

Embedded Linux[®] Hands-on Tutorial for the ZYBOTM

Revised December 5, 2014

Overview

The purpose of this document is to provide step-by-step instructions for customizing your hardware, compiling the Linux Kernel, and writing driver and user applications. This documentation intends to integrate knowledge and skills in FPGA logic circuit design, standalone software programming, Linux operating system and user application development, and apply them to the ZYBO. We will start from the *ZYBO Base System Design* (available on the ZYBO product page of the Digilent website). The system architecture for the *ZYBO Base System Design* is shown in Fig. 1.

In the *ZYBO Base System Design*, we connect UART1 to USB-UART, SD0 to the SD Card Slot, USB0 to the USB-OTG port, Enet0 to the Giga-bit Ethernet Port, and Quad SPI to the on-board QSPI Flash. These cores are hard IPs inside the Processing System (PS) and connect to on-board peripherals via Multiplexed I/O (MIO) pins. The use of PS GPIO is connected to Btn 4 and 5. In the Programmable Logic (PL), we have an HDMI TX Controller, VDMA, and GPIO IP cores to talk to the ADV7511 HDMI transmitter chip and I2S and GPIO IP cores for ADAU1761 audio codec. More details of the hardware design can be found in the documentation inside the *ZYBO Base System Design* package.



Figure 1. Reference Basic Hardware System Architecture for ZYBO.



In this tutorial, we are going to detach the LEDs from the AXI GPIO core and implement our own myLed core for it in PL, as shown in Fig. 2. We will then add our own LED controller into the device tree, write a driver for it, and develop user applications to control the status of the LEDs.



Figure 2. Hardware System Architecture of the system we are going to implement in this Tutorial.

Before going through this tutorial, we recommend that you read *Getting Started with Embedded Linux - ZYBO*. You can follow this tutorial with the *Embedded Linux Development Guide* (available on the Digilent website Embedded Linux Page). The guide will provide you with the knowledge you may need in each step of the development.

In this tutorial, we are going to use Xilinx[®] Vivado[™] 2014.1 WebPACK[™] in a Linux environment. All of the screenshots and codes are done using Vivado Design Suite 2014.1 in CentOS 6 x86_64.

That's it for the background information on this tutorial, now it's time to get our hands dirty with some real design!



1 Hardware Customization

1.1 **Prerequisites**

- Vivado 2014.1 WebPACK: available at the Xilinx website <u>Download Page</u>.
- > ZYBO Base System Design: available at the Digilent website on the <u>ZYBO Page</u>.

1.2 Instructions

1. Download the *ZYBO Base System Design* from the Digilent website and unzip it into our working directory, as in Fig. 3 (our working directory is named **tutorial** throughout this document). For more information on the hardware design, please refer to *Project Guide* under doc folder.



Figure 3. Unzip the ZYBO_Base_System.

2. Source Vivado 2014.1 settings and open the design with Vivado Design Suite. You will see the Vivado window pop up as shown in Fig. 4.

Note: There are four settings files available in the Vivado toolset: settings64.sh for use on 64-bit machines with bash; settings32.sh for use on 32-bit machines with bash; settings32.csh for use on 32-bit machines with C Shell; and settings64.csh for use on 64-bit machines with C Shell.

```
[kfranz@localhost Tutorial]$ source /opt/Xilinx/Vivado/2014.1/settings64.sh
[kfranz@localhost Tutorial]$ vivado zybo_base_system/source/vivado/hw/zybo_bsd/z
ybo_bsd.xpr
****** Vivado v2014.1 (64-bit)
 **** SW Build 881834 on Fri Apr 4 14:00:25 MDT 2014
 ***** IP Build 877625 on Fri Mar 28 16:29:15 MDT 2014
 ** Copyright 1986-2014 Xilinx, Inc. All Rights Reserved.
start_gui
```

Figure 4. Open the Project.



Activities Activities	Wed10:46 AM ቀባ ନ 🔒 🗣 ዘ	aitlyn Franz
	zybo_bsd ~ [/home/kfranz/Desktop/Tutorial/zybo_base_system/source/vivado/hw/zybo_bsd/zybo_bsd.xpr] - Vivado 2014.1	×
Elle Edit Flow Tools Window Layout View Help	Q- Search comma	inds
🟄 😂 🖩 ថ 🕫 🐘 🦍 X 🤯 🐎 🕨 🐮 🚳 🐝 🗵 🎯 😑 De	suit Layout 🔷 🖉 🚸 🗽 😡	Ready
Flow Navigator «	Block Design - system	×
< Ⅲ 申	The design has 9 blocks that should be upgraded. Show IP Status Upgrade Later	
Nov Hasgator	The design that about the though be opgraded. Those If Status Lagradie Later The design that about the though be opgraded. Those If Status Lagradie Later Togetine Toget	
 (%) implementation Stittings) Program by/enerrisation >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>	# Statue # Statue ************************************	

Figure 5. Vivado platform studio GUI.

We are going to detach LEDs from the GPIO core in the PS first. So we need to click on the IP integrator and open the **Block Diagram** as shown in Fig. 5. Then we need to delete the current LED IP as shown in Fig. 6. We will handle the modification of external pin location configuration (**xdc** file) in later steps.

Note: In Fig. 6 there is a yellow bar indicating the need for an upgrade. To upgrade, hit **show IP status**, make sure all are selected and hit **Upgrade Selected**.

Block Design - system * X					
The design has 9 blocks that should be upgraded. Show IP Status Upgrade Later					
Design Hierarchy _ C X	🚰 Diagram 🗙 💐 Address Editor 🗙	oe×			
오 🛣 🛃 😓	🕺 🎄 system 🕨				
Swise S	A system • mel, axi, j Mel, axi, j mel, axi, j Mel, axi, j	Pre-Production) Bits Pro-Production Bits Pro-Production Bits Pro-Production Bits Pro-Production Bits Pro-Production Bits Pro-Production Bits Bits Pro-Production Bits Bits Pro-Production Bits Bits Bits			
General Properties		Create Comment Create Port Ctrl+K Create Interface Port Ctrl+L			
IP Status		Save as PDF File X			
🔍 🗹 3 Minor Changes 🛛 6 Revision Changes 🗌 🗄	Other Changes Show All				

Figure 6. Delete existing LED IP.

4. (Vivado 2014.1 only) Before we can start implementing our myLed IP Core, we need to name the vendor that will automatically be applied in the IP packager. In Vivado 2014.1, this is not automatically done for the user. To do this, first go to the **Project Settings** under **Project Manager** on the left side of the window (Fig. 7) and the project settings window will pop up. In the Project Settings window, select **IP** (Fig. 8). Notice that the vendor is chosen as "(none)", this will cause a Vivado internal exception. You can name the Vendor whatever you like (Fig. 9).

Activities Activities
File Edit Flow Tools Window Lavout View Help
🧶 🖄 💾 🙋 💷 🖿 🗙 😺 🔈 🕨 🚷 💥 ∑ 🚳 📟 Defa
Elmy Nevinetor
 Project Manager
Project Settings
Add Sources Project Settings
IP Catalog Configure synthesis, simulation, implement
IP Integrator
ở Create Block Design
🔂 Open Block Design
🎭 Generate Block Design
 Simulation
n Simulation Settings
🔍 Run Simulation
 RTL Analysis
Open Elaborated Design
4 Suppliers
Synthesis Settings
Bun Synthesis
Open Synthesized Design
Implementation Cattings
Bus Implementation
Profilemented Design
 Program and Debug
10 Bitstream Settings
Generate Bitstream
Image: Part of the second s

Figure 7. Project settings.





Project Settings		Project Settings	
General General Simulation Synthesis Implementation Bitstream Implementation Bitstream	IP Repository Manager Packager Default Values Image: Packager Wizard. Yendor: (none) Image: Packager Wizard. Ugrany: Ugrany: Create IP Location: /p_repo Automatic Behavior Image: Create archive of IP Image: Add IP to the IP Catalog of the current project Close IP Packager window Edit IP in IP Packager Image: Delete project after packaging Filtered Extensions Image: Create a list of file extensions that will be automatically filtered when adding a directory to a File Group. File Extensions to Filter on Add Directory Image: Packager Image: Pack	Image: Simulation Image: Simulation Image: Synthesis Image: Simulation <t< td=""><td>P C C C C C C C C C C C C C</td></t<>	P C C C C C C C C C C C C C

Figure 8. Unnamed vendor.

Figure 9. Named vendor.

5. Now we can start implementing our myLed IP Core. Click **Tools** -> **Create and Package IP...** from the menu (as shown in Fig. 10). The Create and Package New IP window will pop up (as shown in Fig. 11), Click **Next**. In the next window, name the new IP and click next again (Fig. 12).



Figure 10. Create and Package IP.



Create And Package New IP		Create And Package New IP	×
	Create and Package IP This wizard can be used to accomplish two tasks:	Peripheral Details Specify name, version and description for the new peripheral	1
	This wizard will guide you through the process of creating a new Vivado IP using source files and information from your current project or specified directory. Create a new AXIA Peripheral This wizard will guide you through the process of creating a new AXIA peripheral which includes HDL, driver, software test application, IPI BFM simulation and debug demonstration design.	Name: myLed Version: 1.0 Display Name: myLed_v1.0	8
		Description: My Led IP IP Location: //home/kfranz/Tutorial2/zybo_base_system/source/vivado/hw/ip_repo	0 -
VIVADO. ^A	Click Next to continue		
	< Back Next > Einish Cancel	< <u>Back</u> <u>Next ></u> <u>Finish</u>	Cancel

Figure 11. IP Options.

Figure 12. Peripheral Details.

6. The next window will be the Add Interfaces Window. This will create the AX14 Interface for the myLed peripheral (Fig. 13). Make sure the interface type is Lite, the mode is Slave, the data width is 32 bits and the number of registers is 4. Change the Name to S_AXI rather than S00_AXI. We only need 1 register but the minimum we can select is 4. Click next to proceed.

