Vivado Design Flow

Introduction

This lab guides you through the process of using Vivado IDE to create a simple HDL design targeting the Zynq device. You will simulate, synthesize, and implement the design with default settings. Finally, you will generate the bitstream and download it in to the hardware to verify the design functionality

Objectives

After completing this lab, you will be able to:

- Create a Vivado project sourcing HDL model(s) and targeting a specific FPGA device located on the ZedBoard or Zybo
- Use the provided Xilinx Design Constraint (XDC) file to constrain the pin locations
- Simulate the design using the Vivado simulator
- Synthesize and implement the design
- Generate the bitstream
- Configure the FPGA using the generated bitstream and verify the functionality

Procedure

This lab is broken into steps that consist of general overview statements providing information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab1.

Design Description

The design consists of some inputs directly connected to the corresponding output LEDs. Other inputs are logically operated on before the results are output on the remaining LEDs as shown in **Figure 1**.

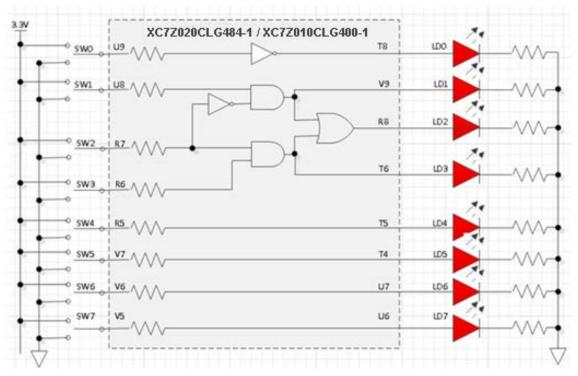
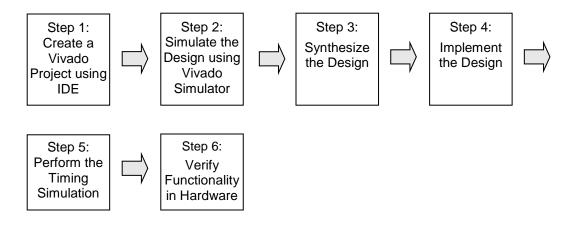


Figure 1. The Completed Design



General Flow



Create a Vivado Project using IDE

Step 1

1-1. Launch Vivado and create a project targeting the appropriate Zynq device and using the Verilog HDL. Use the provided lab1.v and lab1.xdc. The files are added to the project from the <2014_2_zynq_sources>\<board>Vab1 directory.

References to <2014_2_zynq_labs> is a placeholder for the c:\xup\fpga_flow\2014_2_zynq_labs directory and <2014_2_zynq_sources> is a placeholder for the c:\xup\fpga_flow\2014_2_zynq_sources directory.

Reference to **<board>** means either the **ZedBoard** or the **Zybo**.

- 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 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 **lab1** in the *Project name* field. Make sure that the *Create Project Subdirectory* box is checked. Click **Next**.



🚴 New Project	x
Project Name	
Enter a name for your project and specify a directory where the project data files will be stored	
Project name: lab1	8
Project location: C:/xup/fpga_flow/2014_2_zyng_labs	
Create project subdirectory	
Project will be created at: C:/xup/fpga_flow/2014_2_zynq_labs/lab1	
< <u>Back</u> <u>N</u> ext > Einish Car	ncel

Figure 2. Project Name and Location entry

- 1-1-5. Select RTL Project option in the Project Type form, and click Next.
- **1-1-6.** Select **Verilog** as the *Target Language* in the *Add Sources* form. Select **Verilog** as the *Simulator Language*.
- 1-1-7. Click on the Add Files... button, browse to the <2014_2_zynq_sources> directory, double click the appropriate \<board>\lab1\ subdirectory and select lab1.v, click OK. Make sure the source file is copied into your project directory (via the check box) and then click Next.
- 1-1-8. Click Next again at the Add Existing IP (optional) form to get to the Add Constraints form.
- **1-1-9.** The constraints file *lab1_zedboard.xdc* or *lab1_zybo.xdc* has automatically been added depending on the board directory previously selected while adding sources. Click **Next.**

This Xilinx Design Constraints file assigns the physical IO locations on FPGA to the switches, buttons and LEDs located on the board. This information can be obtained either through the board's schematic or the board's user guide.

1-1-10. In the Default Part form, using the Parts option and various drop-down fields of the Filter section (as seen in the screen shot below), select the XC7Z020CLG484-1 device for the ZedBoard or the XC7Z010CLG400-1 device for the Zybo.



🚴 New Project								x		
Default Part Choose a default Xil	inx part or board for	^r your project. T	'his can be chan	ged later.						
- Specify	odu <u>c</u> t category <mark>,</mark> Gen Eamily, Zyn S <u>u</u> b-Family, Zyn	q-7000	Res	👻 Spee <u>d</u> g	kage clg484 Irade -1 Irade C		* * *			
<u>S</u> earch: Q→										
Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceivers	GTPE Trans		
🔊 xc7z020clg484-1	484	200	53200	106400	140	220	0	0		
For the ZedBoard										
				< <u>B</u>	ack <u>N</u> e	xt > Fi	nish Car			

Figure 3. Part Selection to select the XC7Z020-1CLG484C device for the ZedBoard



New Project Default Part Choose a default Xilin: Specify Filter	< part or board fo	r your project. T	his can be chan	ged later.				
Parts Prod	u <u>c</u> t category Ger Eamily Zyr S <u>u</u> b-Family Zyr	ng-7000	Res	👻 Spee <u>d</u> g	kage clg400 rade -1 rade C			* * *
Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceive	GT ers Tra
xc7z010clg400-1 xc7z020clg400-1	400 400	100 125	17600 53200	35200 106400	60 140	80 220	0	0
(III	F	or the Zy	bo				4
				< <u>B</u>	ack <u>N</u> e	ext > E	inish	Cancel

Figure 3. Part Selection to select the XC7Z010-1CLG400C device for the Zybo

1-1-11. Click Next to review the Project Summary page.

