sdk\Lab1 Browse						
Choose file s	system: default 🗸						
OS Platform: standa	lone ~						
Target Hardware							
Hardware Platform:	lab_1_wrapper_hw_platform_0 v New						
Processor:	ps7_cortexa9_0 ~						
Target Software							
Language:	● C ○ C++						
Compiler:	32-bit \vee						
Hypervisor Guest:	N/A \vee						
Board Support Packa	ge: Create New Lab1_bsp						
	\bigcirc Use existing \lor						
?	< Back Next > Finish Cancel						

Figure 3.7: Create Application Project



Figure 3.8: Empty Application

Although we get most drivers when SDK create BSP for us, the driver of the IP created by ourself cannot be generated by SDK. So we have to add this first.

Navigate to **Xilinx > Repositories** in *Menu Bar*. In the *Repositories Preferences* windows, click on **New**, as shown in Fig.3.9.

be filter text	Add, remove or change the order of SDK's software repositories.	(> ▼ ⊂> ▼
General	Local Repositories (available to the current workspace)	
C/C++		
Install/Lindate		New
Remote Development		Remove
Remote Systems		Llo
Run/Debug		op
Team		Down
Terminal		Relative
		Relative
Boot Image	Global Repositories (available across workspaces)	
BSP Preferences		New
Flash Programming Hardware Specificatio		Remove
Log Information Level		Up
SDK Capabilities		Down
Toolchain Preferences	SDK Installation Repositories	
	C·/Xiliny/SDK/2018 1/data/embeddedsw	
	Rescan Repositories	
	Note: Local repository settings take precedence over global repository settings.	
	Restore Defaults Apply	

Figure 3.9: SDK Repository Peripherals window

Browse to the directory: **c:/MSOC/ip_repo/led_controller_1.0** as Fig.3.10.

瀏覽資料夾	\times
Choose a repository directory. A repository directory typically contains the 'drivers', 'bsp' or 'sw_services' sub-directories.	
> MinGW	^
V MSOC	
✓ ip_repo	
Ied_controller_1.0	
bd	
> drivers	
> example_designs	
hdl	
xgui	~
資料夾(F): led_controller_1.0	
建立新資料夾(M) 確定 取消	

Figure 3.10: led_controller repository selection

The **system.mss** tab should be open in the Workspace. If it is not, open it by expanding **Lab1_bsp** in *Project Explorer* and double-clicking on **system.mss**.

At the top left of the **system.mss** tab, click **Modify this BSP's Settings** Fig.3.11.

🕼 system.hdf 🛛 👔 system.mss 🕴	
Lab1_bsp Board Support Package	^
Modify this BSP's Settings Re-generate BSP Sources	
This Barry Support Backage is compiled to up on the following target	
Ins cours support socially a complete to informate tomorning angle: Hardware Specification: C:MSOLLabT(LabTLabTLabL_Iwrapper_hw_platform_0)system.hdf Target Processor: ps7_cortexa9_0	
Operating System	
Board Support Package OS.	
Name: standalone Version: 6.6	
Description: Standaione is a simple, low-level software layer, it provides access to basic processor retatures such as caches, interrupts and exceptions as well as the basic feature of a hosted environment, such as standard input and output, profiling, abort and exit.	es
Documentation: standalone v6.6	
Peripheral Drivers	
Drivers present in the Board Support Package.	
led_controller_0 led_controller	
ps7_afi_0 generic	
ps7_afi_1 generic	
ps7,afi2 generic	
ps/,an_3 genenc	
ps/_coresignt_comp_U_coresigntps_acc_Documentation	
ps/_dor_0_dorps <u>Documentation</u>	
ps/ dar_C generic	
ps/_dee_rdg_0_deerdg	
ps/_uma_ns_umaps	
ps/ uma s umaps <u>Documentation import examples</u>	
ps/guernet_0 enacts Documentation import examples	
ps/_globalumer_o generic	
ps: jpp-0 generic	
ps7 inc bis confit of generic	
ps7_l2cachec_0_generic	
Overview Source	~

Figure 3.11

The Board Support Package Settings window will open. Select **driver** from the left-hand menu. From the list of components in the *Drivers* pane, identify **led_controller_0** and ensure **led_controller** is selected from the drop-down menu in the **Driver** column, as shown in Fig.3.12.

Board Support Package Settings

Board Support Package Settings

Control various settings of your Board Support Package.