Create And Package New IP ×					
Add Interfaces Add AXI4 interfaces supported by your peripheral					
🗆 Enable Interrupt Support 🛛 🕂 🗙	Name S_AXI (
Interfaces Image: S_AXI myLed_v1.0	Interface Type Interface Mode Data Width (Bits) 32 Memory Size (Bytes) 64 Number of Registers 4 [4512]				
	< Back Next > Finish Cancel				

Figure 13. Add Interface



7. The next window will prompt the finishing steps to create the IP (Fig. 14). Change the Radio button to select **Edit IP** and hit finish. We need to add user logic to the IP so that our slave is connected to the LED output.

Create And Package New IP			
	Create Peripheral		
	Peripheral Generation Summary 1. IP (Digilent:user:myLED:1.0) with <u>1 interface(s)</u>		
	2. Driver(v1_00_a) and testapp <u>more info</u> 3. AXI4 BFM Simulation demonstration design <u>more info</u>		
	4. AXI4 Debug Hardware Simulation demonstration design more info		
	Peripheral created will be available in the catalog : /home/kfranz/Desktop/Tutorial/zybo_base_system/source/vivado/hw/ip_repo		
	Next Steps:		
	○ Add IP to the repository		
	○ Verify peripheral IP using AXI4 BFM Simulation interface		
	O Verify peripheral IP using JTAG interface		
VIVADO.	Click Finish to continue		
	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish Cancel		

Figure 14. Edit IP.

8. After selecting finish, the Create and Package IP window will disappear and the next window you will see is the edit_myLed window (Fig. 15). This is where we will add our user logic.





Activities	Wed 11:35 AM	🐠 😌 🔐 Kaitlyn Franz			
	edit_myLED_v1_0 - [/home/kfranz/Desktop/Tutorial/zybo_base_system/source/vivado/hw/ip_repo/edit_myLED_v1_0	xpr] - Vivado 2014.1			
File Edit Flow Tools Window La	ayout View Help	Q _v Search commands			
2 🔁 10 67 🗈 🗈 🗙 🔈 🗎	🙆 🚳 💥 ∑ 👰 🖾 Default Lavout 🚽 🖉 🛸 🕲	Ready			
Eleve Mexico ter					
o w w	v Project manager - eutringtzouri.v				
× mar 101	Sources Zroject summary x = Package PrimyLeD x = myLeD_11_0_50_AXV x	1.0.500 M/I			
 Project Manager 	A set of the set of	10,000,000			
🚱 Project Settings	Design sources (2) de-an myLED v1 0.(m/LED v1 0.v)(1) de-an myLED v1 0.(m/LED v1 0.v)(1)				
😚 Add Sources	- motule avLED v1 0 S00 AXI inst - myLED v1 0 S00 AXI in				
🐓 IP Catalog	Generation (1) La component xml				
👶 Package IP	Constraints T T T				
 IP Integrator 	Simulation Sources (1)				
🗱 Create Block Design	n 10				
Dpen Block Design	// 11 // Width of S AXI data bus				
Senerate Block Design	12 parameter integer C. San Jaka Multin = 32, 1 / Width of S. AXT address bus				
	14 parameter integer C_S_AXI_ADDR_WIDTH = 4				
 Simulation 					
Simulation Settings	de 17 // Users to add ports here				
Run Simulation	Hierarchy Libraries Compile Order				
 RTL Analysis 	A Sources Templates				
👂 📑 Open Elaborated Design	Source Elle Properties				
4 Sontheric	A a log light and light an				
Synchesis Suthesis Settings	myter vice Statt ARSETN ARSETN				
Dun Sunthesis	/ Write address (issued by mater, accept by Slave)				
Dinen Synthesized Design	Location: /home/kfranz/Desktop/Tutorial/zybo_base_syst 28 // Write channel Protection type. This signal indicates the				
	Type: Verilog 29 // privilege and security level of the transaction, and whether 30 // the transaction is a data access 30 // the transaction is a data access.				
 Implementation 	Library: wildefaultib - 31 input vire [2:0] S_AXI_AWPROT,				
Implementation Settings	Size: 13.6 KB 32 // Write address valid. This signal indicates that the master signaling // Write address and control information				
Run Implementation	Modified: Today at 11:33:54 AM 34 input vire S_AXI_AWVALID.				
Open Implemented Design	Read-only: No 35 // Write address ready. This signal indicates that the slave is ready // Write address ready. This signal indicates that the slave is ready // Write address ready. This signal indicates that the slave is ready // Write address ready.				
 Program and Debug 	Encrypted: No 37 eutput wire S_AXI_AWREADY,				
🚳 Bitstream Settings	Global include include include include include include a close of the				
😤 Generate Bitstream	General Properties 40 // Write strokes. This signal indicates which byte lanes hold				
Open Hardware Manager	Veneral risperses	•			
	Design Runs	_ D & ×			
	Name Part Constraints Strategy Host Stratus Progr Strategy Host Status Progr Not started	ess Start Elapsed WNS TNS WHS THS TPWS Failed R			
	📥 🛏 impl 1 xc7z010clg400-1 constra 1 Vivado implementation Defaults (Vivado Implementation 2014) Not started	0%			
		•			
	📾 Ici console 🛩 Messages 🔍 Log 🧝 Reports 🕐 Design Runs				
		4:0 Insert Verilog			

Figure 15. Edit_myLed.

9. In the Project Manager, click the circle next to myLed_v1_0 and highlight myLed_v1_0_S_AXI (Fig. 16). This contains the user logic inside of the myLed IP. We need to add two lines of code to complete the user logic for this module. First, we need to create a user port called led (Fig. 17). Next, we need to connect the internal slave to this user port. We will connect slv_reg0[3:0] as we have four LEDs (Fig. 18).



Figure 16. Select user logic file.



1 2 3	timescale	1 ps / 1 ps
4	module	myLed_v1_0_S00_AXI #
6	- 1/	Users to add parameters here
8 9	11	User parameters ends Do not modify the parameters beyond this line
10 11 12 13	// pa	Width of S_AXI data bus rameter integer C_S_AXI_DATA_WIDTH = 32, Width of S_AXI address bus
14 15 16) (<pre>rameter integer C_S_AXI_ADDR_WIDTH = 4</pre>
18	out	tput wire [3:0]led,
19 20 21	11	User ports ends Do not modify the ports beyond this line

395	end
396	end
397	end
398	
399	// Add user logic here
400	assign led = slv_reg0[3:0];🧲
401	// User logic ends
402	
403	endmodule

Figure 18. Add user logic.

Figure 17. Add user port.

Next, we need to connect the user logic to myLed. In the project manager select the file myLed_v_0. To complete the IP, there are two lines of code we need to add to this file. Under the comment that says "Users to add ports here," add a port for the LEDs (Fig. 19). Connect the led output from the previous file containing the user logic to myLed (Fig. 20).

21	timesc	ale 1 ps / 1 ps	46//	' Instantiation of Axi Bus Interface S00_AXI
3			47	myLed_v1_0_S00_AXI # (
4	mod	ule myLed_v1_0 #	48	.C_S_AXI_DATA_WIDTH(C_S00_AXI_DATA_WIDTH),
5	(II llases to add exercise base	49	.C_S_AXI_ADDR_WIDTH(C_S00_AXI_ADDR_WIDTH)
0		// Users to add parameters nere	50) myLed_v1_0_SO0_AXI_inst (
é.		// liser narameters ends	51	.led(led), 🗲
9		// Do not modify the parameters beyond this line	52	.S_AXI_ACLK(s00_axi_aclk),
10		,,,,,	53	.S_AXI_ARESETN(s00_axi_aresetn),
11			54	.S_AXI_AWADDR(sOO_axi_awaddr),
12		// Parameters of Axi Slave Bus Interface S00_AXI	55	.S_AXI_AWPROT(s00_axi_awprot),
13		<pre>parameter integer C_SOO_AXI_DATA_WIDTH = 32,</pre>	56	.S_AXI_AWVALID(s00_axi_awvalid),
14	,	parameter integer C_SOO_AXI_ADDR_WIDTH = 4	57	.S_AXI_AWREADY(s00_axi_awready),
15)		58	.S_AXI_WDATA(s00_axi_wdata),
17	(// lisers to add ports here	59	.S_AXI_WSTRB(s00_axi_wstrb),
18		output [3:0] led.	60	.S_AXI_WVALID(s00_axi_wvalid),
19		// liser ports ends		

Figure 19. Add External.

Figure 20. Connect myLed to User Logic.

11. Now that our IP is created and the user logic is defined, we need to package our IP. Under Project Manager on the left side of the window, select Package IP. A new tab will open that is called Package IP. On the left side of this tap there are a series of labels. We need to complete those that do not have green check marks.

First, select **IP customization Parameters**. At the top of that window select the option to **merge changes from IP Customization Parameters Wizard**, as in Fig. 21.



E Project Summary × 🔮 Package I	P-myLed × @myLed_v1_0_SO0_AXI.v × @myLed_v1_0.v ×	
IP Packaging Steps 《	IP Customization Parameters	wizard more info
✓ IP Identification	Merge changes from IP Customization Parameters Wizard	
✓ IP Compatibility	Name Merge changes from IP Customization Parameters Wizard Changes detected in Market principal Parameters Wizard	Value Format Value Va
✓ IP File Groups	C S00_AXI_DAT Click here to merge project changes.	ong
➢ IP Customization Parameters	- © C SOO AXI BASEADDR C SOO AXI BASEADDR 0xFFFFFFF 32 b	bitString
IP Ports and Interfaces		vicouring.
✓ IP Addressing and Memory		
P GUI Customization		
Review and Package		

Figure 21. Customization Parameters.

Next, select the IP Ports and Interfaces. Notice that your new LED IP is there (Fig. 22).

∑ Project Summary × ♦ Package II	P•myLed × 🐵 myLed_v1_0_S30_AXLv × @ myLed_v1_0.v ×	□ 2 ×
P Packaging Steps	IP Ports and Interfaces	wizard more info
IP Identification	Image: Provide the state of the s	Size Left Dependency
✓ IP Competibility	Clock and Reset Signals	
✓ IP Fie Groups		
 IP Customization Parameters 	+	
✓ IP Ports and Interfaces	99 ·	
IP Addressing and Memory		
IP GUI Customization		
Review and Package		
		1

Figure 22. IP Ports and Interfaces.