1-1-12. Click Finish to create the Vivado project.

Use the Windows Explorer and look at the <2014_2_zynq_labs>\lab1 directory. You will find that the lab1.cache and lab1.srcs directories and the lab1.xpr (Vivado) project file have been created. The lab1.cache directory is a place holder for the Vivado program database. Two directories, constrs_1 and sources_1, are created under the lab1.srcs directory; deep down under them, the copied lab1.xdc (constraint) and lab1.v (source) files respectively are placed.

🔒 lab1	
鷆 lab1.cache	
퉬 lab1.srcs	
퉬 constrs_1	
鷆 imports	
🌗 lab1	
鷆 sources_1	
鷆 imports	
鷆 lab1	

Figure 4. Generated directory structure



1-2. Open the lab1.v source and analyze the content.

1-2-1. In the *Sources* pane, double-click the **lab1.v** entry to open the file in text mode.



Figure 5. Opening the source file

1-2-2. Notice in the Verilog code that the first line defines the timescale directive for the simulator. Lines 2-4 are comment lines describing the module name and the purpose of the module.

Line 7 defines the beginning (marked with keyword **module**) and **endmodule** denotes the end of the module.

Lines 8-9 defines the input and output ports whereas the assignment statements defines the combinatorial functionality of the design.

1-3. Open the lab1.xdc source and analyze the content.

1-3-1. In the *Sources* pane, expand the *Constraints* folder and double-click the **lab1.xdc** entry to open the file in text mode.

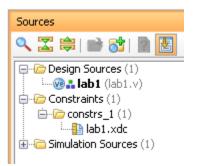


Figure 6. Opening the constraint file

1-3-2. The design takes input from the onboard slide switches and outputs to the onboard LED. The XDC file is different for the ZedBoard and Zybo due to pinout considerations.

1-4. Perform RTL analysis on the source file.

1-4-1. Expand the Open Elaborated Design entry under the RTL Analysis tasks of the Flow Navigator pane and click on **Schematic**.

The model (design) will be elaborated and a logic view of the design is displayed.



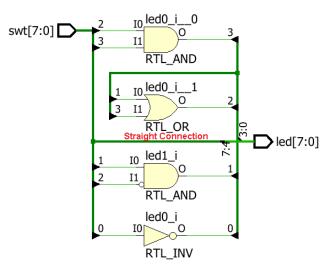


Figure 7. A logic view of the design for the ZedBoard

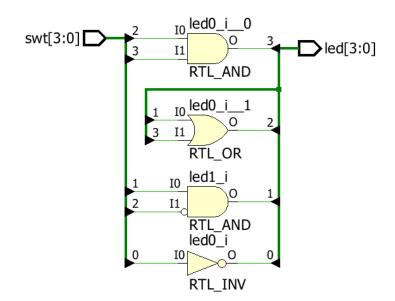


Figure 7. A logic view of the design for the Zybo

Notice that the switch inputs go through gates before being output to LEDs and the rest go straight through to LEDs as modeled in the file (that functionality is for the ZedBoard only).

Simulate the Design using the Vivado Simulator

Step 2

2-1. Add the lab1_tb.v testbench file.

2-1-1. Click Add Sources under the Project Manager tasks of the Flow Navigator pane.





Figure 8. Add Sources

2-1-2. Select the Add or Create Simulation Sources option and click Next.

\lambda Add Sources	
	Add Sources
	This guides you through the process of adding and creating sources for your project
	Add or Create Constraints
	Add or Create Design Sources
	Add or Create Simulation Sources
	Add or Create DSP Sources
	Add Existing Block Design Sources
	Add Existing IP

Figure 9. Selecting Simulation Sources option

- 2-1-3. In the *Add Sources* form, click the **Add Files...** button.
- **2-1-4.** Browse to the **<2014_2_zynq_sources>** folder and the double click on the appropriate **\<board>\lab1** subdirectory. Select *lab1_tb.v,* and click **OK**.
- 2-1-5. Make sure the file is **copied** into the project and click **Finish**.
- **2-1-6.** Select the *Sources* tab and expand the *Simulation Sources* group.

The lab1_tb.v file is added under the *Simulation Sources* group, and **lab1.v** is automatically placed in its hierarchy as a dut1 instance.





Figure 10. Simulation Sources hierarchy

- **2-1-7.** Using the Windows Explorer, verify that the **sim_1** directory is created at the same level as constrs_1 and sources_1 directories under the lab1.srcs directory, and that a copy of lab1_tb.v is placed under **lab1.srcs > sim_1 > imports > lab1**.
- 2-1-8. Double-click on the lab1_tb in the Sources pane to view its contents.



```
1 timescale lns / lps
10
   💹 3 // Module Name: lab1_tb
J.
   5module labl_tb(
Ð
   6
2 🔁
       );
× 8
       reg [7:0] switches;
   9
//
  10
       wire [7:0] leds;
11
       reg [7:0] e_led;
1
  12
  13
       integer i;
-
  14
15
       labl dut(.led(leds),.swt(switches));
  16
Q
  17
       function [7:0] expected_led;
18
         input [7:0] swt;
🐺 <sup>19</sup>
       begin
  20
          expected_led[0] = ~swt[0];
⊉
  21
         expected_led[1] = swt[1] & ~swt[2];
  22
         expected_led[3] = swt[2] & swt[3];
  23
          expected_led[2] = expected_led[1] | expected_led[3];
  24
          expected_led[7:4] = swt[7:4];
  25
       end
  26
       endfunction
  27
  28
       initial
  29
       begin
  30
          for (i=0; i < 255; i=i+2)</pre>
  31
          begin
  32
              #50 switches=i;
  33
              #10 e_led = expected_led(switches);
  34
              if(leds == e_led)
  35
                 $display("LED output matched at", $time);
  36
              else
  37
                  $display("LED output mis-matched at ",$time,": expected: %b, actual: %b", e_led, leds);
  38
           end
  39
       end
  40
  41 endmodule
  42
```