Component	Component Tune	Driver	Dri	
component	component type	Diver	2.6	
ps/_contexa9_0	ps/_contexa9	cpu_contexa9	2.6	
led_controller_0	led_controller	led_controller	1.0	
ps/_ati_0	ps7_ati	generic	2.0	
ps/_afi_1	ps7_ati	generic	2.0	
ps/_ati_2	ps/_ati	generic	2.0	
ps/_ati_3	ps/_ati	generic	2.0	
ps/_coresignt_comp_0	ps7_coresignt_comp	coresigntps_dcc	1.4	
ps/_ddr_0	ps/_ddr	ddrps	1.0	
ps/_ddrc_0	ps7_ddrc	generic	2.0	
ps/_dev_ctg_0	ps/_dev_ctg	aevctg	3.5	
ps/_dma_ns	ps/_dma	amaps	2.3	
ps/_dma_s	ps/_dma	amaps	2.3	
ps/_ethernet_0	ps/_ethernet	emacps	3.7	
ps7_globaltimer_0	ps7_globaltimer	generic	2.0	
ps/_gpio_0	ps/_gpio	gpiops	3.3	
ps7_gpv_0	ps7_gpv	generic	2.0	
ps7_intc_dist_0	ps7_intc_dist	generic	2.0	
ps7_iop_bus_config_0	ps7_iop_bus_config	generic	2.0	
ps7_l2cachec_0	ps7_l2cachec	generic	2.0	
ps7_ocmc_0	ps7_ocmc	generic	2.0	
ps7_pl310_0	ps7_pl310	generic	2.0	
ps7_pmu_0	ps7_pmu	generic	2.0	
ps7_qspi_0	ps7_qspi	qspips	3.4	
ps7_qspi_linear_0	ps7_qspi_linear	generic	2.0	
ps7_ram_0	ps7_ram	generic	2.0	
ps7_ram_1	ps7_ram	generic	2.0	
ps7_scuc_0	ps7_scuc	generic	2.0	
ps7_scugic_0	ps7_scugic	scugic	3.9	
ps7_scutimer_0	ps7_scutimer	scutimer	2.1	
ps7_scuwdt_0	ps7_scuwdt	scuwdt	2.1	
ps7_sd_0	ps7_sdio	sdps	3.4	
ps7_slcr_0	ps7_slcr	generic	2.0	
ps7_ttc_0	ps7_ttc	ttcps	3.5	
ps7_uart_1	ps7_uart	uartps	3.6	
ps7_usb_0	ps7_usb	usbps	2.4	
ps7_xadc_0	ps7_xadc	xadcps	2.2	

Figure 3.12: led_controller repository selection

Click OK.

Now we prepare all the necessary drivers for our system. The last step is to prepare our software to be run on CPU.

In fact, this software is already written for you. In *Project Explorer*, right click on **Lab1** and select **import**. An *Import* window will be opened, expand **General** and double-click on **File System** as Fig.3.13.

Sox Import	_		×
Select Import resources from the local file system into an existing project.		Ľ	5
Select an import wizard:			
type filter text			
 ✓ ➢ General Marchive File ➢ Existing Projects into Workspace File System Projects from Folder or Archive ➢ C/C++ ➢ Git ➢ Install ➢ Remote Systems ➢ Run/Debug ➢ Team ➢ Tracing 			
C Kack Next > Finish		Cancel	

Figure 3.13: Import file

Click **Next**. Type **c:/MSOC/Lab1/source/** in *From directory* box or click **Browse...** to find this route. Select **led_controller.c** as in Fig.3.14.

sok Import					×
File system Import resources from the lo	cal file system.				
From directory: C:\MSOC\L	ab1\source		~	Browse	
source			aints.xdc] Jller_v1_0_S00_AXI. Jller_v1_0.v Jller.c	v	
Filter Types Select A	All Deselect All				
Into folder: Lab1				Browse	
Options Overwrite existing resour Create top-level folder Advanced >>	rces without warning				
?	< Back	Next >	Finish	Cancel	I

Figure 3.14: led_controller.c

Click Finish.

Before we execute our software, have our ZedBoard ready to connect to PC. Plug the power cable, connect **J17** and **J14** ports on the ZedBoard to PC with mirco-USB cables. The **J14** port is used to communicate between PC and ZedBoard. Turn on the ZedBoard power now. If your PC asks you for driver installation, select **Cypress CY7C64225 chipset**. If there is any problem in installation of the driver, please refer to Cypress Chipset.