Next, select IP GUI Customization. Our IP GUI is fine as is, so we won't make any changes here (Fig. 23).



P Packaging Steps ✓ IP GUI Customization ✓ IP Identification Preview	wizard more info
V IP Identification Preview	
 ✓ IP Competibility ✓ IP File Groups ✓ IP Customization Parameters ✓ IP Ports and Interfaces ✓ IP Addressing and Memory ✓ IP GUI Customization Review and Package 	

Figure 23. GUI Customization.

Now we can **Review and Package** our myLed IP. Select Review and Package IP and press the **Re-Package IP button**. Our IP is now completed and packaged.

12. We are going to add our IP to our design. Right click anywhere on the block design and click Add IP (as shown in Fig. 24). Select the correct IP, myLed_v1.0, and press enter (Fig. 23).







	LEDs_4Bits
 Search: Q→ myLed ② (2 matc Name 1 VLNV IF myLed_v1.0 dsf:user:. IF myled_v1.0 dsf:user:.	hes) SWs_4Bits
Select and press ENTER or drag and dro axi_dispctrl_1	эр,

Figure 25. Select IP.

13. The AXI4-Lite bus of myLed IP Core needs to be connected to the processing system. At the top of the window, click the blue text that says **Run connection automation** (Fig. 26). This will connect the inputs of the myLed IP Core. You should see that S_AXI is now connected to the first output of the AXI Interconnect.



Figure 26. Connect IP.

14. Next, we need to connect the myLed IP to an external port. The myLed IP Core that we implemented will not connect to the existing LEDs_4Bits port, so we need to make a new external port called led. Click on the existing LED port and press delete. To create the new port, right click and select **create port** (Fig. 27). Name the port, select output, select vector [3:0] and press enter.





Figure 27. Create Port.

Next, connect the LED port to the $\tt myLed$ IP by clicking on the port and dragging a connection to $\tt myLed$ (Fig. 28).



Figure 28. Connect IP to Port.

15. The final step is to specify the pin numbers for myled_0_LED_pin to physically connect our customized IP Core to the on-board LEDs. In the Project Manager, expand the Constraints section and select the **base.xdc** file (Fig. 29). Within that file, change the names of the external LED pins so that they match the name of our external led port (Fig. 30).



Project Manager - zybo_bsd
Sources _ D L ² ×
a 🔀 😂 🔂 📓 🛃
စု–ြာ Design Sources (11)
ø– ø▲ system_wrapper (system_wrapper.v) (1)
🐵- 🙏 system_i - system (system.bd) (24)
myLed_v1_0_hw_1_wrapper (myLed_v1_0_hw_1_wrapped)
myLed_v1_0_hw_1_i - myLed_v1_0_hw_1 (myLed_v1_)
o ि IP Update Log (9)
∲– 🗁 Constraints (1)
base.xdc (target)
φ− [−] sim_1 (2)
• • • • • • • • • • • • • • • • • • •
• myLed_v1_0_nw_1_wrapper (myLed_v1_0_hw_1_wra
Hierarchy IP Sources Libraries Compile Order
🕹 Sources 💡 Templates

Figure 29. Open XDC File.



Figure 30. Connect Port led to the LEDs on the ZYBO Board.

16. Regenerate the bitstream for the hardware design by clicking on **Generate Bitstream** under Program and Debug on the left side of the window.



2 Compile U-Boot (Optional)

2.1 Prerequisites

- Vivado 2014.1 WebPACK: available at the Xilinx Website <u>Download Page</u>.
- > ZYBO Base System Design: available at the Digilent Website on the <u>ZYBO Page</u>.

2.2 Instructions (Use the Master-Next Branch Until Further Notice)

1. Get the source code for U-Boot from the Digilent Git repository. There are two ways to retrieve the source code:

Using git command: If you have Git installed in your distribution, you can clone the repository to your computer by command Git clone: https://github.com/DigilentInc/u-boot-Digilent-Digilent-Digilent-Dev.git. The whole Git Repository is around 55MB, as shown in Fig. 31. If you want to get a separate branch, follow Fig. 32. The next contains the U-boot that is not yet released. The clone URL referenced above can be found on the Digilent GitHub page, as seen in Fig. 33.

[kfranz@DIGILENT_LINUX ~]\$ git clone https://github.com/DigilentInc/u-boot-Digil ent-Dev.git Initialized empty Git repository in /home/kfranz/u-boot-Digilent-Dev/.git/ remote: Reusing existing pack: 253078, done. remote: Counting objects: 10, done. remote: Compressing objects: 100% (10/10), done. remote: Total 253088 (delta 0), reused 0 (delta 0) Receiving objects: 100% (253088/253088), 69.54 MiB | 4.88 MiB/s, done. Resolving deltas: 100% (201962/201962), done.

Figure 31. U-Boot repository.

[kfranz@DIGILENT_LINUX ~]\$ git clone -b master-next https://github.com/DigilentI nc/u-boot-Digilent-Dev.git Initialized empty Git repository in /home/kfranz/u-boot-Digilent-Dev/.git/ remote: Reusing existing pack: 253078, done. remote: Counting objects: 10, done. remote: Compressing objects: 100% (10/10), done. remote: Total 253088 (delta 0), reused 0 (delta 0) Receiving objects: 100% (253088/253088), 69.54 MiB | 5.24 MiB/s, done. Resolving deltas: 100% (201962/201962), done. [kfranz@DIGILENT_LINUX ~]\$

Figure 32. Next-repository.



nonths ago	2
nonths ago	Network
nonths ago	HTTP\$ close USL
nonths ago	https://github.com
nonths ago	You can clone with HTTPS or
nonths ago	
nonths ago	Clone in Desktop
nonths ago	Download ZIP
nonths ago	
nonths ann	

Figure 33. Clone URL.

2. To compile U-Boot, we need cross-compile tools which are provided by Vivado 2014.1. Those tools have a prefix arm-xilinx-linux-gnueabi- to the standard names for the GCC tool chain. The prefix references the platforms that are used. The ZYBO board has two arm cores, so we reference arm. In order to use the cross-platform compilers, please make sure the Vivado 2014.1 settings have been sourced. If not, please refer to step 1 above. To configure and build U-Boot for ZYBO, follow Fig. 34.

```
[kfranz@DIGILENT_LINUX u-boot-Digilent-Dev]$ make CROSS_COMPILE=arm-xilinx-linux-gnueabi-
zynq_zybo_config
Configuring for zynq_ZYBO board...
[kfranz@DIGILENT_LINUX u-boot-Digilent-Dev]$ make CROSS_COMPILE=arm-xilinx-linux-gnueabi-
Generating include/autoconf.mk
Generating include/autoconf.mk.dep
. . .
arm-xilinx-linux-gnueabi-ld -gc-sections -Ttext 0x1000000 -o demo crt0.o demo.o libgenwrap.o
lent-Dev/arch/arm/lib/eabi_compat.o -L /opt/Xilinx/SDK/2014.1/gnu/arm/lin/bin/../lib/gcc/arm
arm-xilinx-linux-gnueavi-objcopy -O binary demo demo.bin 2>/dev/null
make[1]: Leaving directory /home/kfranz/Tutorial/u-boot-Digilent-Dev/examples/api'
[kfranz@DIGILENT_LINUX u-boot-Digilent-Dev]$
```

Figure 34. Compile U-Boot.

3. After the compilation, the ELF (Executable and Linkable File) generated is named u-boot. We need to add a .elf extension to the file name so that Xilinx SDK can read the file layout and generate BOOT.BIN. In this tutorial, we are going to move the u-boot.elf to the sd_image folder and substitute the u-boot.elf that comes along with the ZYBO Base System Design Package, as shown in Fig. 35.

[kfranz@DIGILENT_LINUX u-boot-Digilent-Dev]\$ cp u-boot ../zybo_base_system/sd_image/u-boot.elf
[kfranz@DIGILENT_LINUX u-boot-Digilent-Dev]\$

Figure 35. Add .elf.



3 Generate BOOT.BIN

3.1 Prerequisites

- Vivado 2014.1 WebPACK: available at the Xilinx Website <u>Download Page</u>.
- > ZYBO Base System Design: available at the Digilent Website on the <u>ZYBO Page</u>.
- Finished the hardware customization from <u>Section 1</u> and u-boot.elf from <u>Section 2</u> (Section 2 optional).

3.2 Instructions

 Export the hardware design (after Section 1, step 16) to Xilinx SDK by clicking on File -> Export -> Export Hardware for SDK..., as shown in Fig. 36.



Figure 36. Export Hardware Design to SDK.

 Leave the workspace as <Local to Project>. Make sure that the "Launch SDK" box is checked and click OK, as shown in Fig. 37.

Note: If you are using Vivado 2014.1, you may have to export twice.



λ.	Export Hardware for SDK	×	
Export ha Options	ardware platform for SDK.	_	
Source:	📥 system.bd 🔹		
Export to:	🙃 <local project="" to=""> 💌</local>		
<u>W</u> orkspace:	🖕 /home/kfranz/Tutorial/zybo_base_system/source 🔻		
🗹 Export <u>H</u> a	🛜 <local project="" to=""></local>		
🗹 Include bit	 /home/kfranz/Tutorial/zybo_base_system/source/vivado None 	i/hw	/zybo_bsd/zybo_bsd.sdk/SDK/SDK_Export
☑ Launch SD	ĸ		@ ×
	OK Cancel		

Figure 37. Set SDK Workspace Path.

 After SDK launches, the hardware platform project is already present in Project Explorer on the left of the SDK main window, as shown in Fig. 38. We now need to create a First Stage Bootloader (FSBL). Click File->New->Project..., as shown in Fig. 39.