Figure 11. The self-checking testbench for the ZedBoard



l`timescale lns / lps 3 // Module Name: lab1_tb 5 module labl_tb(6 7): 8 9 reg [3:0] switches; 10 wire [3:0] leds; 11 reg [3:0] e_led; 12 13 integer i; 14 15 lab1 dut(.led(leds),.swt(switches)); 16 17 function [3:0] expected_led; 18 input [3:0] swt; 19 begin 20 expected_led[0] = ~swt[0]; 21 expected_led[1] = swt[1] & ~swt[2]; 22 expected_led[3] = swt[2] & swt[3]; 23 expected_led[2] = expected_led[1] | expected_led[3]; 24 end 25 endfunction 26 27 initial 28 begin 29 for (i=0; i < 15; i=i+1)</pre> 30 begin 31 #50 switches=i; 32 #10 e led = expected led(switches); 33 if(leds == e led) 34 \$display("LED output matched at", \$time); 35 else 36 \$display("LED output mis-matched at ",\$time,": expected: %b, actual: %b", e_led, leds); 37 end 38 end 39 40 endmodule 41

Figure 11. The self-checking testbench for the Zybo

The testbench defines the simulation step size and the resolution in line 1. The testbench module definition begins on line 5.

For the ZedBoard, line 15 instantiates the DUT (device/module under test). Lines 17 through 26 define the same module functionality for the expected value computation. Lines 28 through 39 define the stimuli generation, and compare the expected output with what the DUT provides. Line 41 ends the testbench. The \$display task will print the message in the simulator console window when the simulation is run.

For the Zybo, line 15 instantiates the DUT (device/module under test). Lines 17 through 25 define the same module functionality for the expected value computation. Lines 27 through 38 define the stimuli generation, and compare the expected output with what the DUT provides. Line 40 ends the testbench. The \$display task will print the message in the simulator console window when the simulation is run.



2-2. Simulate the design for 200 ns using the Vivado simulator.

2-2-1. Select Simulation Settings under the *Project Manager* tasks of the *Flow Navigator* pane.

A **Project Settings** form will appear showing the **Simulation** properties form.

2-2-2. Select the Simulation tab, and set the Simulation Run Time value to 200ns and click OK.

👃 Proj	ject Settings		×
	9 70	Simulation	
	General	Target simulator:	Vivado Simulator 👻
		Si <u>m</u> ulator language:	Mixed 💌
	Simulation	Simulation set:	🚡 sim_1 💌
		Simulation top module name:	ab1_tb
	Synthesis	☑ Clean up simulation files	
		Generate scripts only	
Im	plementation		
	1010	Compilation Simulation	on Netlist Advanced
	Bitstream	Simulation Run Time	200ns
	===	Design Under Test Instar	nce
	<u> </u>	SAIF Filename	
	ĪP	More Simulation Options	
		Simulation Run Time Specify simulation run time. I input after loading simulation	Default is 1000ns. If not specified, simulation will wait for user n snapshot.
			OK Cancel Apply

Figure 12. Setting simulation run time

2-2-3. Click on **Run Simulation > Run Behavioral Simulation** under the *Project Manager* tasks of the *Flow Navigator* pane.

The testbench and source files will be compiled and the Vivado simulator will be run (assuming no errors). You will see a simulator output similar to the one shown below.



e <u>E</u> dit F <u>l</u> ow <u>T</u> ool	ls <u>W</u> indow La <u>v</u> out	<u>View R</u> un <u>H</u> elp						0	∖ ≁ Search comm	ands
: 🖻 🛛 🖉 🗎	🋍 × I 🕭 🕨 🎙	a 🇠 💥 🔀 🧔 🗉	Default Layout		🗼 🕅 ⊾ Þ(r) 50 ns 🔻 🍤	I 🗖 🖏		write_bi	itstream Comple
Behavioral Simula	tion - Functional - sim_1	l - lab1_tb								
Scopes	C & ×		- 🗆 🖉 🗡	🔡 Untitled 1 🗙						00
् 🛣 🖨 🚺 🔇) 🗄 G 🔒 🎯 🖻		6 6 6	≥ ∎						
Name	Design Unit	Name	Value	Pame Name	Value	0 ns	20	0 ns		400 ns
I ab1_tb	lab1_tb lab1	₽	1110 1101	🔍 🖭 🖏 switches[1110		0001 (0010			
9 glbl	glbl	±	1101	🔍 🖪 📲 leds[3:0]	1101	XXXXX 0001 X	0000 / 0111	X 0110 X	0001 🗙 0000	X 0001 X0000
	-	🗄 😽 i[31:0]	15	🔍 🗉 📲 e_led[3:0	1101	XXXX 0001	0000 011	1 (0110)	0001 000	0001 0
				👢 🖽 📲 i[31:0]	0000000000	000 000	000 000		000 000	. \ 000 \ 0
			1							
<	•			1						
🔒 Scope 💧	Sources			*						
Source File Properties	5 _ D L ^a ×									
🗲 🔶 🗞				4						
🏮 lab1_tb				G.						
Name: /lab1	њ.									
Design unit: lab1_t		=								
Block type: Verilog				31						
	· · · · · · · · ·	-			∢	<				
Tcl Console										
	ut matched at	900 00:08 ; elapsed =	00.00.13	mory (MB) · neel	= 1601 98	90 - ain - 00	100			1
INFO: [V:		im completed. Des				, yum - 0.0				
		im simulation ran								
	sim: Time (s): cp	u = 00:00:08 ; el	apsed = 00:00:	19 . Memory (MB	3): peak =	1601.980 ; gai	n = 0.000.			c
	command here									
	Messages 🛛 🖾 Li	og								
	~									