Now you need to open the terminal in SDK. Type **terminal** in the *Quick Access* box at the top-right as Fig.3.15 to add a terminal tab.

	terminal				
Views	📮 SDK Terminal (Xilinx)				
	🖉 Terminal (Terminal)				
	Terminals (Remote Systems)				
Commands	Launch Terminal				
	Preferences (Terminal) - Open the				
	Show In (SDK Terminal)				
	Show In (Terminal)				
	Show In (Terminals)				
	Show View (SDK Terminal) - Shows				
	Show View (Terminal) - Shows a pa				
Preferences	Terminal				
Results per category are limited. Press 'Ctrl+3' to see all					

Figure 3.15: Quick Access

Find the **Terminal** tab as shown in Fig.3.16, it may appear at the bottom, bottom-right or topright of SDK window. Click the \checkmark to connect the PC and the board through the UART. The **Terminal Settings** dialogue will appears, select your UART port (default COM3) and set the other options as Fig.3.17

🗐 SDK Log 🥒 Terminal 1 🙁	- × × 🐿 🖬 🚮 🖉 × 📽 🕶 🛙	3
No Connection Selected	Connect icon	
		1
		~

Figure 3.16: Terminal

Terminal Settings X						
View Settings: View Title: Terminal 1 Encoding: ISO-8859-1 ~ Connection Type:						
Serial ~						
Settings:						
Port:	COM3	~				
Baud Rate:	115200	~				
Data Bits:	8	~				
Stop Bits:	1	~				
Parity:	None	~				
Flow Control:	None	~				
Timeout (sec):	5					
OK Cancel						

Figure 3.17: Terminal Settings

Click **OK** to initiate the new Terminal connection.

Before we execute our application, we have one last thing to do: **Program FPGA** by the bitstream generated by Vivado. Select **Xilinx > Program FPGA** from the *Menu Bar*. Click **Program**.

After the bitstream being loaded into the FPGA, the blue LED **LD12** on the board will be turned on. We can execute our last step now.

Right click on the **Lab1** in the **Project Explorer** in the left-hand pane. Select **Run As > Launch on Hardware(GDB)** as shown in Fig.3.18.

🚾 Lab1.sdk - C/C++ - Lab1_bsp/system.mss - Xilinx SDK

File E	dit I	Navigate Search Project Ru	ın Xilinx Win	dow	Help	
📑 🗖		🗑 - 🍕 - 👼 - 📀	▼ 🔌 💽 4	9] 🖬 🕃 🛷 🕇	*
Proj	ect Ex	(plorer 🛛	E	€ ₽	▼ ▽ □ □	system.hdf
v 🕘	lab_1	_wrapper_hw_platform_0				Lab1 bsn Roard Sunnort Packag
>	🗁 dr	ivers h 1 wanner hit				Lab 1_DSP Board Support lackag
		s7_init_gpl.c				Modify this BSP's Settings Re-generat
	🔓 ps	7_init_gpl.h				
	🔮 ps	s7_init.c s7_init.h				Target Information
	🍈 ps	7_init.html				This Board Support Package is compiled t
	ps 📄	7_init.tcl				Hardware Specification: C:\MSOC\Lab1\
> P9	∎ SY ah1	stem.hdf				Talget Hotessol. psr_collexas_o
5 👧	i	New	>	1		Operating System
		Go Into				Board Support Package OS.
		Open in New Window				Name: standalone
	P	Сору	Ctrl+C			Description: Standalone is a simple,
	ß	Paste	Ctrl+V	L .		of a hosted environmer
	×	Delete	Delete	L .		Documentation: standalone v6 6
		Source	>	L .		Peripheral Drivers
		Move		L .		Drivers present in the Board Support Pack
		Rename	F2			led_controller_0 led_controller
	è	Import		L .		ps7_afi_0 generic
	2	Export				ps7_afi_2 generic
		Build Project		L .		ps7_afi_3 generic
		Clean Project		L .		ps7_coresight_comp_0_coresightps_dcc [
		Refresh	F5	L .		ps7_ddr_0_ddrps
		Close Project				ps7_dev_cfg_0 devcfg
		Close Unrelated Projects				ps7_dma_ns dmaps I
		Build Configurations	>			ps7_dma_s_dmaps [
		Run As	>	E	1 Launch on Hardy	vare (System Debugger)
		Debug As	>		2 Start Performance	e Analysis
		Compare With	>	E TEF	3 Launch on Hardy	vare (System Debugger on QEMU)
		Restore from Local History		GDB	4 Launch on Hardy	vare (GDB)
		C/C++ Build Settings		С	5 Local C/C++ Ap	plication
	2	Generate Linker Script		_	Run Configuration	S
- Tan		Change Referenced BSP		F	2 P P R	
		Team	>		-u U	Program FPGA
> 6	I	Configure				
> 🗁	(Configure	>			
		Properties	Alt+Enter			

Figure 3.18: Run Application on hardware

This application sends an 8-bits integer value to PL part, and the LEDs on the board show this value. The value will also displayed on the terminal as Fig.3.19. The LEDs turn on or off

depending on the corresponding bit.

 N°
 Image: I

Figure 3.19: Run Application on hardware