Image: Solution of the second of	ြဲ Project Explorer 🕱 🛛 🗖 🗖	🗊 system.xml 🛙	- 0	🗄 Ou 🔉	💿 Ma	- 0
 Besign Information Besign Informatio	□ 🛓 🔻 🎽	hw_platform_0 Hardware Platform Specification	^	An outline	is not avai	ilable.
A Device: 72010 Created With: Vivado 2014.1 Created With: Vivado 2014.1 Created On: Fri May 23 120053 2014 Address Map for processor ps7_cortexa9_0 BTNs_4Bits 0x4120000 0x4120ffff Swy.4Bits 0x4120000 0x4120ffff Swy.4Bits 0x4120000 0x4120ffff axi_dispct1_1 0x4320000 0x4320ffff axi_dispct1_0 0x4320000 0x4320ffff axi_dispct1_0 0x4320000 0x4320ffff axi_dispct1_0 0x43200000 0x4320ffff ps7_aff_0 0x4300000 0x4300ffff ps7_aff_0 0x4300000 0x4300ffff ps7_aff_0 0x4300000 0x4300fff ps7_aff_0 0x43000000 0x4300fff ps7_aff_0 0x43000000 0x4300fff		Design Information				
inige: Trotter::::Nide inige: Trotter::::::::::::::::::::::::::::::::::	ps7_init.c	Tarrat EDGA Device: 72010	=			
Ps7_init.thmi Ps7_init.td Ps7_init.td Ps7_init.d	🖻 ps7_init.h	Croated With: Vivada 2014 1				
Address Map for processor ps7_cortex8_0 BY3_clinit.cl System_wrapper.bit System_wrapper.bit System.xml BTMs_4Bits 0x41220000 0x4120ffff SWs_4Bits 0x41220000 0x4320ffff axi_dispct1_0 0x4320000 0x4320ffff axi_dispct1_0 0x4320000 0x4320ffff axi_dispct1_0 0x4320000 0x4320ffff axi_dispct1_0 0x4320000 0x4320ffff ps7_afi_0 0xF800000 0x4300ffff ps7_afi_0 0xF800000 0xF800fff ps7_afi_0 0xF800000 0xF800fff ps7_afi_0 0xF800000 0xF800fff ps7_ddr_0 0x61000000 0xF800fff ps7_ddr_0 0x61000000 0xF800fff ps7_ddr_0 0xF800000 0xF800fff ps7_ddr_0 0xF	() ps7_init.html	Created On: Eri May 23 12:00:52 2014				
System_wrapper.bit Address Map for processor ps7_cortexa9_0 BTNs_4Bits 0x4120000 0x4120fff SW5_4Bits 0x4120000 0x4320fff axi_dispctrl_0 0x4320000 0x4320fff axi_dispctrl_0 0x4320000 0x4320fff axi_ydma_0 0x4300000 0x4320fff axi_ydma_1 0x4300000 0x4300fff ps7_afi_0 0xf800800 0xf800fff ps7_afi_2 0xf800800 0xf800fff ps7_ddr_0 0x0000 0xf800fff ps7_ddr_0 0x0000 0xf800fff ps7_ddr_0 0x680000 0xf800fff ps7_ddr_0 0x6800000 0xe0000fff ps7_ddr_0 0x6800000 0xe0000fff ps7_ddr_0 0x6800000 0xe0000fff ps7_dor_0 0x6800000 0xe0000fff ps7_dor_0 0x6800000 0xe00000fff ps7_dor_0 0x6800000 0xe00000fff <	⊗ ps7_init.tcl	Cicaca on. 111 May 25 12:00:55 2014				
Image: System.xml BTNs_4Bits 0x4120000 0x4120fff SWS_4Bits 0x41220000 0x4320ffff SWS_4Bits 0x41220000 0x4320ffff axi_idspct1_0 0x4300000 0x4301ffff axi_idspct1_1 0x4300000 0x4301fff axi_idspct1_1 0x4300000 0x4300fff ps7_aff_1 0xf800000 0xf8000fff ps7_aff_2 0xf800000 0xf8000fff ps7_aff_2 0xf800000 0xf8000fff ps7_dm_0 0xf800000 0xf8000fff ps7_gpv_0 0xf8000000 0xf8000f	📄 system_wrapper.bit	Address Map for processor ps7_cortexa9_0				
SWS_4Bits 0x4122000 0x122fff axi_dispct1_0 0x43c0000 0x132fff axi_dispct1_0 0x43c0000 0x132fff axi_2s_adi_0 0x1300000 0x1300fff axi_ydma_0 0x4300000 0x1300fff ps7_afl_0 0x1800000 ft8000fff ps7_afl_0 0x1800000 0x1800fff ps7_afl_0 0x1800000 0x1800fff ps7_der_0 0x1800000 0x1800fff ps7_jop_bus_config_0 0xe02000fff ps7_jop_bus_config_0 0xe02000fff ps7_jop_bus_config_0 0xe02000fff ps7_jop_bus_config_0 0xe02000fff ps7_jop_bus_config_0 0xe02000fff ps7_jop_bus_config_0 0xe02000fff ps7_jop_bus_config_0 0xe02000fff ps7_jop_bus_config_0 0xe02000fff ps7_jop_bus_config_0 0xe020000 0xe0200fff ps7_jop_bus_config_0 0xe020000 0xe0200ff ps7_jop_bus_config_0 0xe020000 0xe0200ff ps7_jop_bus_config_0 0xe02000 0xe0200ff ps7_jop	🖥 system.xml	BTNs_4Bits 0x41200000 0x4120ffff				
<pre>axi_dispctrl 0 &x43.c0000 0x43.c0fff axi_dispctrl 1 0x43.c0000 0x43.c2ffff axi_dispctrl 1 0x43.c0000 0x43.c2ffff axi_vdma_1 0 x4300000 0x4300fff ps7_afi_0 0x4800800 0x4800fff ps7_afi_0 x4800800 0x4800fff ps7_afi_0 x4800800 0x4800fff ps7_afi_0 x4800800 0x4800fff ps7_der_0 0x4800000 0x4800fff ps7_jen_0 0x4800000 0x48000fff ps7_jen_0 0x4800000 0x48000fff ps7_jen_0</pre>		SWs_4Bits 0x41220000 0x4122ffff				
<pre>axi_dispctri 1 &x43:10000 0x43:1fff axi_12s_adi_1 0x43:20000 0x43:0fff axi_vdma_0 x4300000 0x430:0fff axi_vdma_1 0x4300000 0x430:0fff axi_vdma_1 0x4300000 0x430:0fff ps7_afi_0 0xf800000 0xf800:0ff ps7_afi_2 0xf800000 0xf800:0ff ps7_afi_3 0xf800000 0xf80:0ff ps7_dtr_0 0x00000 0xf80:0ff ps7_dtr_0 0x6800000 0xf80:0ff ps7_dtr_0 0xf800000 0xf80:0ff ps7_jc_0 0xce00000 0xe00:0ff ps7_jc_0 0xce00000 0xe00:0ff ps7_</pre>		axi_dispctrl_0 0x43c00000 0x43c0ffff				
<pre>axi_l2s_adi_1 0x43c2000 0x43c2ffff axi_vdma_1 0x4301000 0x4300ffff axi_vdma_1 0x4301000 0x4300ffff ps7_afi_0 0xf8008000 0xf8008fff ps7_afi_0 0xf8008000 0xf8008fff ps7_afi_2 0xf8008000 0xf8008fff ps7_ddr_0 0x010000 0xf80ffff ps7_ddr_0 0x010000 0xf80ffff ps7_ddr_0 0xf800000 0xf800fff ps7_ddr_0 0xf800000 0xf800fff ps7_jobaltime_0 0xf800000 0xf800fff ps7_jobaltime_0 0xf800000 0xf800fff ps7_job_0xf800000 0xf800fff ps7_job_0xf80000</pre>		axi_dispctrl_1 0x43c10000 0x43c1ffff				
<pre>axi_vdma_0 0x4300000 0x4300fff axi_vdma_1 0x4301000 0x4301fff ps7_afi_0 0x6800000 0x6800fff ps7_afi_2 0x6800000 0x6800fff ps7_afi_2 0x6800000 0x6800fff ps7_afi_2 0x6800000 0x6800fff ps7_der_0 0x0100000 0x16800fff ps7_der_0 0x6800000 0x6800fff ps7_der_0 0x6800000 0x6800fff ps7_der_0 0x6800000 0x6800fff ps7_der_0 0x6800000 0x6800fff ps7_der_0 0x6800000 0x6800fff ps7_der_0 0x6800000 0x6800fff ps7_der_0 0x6800000 0x6800fff ps7_gip_Dus_config_0 0x60000fff ps7_inc_dist_0 0x6800000 0x6800fff ps7_inc_dist_0 0x6800000 0x68000fff ps7_inc_dist_0 0x6800000 0x6800fff ps7_inc_dist_0 0x68000000 0x6800fff ps7_inc_d</pre>		axi_i2s_adi_1 0x43c20000 0x43c2ffff				
<pre>axi_vdma_1 0x4301000 0x4301fff ps7_afi_0 0xf800900 0xf8008ffi ps7_afi_2 0xf800900 0xf8008ffi ps7_afi_3 0xf800b00 0xf8008ffi ps7_drc_0 0xf800000 0xf80ffi ps7_drc_0 0xf800000 0xf80ffi ps7_drc_0 0xf800f00 0xf800ffi ps7_drc_0 0xf800f00 0xf800ffi ps7_drc_0 0xf800000 0xf800ffi ps7_grc_0 0xf800000 0xf80ffi ps7_ichc_dist_0 0xf8f01000 0xf80ffi ps7_ichc_dist_0 0xf8f01000 0xf80ffi ps7_ichc_dist_0 0xf8f01000 0xf80ffi ps7_ichc_dist_0 0xf8f01000 0xf8ffiffi ps7_ichc_dist_0 0xf8ffiffiffiffiffiffiffiffiffiffiffiffiff</pre>		axi_vdma_0 0×43000000 0×4300ffff				
<pre>ps7_afi_0 0xf8008000 0xf8008ff ps7_afi_1 0xf800900 0xf800aff ps7_afi_2 0xf800x000 0xf800aff ps7_afi_2 0xf800x000 0xf800aff ps7_dr_0 0x0100000 0xf800ff ps7_dr_0 0x0100000 0xf800fff ps7_dr_ns_0 xf8004000 0xf800fff ps7_dr_s 0xf8004000 0xf800fff ps7_jc_0 0xf8000000 0xf800fff ps7_jc_0 0xf8000000 0xf800fff ps7_jc_0 0xf8000000 0xf800fff ps7_jc_0 0xf8000000 0xf800fff ps7_jc_0 0xf8000000 0xf800fff ps7_jc_0 0xf800000 0xf800fff ps7_jc_0 0xf8000000 0xf800fff ps7_jc_0 0xf8000000 0xf800fff ps7_jc_0 0xf8000000 0xf800fff ps7_jc_0 0xf800000 0xf80fff ps7_jc_0 0xf8000000 0xf80fff ps7_jc_0 0xf8000000 0xf80fff ps7_jc_0 0</pre>		axi_vdma_1 0x43010000 0x4301ffff				