Figure 13. Simulator output

You will see four main views: (i) *Scopes*, where the testbench hierarchy as well as glbl instances are displayed, (ii) *Objects*, where top-level signals are displayed, (iii) the waveform window, and (iv) *Tcl Console* where the simulation activities are displayed. Notice that since the testbench used is self-checking, the results are displayed as the simulation is run.

Notice that the **lab1.sim** directory is created under the **lab1** directory, along with several lower-level directories.



Figure 14. Directory structure after running behavioral simulation

You will see several buttons next to the waveform window which can be used for the specific purpose as listed in the table below.



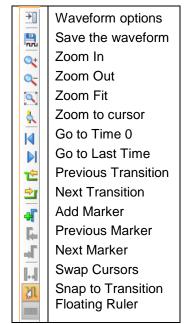


Table 1: Various buttons available to view the waveform

2-2-4. Click on the *Zoom Fit* button (S) to see the entire waveform.

Notice that the output changes when the input changes.

You can also float the simulation waveform window by clicking on the Float button on the upper right hand side of the view. This will allow you to have a wider window to view the simulation waveforms. To reintegrate the floating window back into the GUI, simply click on the Dock Window button.



Figure 15. Float Button



Figure 16. Dock Window Button

2-3. Change display format if desired.

2-3-1. Select i[31:0] in the waveform window, right-click, select *Radix*, and then select *Unsigned Decimal* to view the for-loop index in *integer* form. Similarly, change the radix of switches[7:0] (for the ZedBoard) / switches[3:0] (for the Zybo) to *Hexadecimal*. Leave the leds[7:0] (for the ZedBoard) / leds[3:0] (for the Zybo) and e_led[7:0] (for the ZedBoard) / e_led[3:0] (for the Zybo) radix to *binary* as we want to see each output bit.



2-4. Add more signals to monitor the lower-level signals and continue to run the simulation for 500 ns.

2-4-1. Expand the **lab1_tb** instance, if necessary, in the *Scope* window and select the **dut** instance.

For the ZedBoard, the swt[7:0] and led[7:0] signals will be displayed in the Objects window.

For the Zybo, the swt[3:0] and led[3:0] signals will be displayed in the *Objects* window.

Behavioral Simulati	ion - Functional - sim_1	- lab1_tb			
Scopes		_ 🗆 🖻 ×	Objects		_ 🗆
🔍 🛣 🖨 🔲 🛞	i 🔁 🔂 🚺 🔁	F(x) 🚰	< <br< th=""><th>88</th><th></th></br<>	88	
Name	Design Unit	Block Type	Name	Value	Data Type
l 📕 lab1_tb	lab1_tb	Verilog Module	🕀 😽 switches[7:0]	00000100	Array
i 📒 dut	lab1	Verilog Module	🗄 📲 leds[7:0]	00000001	Array
🛄 🥥 glbl	glbl	Verilog Module	🗄 🐳 😽 e_led[7:0]	00000001	Array
			🗄 🐳 i[31:0]	6	Array
	ources	Tornog Hoddie			

Figure 17. Selecting lower-level signals for the ZedBoard

Behavioral Simulation - Functional - sim_1 - lab1_tb										
Scopes		_ 🗆 🖻 ×	Objects		_ 🗆 L					
🔍 🛣 🖨 🔲 🎯	🗉 G 🔲 🌖 🗉	9 1 1 1 10	88							
Name	Design Unit	Block Type	Name	Value	Data Type					
= 📕 lab1_tb	lab1_tb	Verilog Module	🗈 🐳 switches[3:0]	1110	Array					
i 📒 dut	lab1	Verilog Module	🗄 📲 👪 leds[3:0]	1101	Array					
🛄 🥮 glbl	glbl	Verilog Module	🗄 🐳 e_led[3:0]	1101	Array					
			🗄 🐳 i[31:0]	15	Array					

Figure 17. Selecting lower-level signals for the Zybo

- **2-4-2.** For the ZedBoard, select **swt[7:0]** and **led[7:0]** and drag them into the waveform window to monitor those lower-level signals. For the Zybo, do so for **swt[3:0]** and **led[3:0]**.
- **2-4-3.** On the simulator tool buttons ribbon bar, type 500, click on the drop-down button of the units field and select ns (K k. M 500 ns V 1 1 0) if we want to run for 500 ns (total of 700 ns), and click on the (M) button.

The simulation will run for an additional 500 ns.

2-4-4. Click on the *Zoom Fit* button and observe the output.