<pre>ps7_afi_1 0xf800000 0xf8009fff ps7_afi_2 0xf800000 0xf800fff ps7_afi_2 0xf800000 0xf800fff ps7_afi_2 0xf800000 0xf80ffff ps7_ddr_0 0xf8000fff ps7_ddr_0 0xf8000fff ps7_ddr_0 0xf8000fff ps7_ddr_0 0xf800000 0xf800fff ps7_ddr_0 0xf800fff ps7_ddr</pre>		ps7_aff_0 0xf8008000 0xf8008fff				
<pre>ps7_afi_2 0xf800000 0xf800bff ps7_afi_3 0xf800b00 0xf880fff ps7_dcr_0 0x010000 0xf88ffff ps7_dcr_0 0x010000 0xf88ffff ps7_dcr_0 0xf800700 0xf800fff ps7_dcr_0 0xf800700 0xf800fff ps7_dma_s 0xf800700 0xf800fff ps7_dma_s 0xf800700 0xf800fff ps7_dma_s 0xf800700 0xf800fff ps7_globaltimer_0 0xf800200 0xf800fff ps7_jobaltimer_0 0xf800200 0xf80ffff ps7_ipb_us_config_0 0xe02000ff Overview Source ↓ Local [default] D items 32 @ Tasks © Console © Properties @ Terminal ♥ □ 0 items</pre>		ps7_afi_1 0xf8009000 0xf8009fff				
ps7_afi_3 0xf800b000 0xf800bff ps7_coresight_comp_0 0xf800000 0xf800fff ps7_ddr_0 0x00100000 0xf800fff ps7_ddr_0 0xf800000 0xf800fff ps7_ddr_0 0xf800000 0xf800fff ps7_ddr_s 0xf800000 0xf800fff ps7_der_s 0xf800000 0xf800fff ps7_icp_bus_config_0 0xf80000 0xf800fff ps7_icp_bus_config_0 0xf80000 0xf800fff ps7_icp_bus_config_0 0xf800000 0xf800fff ps7_icp_bus_config_0 0xf800000 0xf800fff ps7_icp_bus_config_0 0xf800000 0xf8000fff ps7_icp_bus_config_0 0xe000000 0xe000fff verview Source Problems 32 ③ Tasks Console Properties Ferminal		ps7_afi_2 0xf800a000 0xf800afff				
<pre>ps7_coresight_comp_0_0_xf8800000 xf88fffff ps7_ddr_0_0 xxf800000 xf800ffff ps7_ddr_0_0 xxf800000 xf800fff ps7_ddr_0_0 xxf800000 xxf800fff ps7_dma_s xxf8003000 xxf800fff ps7_dma_s xxf8003000 xxf800fff ps7_gthermet_0 0xe00b000 xx6004fff ps7_gthermet_0 0xe00b000 xx6004fff ps7_idr_0_0 xxf800000 xx6004fff ps7_idr_0_0 xxf800000 xx6004fff ps7_idr_0_0 xxf800000 xx6004fff ps7_idr_0_0 xx600000 xx60004fff ps7_idr_0_0 xx600000 xx60000 xx60004fff ps7_idr_0_0 xx600000 xx60000 xx60004fff ps7_idr_0_0 xx600000 xx60000 xx60004fff ps7_idr_0_0 xx600000 xx60000 xx60000 xx60004fff ps7_idr_0_0 xx600000 xx60000 xx60004fff ps7_idr_0_0 xx600000 xx60000 xx60000 xx60004fff ps7_idr_0_0 xx600000 xx60000 xx60004fff ps7_idr_0_0 xx600000 xx60000 xx600000 xx60000ff 0 xx60000 xx60000 xx60000 xx60000 xx60000 xx60000 xx60000ff 0 xx60000 xx60000 xx60000 xx60000 xx60000 xx600000 xx600000 xx600000 xx600000 xx600000 xx6000000 xx6000000 xx6000000 xx600000 xx600000000</pre>		ps7_afi_3 0xf800b000 0xf800bfff				
<pre>ps7_ddr_0 0x0100000 0x1fffffff ps7_ddr_0 0xf8006000 xf8006fff ps7_dev_cfg_0 0xf8007000 0xf80070ff ps7_dma_s 0xf8003000 0xf8003fff ps7_dma_s 0xf8003000 0xf8003fff ps7_dna_s 0xf8003000 0xf8003fff ps7_globaltimer_0 0xf8002000 0xf8003fff ps7_globaltimer_0 0xf8002000 0xf8003fff ps7_jop_0 0xf800000 0xe0200fff Overview Source </pre>		ps7_coresight_comp_0 0xf8800000 0xf88fffff				
bright for the state of t		ps7_ddr_0 0x00100000 0x1ffffff				
<pre>ps7_dev_cfg_0 0xf8007000 0xf80070ff ps7_dma_ns 0xf800400 0xf8003fff ps7_dma_s 0xf800400 0xf8003fff ps7_dma_s 0xf800400 0xf8003fff ps7_dma_s 0xf800400 0xf8003fff ps7_dna_s 0xf800400 0xf8001fff ps7_jc_0 0xf800000 0xf80fffff ps7_jc_0 0xf800000 0xf80ffff ps7_jc_0 0xf800000 0xf80fff ps7_jc_0 0xf800000 0xf80ffff ps7_jc_0 0xf800000 0xf80ffff ps7_jc_0 0xf800000 0xf80ffff ps7_jc_0 0xf800000 0xf80fff ps7_jc_0 0xf800000 0xf80fff ps7_jc_0 0xf800000 0xf80ffff ps7_jc_0 0xf800000 0xf80ffff ps7_jc_0 0xf800000 0xf80ffff ps7_jc_0 0xf800000 0xe6004fff ps7_jc_0 0xe600000 0xe6004fff ps7_jc_0 0xe6000000 0xe6004fff ps7_jc_0 0xe6000000 0xe6004ff ps7_jc_0 0xe6000000 0xe6004ff ps7_jc_0 0xe60000000000xe6004ff ps7_jc_0</pre>		ps7_ddrc_0 0xf8006000 0xf8006fff				
ps7_dma_ns 0xf8004000 0xf8004fff ps7_dma_s 0xf8003000 0xf8003fff ps7_dma_s 0xf8003000 0xf8003fff ps7_globaltimer_0 0xf800200 0xf8002ff ps7_globaltimer_0 0xf800000 0xe8004fff ps7_jop_0 0xf800000 0xe8004fff ps7_jop_bus_config_0 0xe8004000 0xe8004fff ps7_jop_bus_config_0 0xe800000 0xe8004fff ps7_jop_bus_config_0 0xe80000 0xe8004ff ps7_jop_bus_config_0 0xe80000 0xe8004ff ps7_jop_bus_config_0 0xe80000 0xe8004ff ps7_jop_bus_config_0 0xe800000 0xe8004ff ps7_jop_bus_config_0 0xe80000 0xe8004ff ps7_jop_bus_config_0 0xe804ff ps7_jop_bus_config_0 0xe804ff		ps7_dev_cfg_0 0xf8007000 0xf80070ff				
ps7_dm_s 0xf8003000 0xf8003ff ps7_ethernet_0 0xe000b00 0xe000bff ps7_globaltimer_0 0xf80020f ps7_gpv_0 0xf80020f ps7_ipz_0 0xf800000 0xf8004ff ps7_ipz_0 0xf800000 0xf8004ff ps7_ipz_bus_config_0 0xe02000ff Overview Source Verview Source Problems ⊠ @ Tasks © Console © Properties @ Terminal ♡ □ 0 0 items 0 items © Terminal ♡ □ 0 13:41:21 INF0 : Processing command line option -hwspec /hu		ps7_dma_ns 0xf8004000 0xf8004fff				
br7_ethermet_0 0xe000000 0xe0000ff ps7_globaltimer_0 0xf8f00200 0xf8f002ff ps7_gr_0v_0 0xf8f00200 0xf8f01ff ps7_lzc_0 0xe004000 0xe0004fff ps7_lzc_0 0xe004000 0xe0200ff overview Source verview Source Problems S2 @ Tasks © Console □ Properties @ Terminal ♥ □ □ 0 items 0 items 0 xf8f01000 0xe020000 0xe02000 0xe020000 0xe02000 0xe020000 0xe020000 0xe020000 0xe020000000000		ps7_dma_s_0xf8003000_0xf8003fff				
ps7_globaltimer_0 0xf8f00200 0xf8f002ff ps7_gpv_0 0xf8900000 0xf8f1fff ps7_jz_0 0xe004000 0xe004fff ps7_inc_dist_0 0xf8f01000 0xf8f1fff ps7_inc_dist_0 0xf8f01000 0xf8f1fff ps7_jop_bus_config_0 0xe0200000 0xe0200fff Overview Source Problems ⊠ ② Tasks © Console □ Properties ③ Terminal ♡ □ □ SDK Log ⊠ 13:41:21 INF0 : Processing command line option -hwspec /hu		ps7 ethernet 0 0xe000b000 0xe000bfff				
ps7_gpv_0 0xf890000 0xf89ffff ps7_jzc_0 0xe8004000 0xe004fff ps7_inc_dist_0 0xf801000 0xf80ffff ps7_inc_dist_0 0xf801000 0xe000fff Overview Source Problems X Tasks Console Properties Terminal		ps7_globaltimer_0 0xf8f00200 0xf8f002ff				
ps7_i2c_0 0xe000000 0xe0004ff ps7_intc_dist_0 0xf801000 0xf8011ff ps7_intc_dist_0 0xf801000 0xf8011ff ps7_iop_bus_config_0 0xe02000ff Overview Source verview Source v		ps7_gpv_0 0xf8900000 0xf89ffff				
ps7_intc_dist_0 0 xxf8f01000 0xf8f011000 0xf8f011ff ps7_intc_dist_0 0 xxf8f01000 0xf8f011ff ps7_intc_dist_0 0 xxf8f01000 0xe020000ff overview [Source] verview [Source] 0 items 0 items 0 items 13:41:21 INF0 13:41:21 INF0		ps7 i2c_0 0xe0004000 0xe0004fff				
Target Connections X Ps7_iop_bus_config_0 0xe0200000 0xe0200fff Overview Source Verview Source Image: Console in Properties In Terminal in the second operation of the second operation of the second operation		ps7 intc dist 0 0xf8f01000 0xf8f01fff				
Image: connections of the second of the s	Target Connections 🔉 📃 🗖	ps7_iop_bus_config_0 0xe0200000 0xe0200fff				
Local [default]		Overview Source	<u> </u>			
Local [default]						
0 trems 13:41:21 INF0 : Processing command line option -hwspec /h	🚽 Local [default]					•
Description Resource Path		13:41:21 INFO : Processing command	lin	e option	-hwspec /	/hon
resource rau		Description Resource Path				
						=
	〕□◆					