Name	¥alue				600 ns	⁵ I		80	ns		1,000 ns	1,200 ns
🖽 📲 switches[3:0]	1110		0111	1000	1001	(1010)	1011	1100	1101		1110	
🖽 – 📲 leds[3:0]	1101	X	0000	0001	0000	0111	0110	1101	1100	1101	1101	
⊞ ‰ e_led[3:0]	1101	0	0000	0001	0000	X 0111	0110	X 110	1100	X	1101	
🖽 - 🖏 i[31:0]	0000000000000	0	000	000	000	000	000	X 000.	000	χ	000000000000000000000000000000000000000	00001111
🖽 📲 swt[3:0]	1110										1110	
🖽 📲 led[3:0]	1101	χ	0000	0001	0000	0111	0110	1101	X 1100 X	1101	1101	

Figure 18. Running simulation for additional 500 ns

Observe the Tcl Console window and see the output is being displayed as the testbench uses the \$display task.

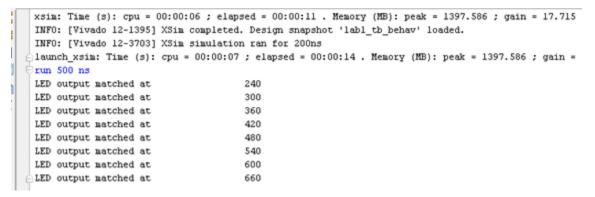


Figure 19. Tcl Console output after running the simulation for additional 500 ns

- **2-4-5.** Close the simulator by selecting **File > Close Simulation**.
- 2-4-6. Click OK and then click No to close it without saving the waveform.

Synthesize the Design

Step 3

3-1. Synthesize the design with the Vivado synthesis tool and analyze the Project Summary output.

3-1-1. Click on Run Synthesis under the Synthesis task of the Flow Navigator pane.

The synthesis process will be run on the lab1.v file (and all its hierarchical files if they exist). When the process is completed a *Synthesis Completed* dialog box with three options will be displayed.

3-1-2. Select the *Open Synthesized Design* option and click **OK** as we want to look at the synthesis output before progressing to the implementation stage.

Click **Yes** to close the elaborated design if the dialog box is displayed.

3-1-3. Select the Project Summary tab and understand the various windows.

If you don't see the Project Summary tab then select Layout > Default Layout, or click the

Project Summary icon



Project Settings				
Project name: lab1 Product family: Zynq- Project part: ZedBo Top module name: lab1	-7000 oard Zyng Evaluation and Development Kit (xc7z020clg4	Device, project nam top module name	e,	
Board				
Board name: em.avn URL: <u>http://w</u> Board overview: 'ZedBoa	net.com:zynq:zed:c www.zedboard.org Ve	o ard name, ersion, overview ^{Id In exploring designs using the Xilinx : Zynq-7000 A oof-of-concept development.¹}	ll Prograf	grammable SoC. The board contains all the necessary interfaces and supporting functions to enable a wide range of applications. The expan
Synthesis			*	* Implementation
Skatus: V Complete Messages: No errors or Part: xc7z020clg48 Strategy: <u>Vivado Synth</u>	warnings completed 84-1			Status: Not started Implementation Messages: No errors or warnings not started Part: xc72020clq494-1 Stategy: Strategy: Wixado Implementation Defaults Incremental Compile: Kone Summary Route Status
DRC Violations			*	* Timing
	DRC information is not available because	iit hasn't been run		Timing information is not available because it hasn't been run
Utilization	Resource utilization	1	*	☆ Power
	graphical overview 25 50 Estimated Utilization elect between Graph ad Table display	75 100 on (%)		Power information is not available because it hasn't been run
	Post-Implementation			-

Figure 20. Project Summary view

Click on the various links to see what information they provide and which allows you to change the synthesis settings.

3-1-4. Click on the Table tab in the Project Summary tab.

Notice that there are an estimated three LUTs and 16 IOs (8 input and 8 output) that are used for the ZedBoard. There are only 8 IOs used for the Zybo (4 input and 4 output). The number of total available pins differ based on the target device used with the XC7Z010 having less pins than the XC7Z020.

Resource	Estimation	Available	Utilization %
LUT	3	53200	0.01
I/O	16	200	8.00

Figure 21. Resource utilization estimation	summary for the ZedBoard
--	--------------------------

Resource	Estimation	Available	Utilization %
LUT	3	17600	0.02
I/O	8	100	8.00

Figure 21. Resource utilization estimation summary for the Zybo

3-1-5. In The *Flow Navigator*, under *Synthesis* (expand *Synthesized Design* if necessary), click on **Schematic** to view the synthesized design in a schematic view.