Figure 38. Export hardware design to SDK.



30 K								C/C++ - h	w_platfo	orm
Ei	e <u>E</u> dit	<u>S</u> ource	Refactor	<u>N</u> avigate	Se <u>a</u> rch	<u>R</u> un	<u>P</u> roject	<u>X</u> ilinx Tools	<u>W</u> indow	H
	New						Applicati	on Project		
	Open Fil	e"					Board Su	pport Package	2	
	Close				Ctrl+W	/	Propert			
	Close Al	I		Shif	ft+Ctrl+W	1	Source Fo	older		
	Save				Ctrl+S	5	Folder			
	Save As						Source Fi	le		
	Sav <u>e</u> All			Shi	ift+Ctrl+S	5	Header F	ile		
	Revert						File from	Template		
	Move					-	Class			
	Rename				F2	2	<u>O</u> ther		Ctrl+	N

Figure 39. Create new project in SDK.

4. In the New Project window, select Xilinx->Application Project, and then Click Next (Fig. 40).

🐵 New Project	×
Select a wizard	
<u>W</u> izards:	
type filter text	4
👂 🗁 General	
▷ 🗁 C/C++	
▽ 🗁 Xilinx	
🚇 Application Project	
🚯 Board Support Package	
Hardware Platform Specification	
(?) < Back <u>N</u> ext > Cancel	<u>F</u> inish

Figure 40. Select Application Project Wizard.

5. We will name the project FSBL. Select **hw_platform_0** for **Target Hardware** because it is the hardware project we just exported. Select **standalone** for **OS Platform**. Click **Next**, as shown in Fig. 41.



50K	New Project ×
Application Project	
Create a managed make	application project.
Project name: FSBL	
✓ Use <u>d</u> efault location	
Location: /home/kfranz/	Tutorial/zybo_base_system/source/vivado/hw/z
Choose file sys	tem: default \$
Target Hardware	
Hardware Platform hw	platform 0
Processor	
processor	
-Target Software	
OS Platform	standalone
Language	● C ○ C++
Board Support Package	<u>C</u> reate New FSBL_bsp
	○ <u>U</u> se existing
0	- Pack Next > Cancel Einich

Figure 41. New Application Project.

6. Select **Zynq FSBL** as template, and click Finish as shown in Fig. 42.



Figure 42. Select Zynq FSBL as template.



7. For the ZYBO, we need to set the mac address for the Ethernet in the fsbl hook. We want the mac address for the Ethernet to remain constant when we turn the ZYBO board off and on. You can swap the fsbl_hooks.c file in the FSBL project with the fsbl_hooks.c under source/vivado/SDK/fsbl in the ZYBO Base System Design (Fig. 43).



Figure 43. fsbl_hooks.c

8. After you have saved the changes to fsbl_hooks.c, the project will rebuild itself automatically. If it does not rebuild, click **Project->Clean** to clean the project files, and **Project->Build All** to rebuild all the projects. The compiled ELF file is located in:

ZYBO_base_system/source/vivado/hw/ZYBO_bsd.sdk/SDK/SDK_Export/FSBL/Debug

 Now we have all of the files ready to create BOOT.BIN. Click Xilinx Tools -> Create Zynq Boot Image, as shown in Fig. 44.



۰		
<u>File Edit Source Refactor Naviga</u>	te Se <u>a</u> rch <u>R</u> un <u>P</u> roject	Xilinx Tools Window Help
	x & @* @* C*	Generate linker script Board Support Package Settings
ြဲ Project Explorer 🛿 🧧 🗖	🔓 system.xml 🔒 sys	<u>R</u> epositories
🖻 🔄 🔻 🎽	FSBL_bsp Board S	Program <u>F</u> PGA
🗢 🛃 FSBL		Program Flash
👂 🖑 Binaries	Modify this BSP's Setti	XMD Console
Includes		Launch Shell
Debug	Target Information	Configure JTAG Settings
Src	This Board Support Pack	System Generator Co-Debug Settings
	Hardware Specification:	Create Zynq Boot Image 🙀 🙀
N + RSB Documentation	Target Processor:	ps7 cortexa9 0

Figure 44. Create Zynq Boot Image.

 In the Create Zynq Boot Image window (as shown in Fig. 45), Click Browse to set the path for FSBL elf. Click Add to add the system.bit file found at:

/ZYBO_base_system/source/vivado/hw/ZYBO_bsd/ZYBO_bsd.sdk/SDK/SDK_Export /hw_platform_0/.Click Add to add the u-boot.elf file found at:

ZYBO_base_system/sd_image/. It is very important that the 3 files are added in this order, or else the FSBL will not work properly (the proper order can be seen in Fig. 45). It is also very important that you set FSBL.elf as the bootloader and system.bit and u-boot.elf as data files. In this tutorial, the sd image folder is set as output folder for the BIN file. Click **Create Image**.

• Create new	BIF file \bigcirc Import from existin	g BIF file		
BIF file path	/home/kfranz/Tutorial/zybo_ba	ase_system/sd_imag	e/output.bif	BI
🗆 Use Auther	itication			
Authenticatio	n keys			
РРК		Browse	PSK	Bro
SPK		Browse	SSK	Bro
SPK Signatur	<u></u>	Browse		
Key store Part name	BRAM O EFUSE			
File path	rtitions	Encrypted	Authenticated	
(bootloader)	home/kfranz/Tutorial/zybo_base	_syster none	none	
/home/kfranz	Tutorial/zybo_base_system/sou	rce/viva none	none	
the second states in the second states and t	Tutorial/zybo_base_system/su_	image/c none	none	
/home/kfranz				
/home/kfranz				

Figure 45. Create Zynq Boot Image Configuration.



11. The created BIN file was named BOOT.bin.

4 Compile Linux Kernel

4.1 **Prerequisites**

- Vivado 2014.1 WebPACK: available at the Xilinx Website <u>Download Page</u>.
- > ZYBO Base System Design: available at the Digilent Website on the <u>ZYBO Page</u>.

4.2 Instructions (Use the Master-Next Branch Until Further Notice)

1. Get the Linux kernel source code from Digilent Git repository. There are two ways to retrieve the source code:

Using git command: If you have Git installed in your distribution, you can clone the repository to your computer by command git clone https://github.com/DigilentInc/Linux-Digilent-Dev.git
The whole Git Repository is around 850MB, as shown in Fig. 46.

```
[kfranz@DIGILENT_LINUX ~]$ git clone https://github.com/DigilentInc/Linux-Digile
nt-Dev.git
Initialized empty Git repository in /home/kfranz/Linux-Digilent-Dev/.git/
remote: Counting objects: 3586185, done.
remote: Compressing objects: 100% (549192/549192), done.
remote: Total 3586185 (delta 3007223), reused 3586185 (delta 3007223)
Receiving objects: 100% (3586185/3586185), 864.81 MiB | 2.76 MiB/s, done.
Resolving deltas: 100% (3007223/3007223), done.
[kfranz@DIGILENT_LINUX ~]$
```

Figure 46. Clone Kernel.

2. We will start to configure the kernel with the default configuration for ZYBO. The configuration is located at arch/arm/configs/xylinx_zynq_defconfig. To use the default configuration, you can follow Fig. 47.

[kfranz@DIGILENT_LINUX Linux-Digilent-Dev]\$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- xilinx_zynq_defconfig # # configuration written to .config # [kfranz@DIGILENT_LINUX Linux-Digilent-Dev]\$ ■

Figure 47. Default Configuration.

3. Follow Fig. 48 to compile the Linux Kernel.

```
[kfranz@DIGILENT_LINUX Linux-Digilent-Dev]$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-
scripts/kconfig/conf --silentoldconfig Kconfig
CHK include/config/kernel.release
CHK include/generated/uapi/linux/version.h
CHK include/generated/utsrelease.h
```



Kernel: arch/arm/boot/Image is ready LD arch/arm/boot/compressed/vmlinux OBJCOPY arch/arm/boot/zImage Kernel: arch/arm/boot/zImage is ready Building modules, stage 2. MODPOST 23 modules [kfranz@DIGILENT LINUX Linux-Digilent-Dev]\$

Figure 48. Compile Kernel.

4. After the compilation, the kernel image is located at arch/arm/boot/zImage. However, in this case the kernel image has to be a uImage (unzipped) rather than a zimage. To make the uimage, follow Fig. 49.

[kfranz@DIGILENT_LINUX Linux-Digilent-Dev]\$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- UIMAGE_LOADADDR=0x8000 uImage CHK include/config/kernel.release CHK include/generated/uapi/linux/version.h CHK include/generated/utsrelease.h make[1]: `include/generated/mach-types.h' is up to date.