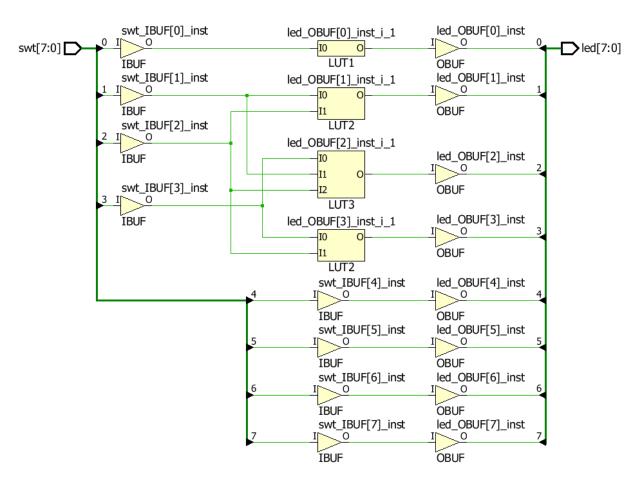


Figure 22. Synthesized design's schematic view for the ZedBoard

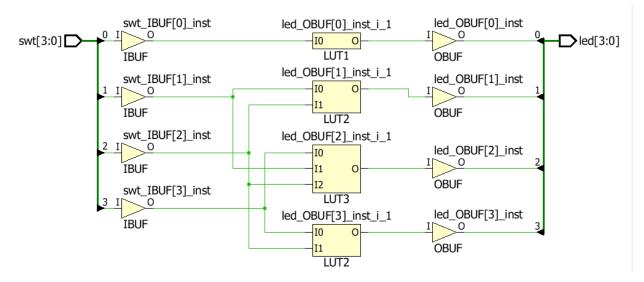


Figure 22. Synthesized design's schematic view for the Zybo

Notice that IBUFs and OBUFs are automatically instantiated (added) to the design as the input and output are buffered. The logical gates are implemented in LUTs (1 input is listed as LUT1, 2 input is listed as LUT2, and 3 input is listed as LUT3). The four gates in RTL analysis output are mapped onto four LUTs in the synthesized output. Also notice that the design is effectively split in



two between the two boards, with one half being available both the ZedBoard and Zybo (LUTs) while the other half toggled by the switches (straight through connection) being only available to the ZedBoard. This is due to the way the two designs are coded.

Using Windows Explorer, verify that **lab1.runs** directory is created under **lab1**. Under the **runs** directory, **synth_1** directory is created which holds several files related to synthesis.

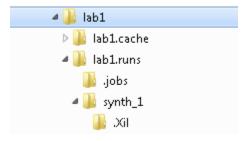


Figure 23. Directory structure after synthesizing the design

Implement the Design

Step 4

- 4-1. Implement the design with the Vivado Implementation Defaults (Vivado Implementation 2014) settings and analyze the Project Summary output.
- 4-1-1. Click on Run Implementation under the Implementation tasks of the Flow Navigator pane.

The implementation process will be run on the synthesized design. When the process is completed an *Implementation Completed* dialog box with three options will be displayed.

- **4-1-2.** Select **Open implemented design** and click **OK** as we want to look at the implemented design in a Device view tab.
- 4-1-3. Click Yes, if prompted, to close the synthesized design.

The implemented design will be opened.

- **4-1-4.** In the *Netlist* pane, select one of the nets by expanding the Nets branch (e.g. led_OBUF[0]) and notice that the net displayed in the Device view tab (you may have to zoom fit in to see it).
- **4-1-5.** If it is not selected, click the *Routing Resources* icon **b** to show the routing resources.

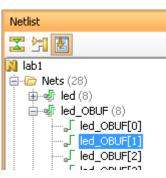


Figure 24. Selecting a net



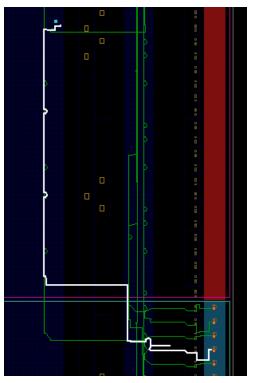


Figure 25. Viewing the implemented ZedBoard design

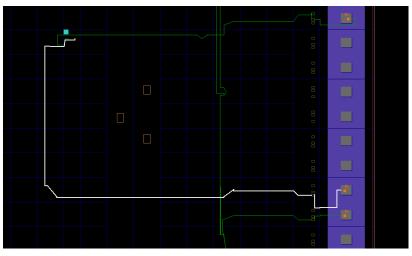


Figure 25. Viewing the implemented Zybo design

4-1-6. Close the implemented design view and select the **Project Summary** tab (you may have to change to the Default Layout view) and observe the results.

Select the Post-Implementation tab in the **Utilization** pane.

Notice that the actual resource utilization is 3 LUTs and 16 IOs for the ZedBoard and 3 LUTs and 8 IOs for the Zybo. Also, it indicates that no timing constraints were defined for this design (since the design is combinatorial).



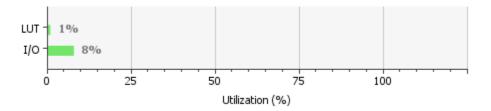


Figure 26. Post implementation resources chart for both boards

Using the Windows Explorer, verify that **impl_1** directory is created at the same level as **synth_1** under the **lab1.runs** directory. The **impl_1** directory contains several files including the implementation report files.

4-1-7. In Vivado, select the **Reports** tab in the bottom panel (if not visible, click *Window* in the menu bar and select **Reports**), and double-click on the *Utilization Report* entry under the *Place Design* section. The report will be displayed in the auxiliary view pane showing resource utilization. Note that since the design is combinatorial no registers are used.

Rep	orts		
9	Name	Modified	Size
\mathbf{Z}	⊖~Synth Design (synth_design)		
	- 📄 Vivado Synthesis Report	6/20/14 1:25 PM	15.8 KB
-	🛄 🗎 Utilization Report	6/20/14 1:25 PM	5.7 KB
	Place Design (place_design)		
	📄 Vivado Implementation Log	6/20/14 1:34 PM	16.6 KB
	📄 Pre-Placement Incremental Reus		
	📄 IO Report	6/20/14 1:33 PM	111.5 KB
	- 📄 Clock Utilization Report	6/20/14 1:33 PM	5.0 KB
	📄 Utilization Report	6/20/14 1:33 PM	7.7 KB
	📄 Control Sets Report	6/20/14 1:33 PM	2.5 KB
	🛄 Incremental Reuse Report		
	🛱 Route Design (route_design)		
	📄 Vivado Implementation Log	6/20/14 1:34 PM	16.6 KB
	📄 WebTalk Report		
	📄 DRC Report	6/20/14 1:34 PM	1.8 KB
	📄 Power Report	6/20/14 1:34 PM	7.3 KB
	📄 Route Status Report	6/20/14 1:34 PM	0.6 KB
	- 📄 Timing Summary Report	6/20/14 1:34 PM	7.0 KB
	🛄 Incremental Reuse Report		
	Post-Route Phys Opt Design (post_route_	phys_opt_design)	
	🛄 🖿 Post-Route Physical Optimizatio		
	🗄 • Write Bitstream (write_bitstream)		
	🖿 Vivado Implementation Log		
	🛄 🖿 WebTalk Report		

Figure 27. Available reports to view



Perform Timing Simulation

5-1. Run a timing simulation.