Data Size: 3819232 Bytes = 3729.72 kB = 3.64 MB Load Address: 00008000 Entry Point: 00008000 Image arch/arm/boot/uImage is ready [kfranz@DIGILENT_LINUX Linux-Digilent-Dev]\$

Figure 49. Create ulmage.

Note: Depending on your distribution of Linux, you may get an error regarding the path of the mkimage. If this is the case, you can change the path following Fig. 50.

[kfranz@DIGILENT_LINUX_Tutorial]\$ PATH=\$PATH:/home/kfranz/Tutorial/u-boot-Digilent-Dev/tools/ [kfranz@DIGILENT_LINUX_Tutorial]\$ echo \$PATH

Figure 50. Change Path.

5 Test Kernel Image with Pre-built File System

5.1 Prerequisites

- Vivado 2014.1 WebPACK: available at the Xilinx Website <u>Download Page</u>.
- Linux Kernel Source Code: available at Digilent GitHub repository <u>https://github.com/DigilentInc/Linux-Digilent-Dev</u>. (Use the Master-Next Branch Until Further Notice)
- > Pre-built File System Image: ramdisk Image is available in ZYBO Linux Reference Design.
- BOOT.BIN from Section 3, ulmage from Section 4.

5.2 Instructions

1. To boot the Linux operating system on the ZYBO, you need BOOT.BIN, a Linux kernel image (ulmage), a device tree blob (DTB file), and a file system. BOOT.BIN has been created in Section 3 and ulmage has been compiled in Section 4. We will now compile the DTB file. The default device tree source file is



located in the Linux Kernel source at arch/arm/boot/dts/zynq-ZYBO.dts. **RAMDISK**: modify the device tree source file according to Fig. 51. For Zynq, only the ramdisk image has to be wrapped in a u-boot header in order for u-boot to boot with it. This is shown in Fig. 52.

```
48 chosen {
49     /* bootargs = "console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk
rootfstype=ext4 rootwait devtmpfs.mount=1"; */
50     bootargs = "console=ttyPS0,115200 root=/dev/ram rw initrd=0x800000,8M
init=/init earlyprintk rootwait devtmpfs.mount=1";
51     linux,stdout-path = "/axi@0/serial@e0001000";
52  };
```

Figure 51. Edit device tree.



Figure 52. Make Uramdisk Image.

2. Generate DTB file, as shown in Fig. 53.

```
[kfranz@DIGILENT_LINUX Linux-Digilent-Dev]$ ./scripts/dtc/dtc -I dts -0 dtb -0 .
./devicetree.dtb arch/arm/boot/dts/zynq-zybo.dts
[kfranz@DIGILENT_LINUX Linux-Digilent-Dev]$
```

Figure 53. Generate DTB File.

3. **(RAMDISK)** Copy BOOT.BIN, devicetree.dtb, uImage and uramdisk.image.gz to the first partition of an SD card, as shown in Fig. 54.

```
[kfranz@DIGILENT_LINUX Tutorial]$ ls
devicetree.dtb linux-digilent-dev u-boot-digilent ZYBO_base_system
[kfranz@DIGILENT_LINUX Tutorial]$ cp ZYBO_base_system/sd_image/BOOT.BIN /media/ZYBO_BOOT/
[kfranz@DIGILENT_LINUX Tutorial]$ cp ZYBO_base_system/sd_image/ uramdisk.image.gz /BOOT.BIN
/media/ZYBO_BOOT/
[kfranz@DIGILENT_LINUX Tutorial]$ cp ./devicetree.dtb /media/ZYBO_BOOT/
[kfranz@DIGILENT_LINUX Tutorial]$ cp Linux-Digilent-Dev/arch/arm/boot/uImage /media/ZYBO_BOOT/
[kfranz@DIGILENT_LINUX Tutorial]$
```

Figure 54. Ramdisk.

4. Plug the SD card into the ZYBO. To boot from the SD card, jumper 7 needs to be configured for USB, as shown on the ZYBO board, and Jumper 5 must be connected to SD. Connect UART port to PC with a micro USB cable and set the UART terminal on PC to 115200 baud rate, 8 data bits, 1 stop bit, no parity, and no flow control. After powering on the board, the console (shown in Fig. 55) should be seen at the UART terminal if you use RamDisk. More information about these file systems can be found in *Getting Started with Embedded Linux - ZYBO*.





Figure 55. Ramdisk, UART Console after boot up.

6 Modify Device Tree and Compose Kernel Driver

6.1 Prerequisites

- Vivado 2014.1 WebPACK: available at the Xilinx Website <u>Download Page</u>.
- Linux Kernel Source Code: available at Digilent GitHub repository <u>https://github.com/Digilentinc/Linux-Digilent-Dev</u> (Use the Master-Next Branch Until Further Notice)

6.2 Instructions

1. Create a directory named "drivers" in the Tutorial folder, as shown in Fig. 56. Inside the driver's directory, we will compose the driver for the myLed controller.

[kfranz@DIGILENT_LINUX Tutorial]\$ mkdir drivers 🧲						
[KTFANZ@DIGILENI_LINUX_IUTOFIAL]\$ [S						
BOOT.bin	output.bif	vivado.jou				
drivers	ps_clock_registers.log	vivado.log				
Linux-Digilent-Dev	u-boot-Digilent-Dev	zybo_base_system				
[kfranz@DIGILENT_LINUX Tutorial]\$						

Figure 56. Driver Directory.

2. We need a Makefile so that we can compile the driver. The Makefile is created in Fig. 57.

[kfranz@DIGILENT_LINUX Tutorial]\$ cd drivers [kfranz@DIGILENT LINUX drivers]\$ vim Makefile

Figure 57. Create Makefile.

After creating the file, hit I to change to insert mode and insert the following text (Fig. 58).





Note: make sure the spacing in the Makefile is made up of tabs, not spaces, where necessary. Then hit esc to exit insert mode and :x to save the file and exit vim editor.

We will start with a simple driver that creates a file named myled under the Linux /proc file system. The status of the on-board LEDs can be changed by writing a number to the file. The driver is coded in Fig. 59.

[kfranz@DIGILENT LINUX drivers]\$ vim myled.c

Figure 59. Create myled.c



```
1 #include <linux/kernel.h>
 2 #include <linux/module.h>
 3 #include <asm/uaccess.h> /* Needed for copy_from_user */
                            /* Needed for IO Read/Write Functions */
 4 #include <asm/io.h>
 5 #include <linux/proc fs.h> /* Needed for Proc File System Functions */
 6 #include <linux/seq file.h> /* Needed for Sequence File Operations */
 7 #include <linux/platform device.h> /* Needed for Platform Driver Functions */
 8
 9 /* Define Driver Name */
10 #define DRIVER NAME "myled"
11
12 unsigned long *base addr; /* Vitual Base Address */
                            /* Device Resource Structure */
13 struct resource *res;
14 unsigned long remap_size; /* Device Memory Size */
15
16 /* Write operation for /proc/myled
17 * -----
18 * When user cat a string to /proc/myled file, the string will be stored in
19 * const char user *buf. This function will copy the string from user
20 * space into kernel space, and change it to an unsigned long value.
   * It will then write the value to the register of myled controller,
21
22 * and turn on the corresponding LEDs eventually.
23 */
24 static ssize t proc myled write (struct file *file, const char user * buf,
25
                  size t count, loff t * ppos)
26 {
27
      char myled phrase[16];
28
      u32 myled value;
29
30
      if (count < 11) {
31
          if (copy from user(myled phrase, buf, count))
32
              return -EFAULT;
33
34
          myled phrase[count] = '\0';
35
      }
36
37
      myled_value = simple_strtoul(myled_phrase, NULL, 0);
38
      wmb();
39
      iowrite32(myled_value, base_addr);
40
      return count;
41 }
42
43 /* Callback function when opening file /proc/myled
44 * ------
45 * Read the register value of myled controller, print the value to
46 * the sequence file struct seq_file *p. In file open operation for /proc/myled
47
   * this callback function will be called first to fill up the seq file,
   * and seq read function will print whatever in seq file to the terminal.
48
49 */
50 static int proc myled show(struct seq file *p, void *v)
51 {
52
      u32 myled_value;
53
      myled value = ioread32(base addr);
54
      seq printf(p, "0x%x", myled value);
55
      return 0;
56 }
57
```

Figure 60. myled.c



```
58 /* Open function for /proc/myled
 59 * -----
 60 * When user want to read /proc/myled (i.e. cat /proc/myled), the open function
 61 * will be called first. In the open function, a seq file will be prepared and the
 62 * status of myled will be filled into the seq file by proc myled show function.
 63 */
 64 static int proc myled open(struct inode *inode, struct file *file)
 65 {
 66
       unsigned int size = 16;
 67
       char *buf;
       struct seq file *m;
 68
 69
       int res;
 70
 71
       buf = (char *)kmalloc(size * sizeof(char), GFP_KERNEL);
72
       if (!buf)
 73
           return -ENOMEM;
 74
 75
       res = single open(file, proc myled show, NULL);
 76
 77
       if (!res) {
 78
           m = file->private data;
 79
           m \rightarrow buf = buf;
 80
           m->size = size;
 81
       } else {
82
           kfree(buf);
83
       }
84
85
       return res;
86 }
87
 88 /* File Operations for /proc/myled */
89 static const struct file operations proc myled operations = {
 90
       .open = proc myled open,
       .read = seq_read,
 91
92
       .write = proc_myled_write,
93
       .llseek = seq lseek,
94
       .release = single release
95 };
96
 97 /* Shutdown function for myled
98 * -----
99 * Before myled shutdown, turn-off all the leds
100 */
101 static void myled shutdown(struct platform device *pdev)
102 {
103
       iowrite32(0, base_addr);
104 }
105
```