5-1-1. Select **Run Simulation > Run Post-Implementation Timing Simulation** process under the *Simulation* tasks of the *Flow Navigator* pane.

The Vivado simulator will be launched using the implemented design and **lab1_tb** as the top-level module.

Using the Windows Explorer, verify that **timing** directory is created under the **lab1.sim > sim_1 > impl** directory. The **timing** directory contains generated files to run the timing simulation.

- 5-1-2. Click on the **Zoom Fit** button to see the waveform window from 0 to 200 ns.
- **5-1-3.** Click on the 50 ns mark (where the switch input is set to 0000000b for the ZedBoard or 0000b for the Zybo) and a marker is automatically placed at the location.
- **5-1-4.** You can also add a marker by clicking on the Add Marker button (⁴). Click on the Add Marker button and left-click at around 60 ns where **e_led** changes.

🔞 lab1.v 🗙 🔛 Untitled 2* 🗙								
				60.000 ns				
Pame Name	¥alue	0 ns	<u>50</u> . 50	000 ns ns	100 ns	5	150 ns	
🔍 🖽 🖏 switches[7:0]	00000000	xxxxxxxxx		00000000		0000001	• <u> </u>	
🔍 🖽 📲 leds[7:0]	00000001	>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>	X	00000001		000001	.11	
📉 🖽 📷 e_led[7:0]	00000001	XXXXXXXXXX		0000000	1	0000	0111	
▲ III:0]	000000000000000000000000000000000000000	(00000000000000000000000000000000000000))	(00000000000	0000)	00000000	000000)	

Figure 28. Timing simulation output for the ZedBoard

😸 Untitled 3* 🗙								
<u>₹</u> 0				60.000 ns				
Pame Name	Value	0 ns		000 ns ns	100 n	s	150 ns	
🔍 🖽 🖏 switches[3:0]	0000	×××××	C	0000	٥	0001		
🔍 🖪 📲 leds[3:0]	0001	X000X	ÞC	0001		0000		
📉 🖽 🖬 e_led[3:0]	0001	X000X		0001		00	00	\mathbf{x}
▲ III-III [31:0]	000000000000000000000000000000000000000	(00000000000000000000000000000000000000	D	(0000000000	<u>0000</u>	0000000	000000	<u> </u>

Figure 28. Timing simulation output for the Zybo

Notice that we monitored the expected led output at 10 ns after the input is changed (see the testbench) whereas the actual delay is about 5.000 ns where leds[7:0] or leds[3:0] changes.



5-1-5. Close the simulator by selecting **File > Close Simulation** without saving any changes.

Generate the Bitstream and Verify Functionality

Step 6

- 6-1. Connect the board and power it ON. Generate the bitstream, open a Hardware Manager, and program the FPGA.
- **6-1-1.** Make sure that the Micro-USB cable is connected to the JTAG PROG connector (next to the power supply connector). Connect the power jack for the ZedBoard (none needed for the Zybo).



Figure 29. ZedBoard connections





Figure 29. Zybo connection

- 6-1-2. Power ON the board.
- 6-1-3. Click on the Generate Bitstream entry under the *Program and Debug* tasks of the *Flow Navigator* pane.

The bitstream generation process will be run on the implemented design. When the process is completed a *Bitstream Generation Completed* dialog box with two (or three) options will be displayed.

Bitstream Generation Completed
Bitstream Generation successfully completed.
Next
O View Reports
Open Hardware Manager
Don't show this dialog again
OK Cancel

Figure 30. Bitstream generation



This process will have generated a lab1.bit file under impl_1 directory in the lab1.runs directory.

If the dialog box is not displayed, to perform the next step, simply select the **Open Hardware Manager** option under *Program and Debug* in the *Flow Navigator* pane.

6-1-4. Select the Open Hardware Manager option and click OK.

The Hardware Manager window will open indicating "unconnected" status.

6-1-5. Click on the Open New Hardware Target link.

You can also click on the Open Recent Hardware Target link if the board was already targeted before.



Figure 31. Opening new hardware target

- 6-1-6. Click Next to see the Hardware Server Settings form.
- 6-1-7. Click **Next** with the Hardware Target selected.

The JTAG cable which should be detected and identified as a hardware target. It will also show the hardware devices detected in the chain.

	-			
Select Hard		-		
			f available targets, then set the appropriate JTAG clock ted devices, decrease the frequency or select a different	
ardware Targ	ets			
Туре И	Port Na	me	JTAG Clock Frequency	
xilinx_tcf	Digi	lent/21024847036	64 15000000	
ardware Devi	:es (for unk	nown devices, sp	ecify the Instruction Register (IR) length)	
	es (for unk	nown devices, sp IR Length	ecify the Instruction Register (IR) length)	
Vame	ID Code	IR Length	ecify the Instruction Register (IR) length)	
Name) arm_dap_0	ID Code 4BA00477	IR Length 4	ecify the Instruction Register (IR) length)	
lame ≽arm_dap_0	ID Code 4BA00477	IR Length 4	ecify the Instruction Register (IR) length)	
lame ≽arm_dap_0	ID Code 4BA00477	IR Length 4	ecify the Instruction Register (IR) length)	
lame ≽arm_dap_0	ID Code 4BA00477	IR Length 4	ecify the Instruction Register (IR) length)	
lame ≽arm_dap_0	ID Code 4BA00477	IR Length 4	ecify the Instruction Register (IR) length)	
lame ≽arm_dap_0	ID Code 4BA00477	IR Length 4	ecify the Instruction Register (IR) length)	
Vame ▶ arm_dap_0 ▶ xc7z020_1	ID Code 4BA00477	IR Length 4 6	ecify the Instruction Register (IR) length)	
Jame arm_dap_0 xc7z020_1 SE server:	ID Code 4BA00477 03727093	IR Length 4 6	ecify the Instruction Register (IR) length)	
Jame arm_dap_0 xc7z020_1 SE server:	ID Code 4BA00477 03727093	IR Length 4 6	ecify the Instruction Register (IR) length)	
ardware Devia Name arm_dap_0 xc7z020_1 SE server: ardware server	ID Code 4BA00477 03727093	IR Length 4 6	ecify the Instruction Register (IR) length)	

Figure 32. New hardware target detection for the ZedBoard



🚴 Open New	Hardware T	arget	a test property	×
	- rdware targ	et from the list o	f available targets, then set the appropriate JTAG clock (TCK) ted devices, decrease the frequency or select a different target.	
Hardware Targe	ets			
Type F	Port Nam	e	JTAG Clock Frequency	
🖉 xilinx_tcf	Digile	nt/2102795402	43A 15000000	
Hardware Devic	es (for unkn	own devices isn	ecify the Instruction Register (IR) length)	
Name	ID Code	IR Length	only the analysis (any origin)	
🔷 arm_dap_0	4BA00477			
xc7z010_1	13722093	6		
VCSE server:	localhost:	60001		
Hardware serve	er: localhost:	3121		
			< Back Next > Einish	Iancel

Figure 32. New hardware target detection for the Zybo

6-1-8. Click Next and then Finish.

The Hardware Manager status changes from Unconnected to the server name and the device is highlighted. Also notice that the Status indicates that it is not programmed.

Name	Status
🖃 🚪 localhost (1)	Connected
📄 🗑 🖉 xilinx_tcf/Digilent/210248470364 (2)	Open
🛶 🔷 arm_dap_0 (0)	Not programmed
😑 🧇 xc7z020_1 (1)	Not programmed
🦉 XADC (System Monitor)	

Figure 33. Opened Hardware Manager for the ZedBoard

Name	Status
💷 🚪 localhost (1)	Connected
🖻 📓 🤌 xilinx_tcf/Digilent/21027954024	Open
🔷 🍥 arm_dap_0 (0)	Not programmed
🗄 🔷 xc7z010_1 (1)	Not programmed
🛄 🤷 🗱 XADC (System Monitor)	

Figure 33. Opened Hardware Manager for the Zybo



6-1-9. Select the XC7Z010 or XC7Z020 device and verify that the lab1.bit is selected as the programming file in the General tab.

Programming file:	C:/xup/fpga_flow/2014_2_zynq_labs/lab1/lab1.runs/impl_1/lab1.bit
Probes file:	C:/xup/fpga_flow/2014_2_zynq_labs/lab1/lab1.runs/impl_1/debug_nets.ltx 📀 🗔

Figure 34. Programming file to be configured into the target device

6-1-10. Right-click on the device and select *Program Device…* to program the target FPGA device.

Hardware Manager - localhost/xilinx_t	cf/Digile	ent/210248470364			
(1) There are no debug cores. Program	<u>device</u>	Refresh device			
Hardware			- 0 0	×	Debug
🔍 🛣 🖨 🛃 🕨 🕨 🔳					۹ 🗄
Name		Status			
🖃 🚪 localhost (1)		Connected			
📄 📴 🤌 xilinx_tcf/Digilent/2102484703	364 (2)	Open			
		Not programmed			
🖃 🔷 xc7z020_1 (1)		Not programmed			
🛄 🤷 🙀 XADC (System Monitor)	<u>6</u>	Hardware Device P	roperties	C	Ctrl+E
	۰	Program Device			
	ø	Refresh Device			
	¢¢	Add Configuration I	Memory Device		
	I	Boot Device			
	1	Export to Spreadsh	eet		

Figure 35. Selecting to program the FPGA

6-1-11. Click Program to program the FPGA.

The DONE light (Blue LED on the ZedBoard and Green LED on the Zybo) will light when the device is programmed. You may see some other LEDs lit depending on switch positions.

- **6-1-12.** Verify the functionality by flipping switches and observing the output on LEDs (Refer to the earlier logic diagram/elaboration schematics).
- 6-1-13. When satisfied, power OFF the board.
- 6-1-14. Close the Hardware Manager by selecting File > Close Hardware Manager.
- 6-1-15. Click OK to close the session.
- 6-1-16. Close the Vivado program by selecting File > Exit and click OK.



Conclusion

The Vivado software tool can be used to perform a complete design flow. The project was created using the supplied source files (HDL model and user constraint file). A behavioral simulation using the provided testbench was done to verify the model functionality. The model was then synthesized, implemented, and a bitstream was generated. The timing simulation was run on the implemented design using the same testbench. The functionality was verified in hardware using the generated bitstream.