Figure 60. myled.c (Cont.)



```
106 /* Remove function for myled
107 * -----
108 * When myled module is removed, turn off all the leds first,
109 * release virtual address and the memory region requested.
110 */
111 static int myled remove (struct platform device *pdev)
112 {
113
       myled shutdown (pdev) ;
114
115
       /* Remove /proc/myled entry */
       remove proc entry (DRIVER NAME, NULL);
116
117
118
       /* Release mapped virtual address */
119
       iounmap(base_addr);
120
       /* Release the region */
121
122
       release mem region(res->start, remap size);
123
124
       return 0;
125 }
126
127 /* Device Probe function for myled
128 * ------
129 * Get the resource structure from the information in device tree.
130 * request the memory regioon needed for the controller, and map it into
131 * kernel virtual memory space. Create an entry under /proc file system
132 * and register file operations for that entry.
133 */
134 static int myled probe(struct platform device *pdev)
135 {
136
       struct proc_dir_entry *myled_proc_entry;
137
       int ret = 0;
138
139
       res = platform get resource(pdev, IORESOURCE MEM, 0);
140
       if (!res) {
141
           dev err(&pdev->dev, "No memory resource\n");
142
           return -ENODEV;
143
       }
144
145
       remap_size = res->end - res->start + 1;
146
       if (!request mem region(res->start, remap size, pdev->name)) {
147
           dev err(&pdev->dev, "Cannot request IO\n");
148
           return -ENXIO;
149
       }
150
151
       base addr = ioremap(res->start, remap size);
       if (base addr == NULL) {
152
153
           dev_err(&pdev->dev, "Couldn't ioremap memory at 0x%081x\n",
154
                (unsigned long)res->start);
155
           ret = -ENOMEM;
156
           goto err_release_region;
157
       }
158
```

Figure 60. myled.c (Cont.)



```
159
        myled proc entry = proc create(DRIVER NAME, 0, NULL,
                           &proc myled operations);
160
161
        if (myled proc entry == NULL) {
            dev_err(&pdev->dev, "Couldn't create proc entry\n");
162
163
            ret = -ENOMEM;
164
            goto err_create_proc_entry;
165
        }
166
        printk(KERN INFO DRIVER NAME " probed at VA 0x%081x\n",
167
168
               (unsigned long) base addr);
169
170
        return 0;
171
172
    err create proc entry:
173
       iounmap(base_addr);
174
    err release region:
175
        release mem region(res->start, remap size);
176
177
        return ret;
178 }
179
180 /* device match table to match with device node in device tree */
181 static const struct of device id myled of match[] = {
        {.compatible = "dglnt,myled-1.00.a"},
182
183
        {},
184 };
185
186 MODULE DEVICE TABLE (of, myled of match);
187
188 /* platform driver structure for myled driver */
189 static struct platform driver myled driver = {
190
       driver = {
191
               .name = DRIVER NAME,
192
               .owner = THIS MODULE,
193
               .of match table = myled of match},
194
      .probe = myled probe,
195
       .remove = myled remove,
196
        .shutdown = myled_shutdown
197 };
198
199 /* Register myled platform driver */
200 module platform driver(myled driver);
201
202 /* Module Infomations */
203 MODULE AUTHOR("Digilent, Inc.");
204 MODULE LICENSE ("GPL");
205 MODULE DESCRIPTION (DRIVER NAME ": MYLED driver (Simple Version)");
206 MODULE ALIAS (DRIVER NAME) ;
207
```

Figure 60. myled.c (Cont.)



4. Compile and generate the driver module using make (as shown in Fig. 61). Don't forget to source Vivado settings.

```
[kfranz@DIGILENT_LINUX drivers]$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-
make -C ../Linux-Digilent-Dev/ M=/home/kfranz/Tutorial/drivers modules
make[1]: Entering directory `/home/kfranz/Tutorial/Linux-Digilent-Dev'
CC [M] /home/kfranz//Tutorial/drivers/myLed.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/kfranz/Tutorial/drivers/myLed.mod.o
LD [M] /home/kfranz/Tutorial/drivers/myLed.ko
make[1]: Leaving directory `/home/kfranz/Tutorial/Linux-Digilent-Dev'
[kfranz@DIGILENT_LINUX drivers]$
```

Figure 61. Compile Driver.

5. We need to add the **myLed** device node into the device tree. Make a copy of the default device tree source in the drivers folder, and modify it according to Fig. 62. The compatibility string of the node is the same as we define in the driver source code (myled.c: line 182). The reg property defines the physical address and size of the node. The address here should match with the address of the myLed IP Core in the address editor tab of the Vivado design, as shown in Fig. 63.

🚰 Diagram 🗙 🗟 Address Editor 🗙 🎯 system_wrapper.v 🗙 👎 IP Catalog 🗙								
0	Cell	Interface Pin	Base Name	Offset Address	Range	High Address		
-	P-∯ processing_system7_0							
	🗣 🖽 Data (32 address bits : 4G)							
	axi_dispctrl_0	S_AXI	S_AXI_reg	0x43C00000	64K 👻	0x43C0FFFF		
net	axi_dispctrl_1	S_AXI	S_AXI_reg	0x43C10000	64K 🔻	0x43C1FFFF		
655	— 🚥 axi_i2s_adi_1	S_AXI	S_AXI_reg	0x43C20000	64K 🔻	0x43C2FFFF		
	– ∞ axi_vdma_1	S_AXI_LITE	Reg	0x43010000	64K 👻	0x4301FFFF		
	— 🚥 BTNs_4Bits	S_AXI	Reg	0x41200000	64K 🔻	0x4120FFFF		
	— 🚥 SWs_4Bits	S_AXI	Reg	0x41220000	64K 🔻	0x4122FFFF		
	— 🚥 axi_∨dma_0	S_AXI_LITE	Reg	0x43000000	64K 🔻	0x4300FFFF		
	└ 🚥 myLed_1	S00_AXI	S00_AXI_reg	0x43C30000	64K 🔻	0x43C3FFFF		
	∲-⊈ axi_vdma_1							
	P- B Data_MM2S (32 address bits : 4G)							
	└─ == processing_system7_0	S_AXI_HP0	HP0_DDR_LO	0x00000000	512M 🔻	0x1FFFFFFF		
	- 🖽 Data_S2MM (32 address bits : 4G)							
	Land Bata_SG (32 address bits : 4G)							
	∲-∯ axi_vdma_0							
	🗣 🖽 Data_MM2S (32 address bits : 4G)							
	└─ == processing_system7_0	S_AXI_HP0	HP0_DDR_LO	0x00000000	512M 🔻	0x1FFFFFFF		
	— III Data_S2MM (32 address bits : 4G)							
	Data_SG (32 address bits : 4G)							

Figure 62. Physical Address for myLed IP Core.

[kfranz@DIGILENT_LINUX drivers]\$ cp ../Linux-Digilent-Dev/arch/arm/boot/dts/zynq-ZYBO.dts ./ [kfranz@DIGILENT_LINUX drivers]\$ vim zynq-ZYBO.dts

Figure 63. Edit device tree.



```
549
                spi-speed-hz = <4000000>;
                spi-sclk-gpio = <&ps7 gpio 0 59 0>;
550
551
                spi-sdin-gpio = <&ps7_gpio_0 60 0>;
552
            };
553
554
            myled {
555
                compatible = "dglnt,myled-1.00.a";
556
                reg = <0x43c30000 0x10000>;
557
            };
558
        };
559 };
```

Figure 64. zynq-ZYBO.dts

6. Recompile the device tree blob as shown in Fig. 65.

```
[kfranz@DIGILENT_LINUX drivers]$ ../Linux-Digilent-Dev/scripts/dtc/dtc -I dts -O dtb -o
devicetree.dtb zynq-ZYBO.dts
DTC: dts->dtb on file "zynq-ZYBO.dts"
[kfranz@DIGILENT_LINUX drivers]$
```

Figure 65. Compile DTB.

7. Copy these two files to the first partition of the SD card, as shown in Fig. 66. We are ready to test our driver on-board now.

```
[kfranz@DIGILENT_LINUX drivers]$ ls
devicetree.dtb Makefile Module.symvers myled.ko myled.mod.o
zynq-ZYBO.dts modules.order myled.c myled.mod.c myled.o
[kfranz@DIGILENT_LINUX drivers]$ cp myled.ko /media/ZYBO_BOOT/d
[kfranz@DIGILENT_LINUX drivers]$ cp devicetree.dtb /media/ZYBO_BOOT/
[kfranz@DIGILENT_LINUX drivers]$
```

Figure 66. Copy files to SD.

8. Plug the SD card into the ZYBO and we can start testing our driver. Use the insmod command to install the driver module into the kernel. After the driver is installed, an entry named myled will be created under the /proc file system. Writing 0x0F to /proc/myled will light up LED 0~3. You can either remove the driver with command rmmod or power off the system by command poweroff. In both cases, all of the LEDs will be turned off, as shown in Fig. 67. For instructions on using the terminal with the ZYBO, please refer to Section 5, Step 4 or the Section Boot from SD in Getting Started with Embedded Linux – ZYBO.



```
U-Boot 2012.04.01-dirty (June 30 2014 - 12:52:36)
DRAM: 512 MiB
WARNING: Caches not enabled
MMC: SDHCI: 0
Using default environment
. . .
reading uImage
2457328 bytes read
reading devicetree.dtb
9728 bytes read
reading uramdisk.image.gz
3694108 bytes read
## Starting application at 0x00008000 ...
Uncompressing Linux... done, booting the kernel.
    0.000000] Booting Linux on physical CPU 0
ſ
    0.000000] Linux version 3.6.0-digilent-13.01-00002-g06b3889 (kfranz@DIGILENT LINUX)
Γ
(qcc version 4.6.3 (Sourcery CodeBench Lite 2012.03-79) ) #1 SMP PREEMPT Sun June 30
23:54:12 PST 2014
. . .
rcS Complete
zynq> mount /dev/mmcblk0p1 /mnt/
zynq> cd /mnt/
zynq> ls
BOOT.BIN
                   devicetree.dtb uramdisk.image.gz
myled.ko
                    uImage
zynq> insmod myled.ko
[ 122.160000] myled probed at va 0xe0d20000
zynq> ls /proc
             567
                          9
1
                                        fs
                                                     partitions
10
             582
                          asound
                                       interrupts
                                                     scsi
                                       iomem
11
             588
                          buddyinfo
                                                     self
                         bus
12
             594
                                       ioports
                                                     slabinfo
13
             595
                                       irq
                          cmdline
                                                     softirqs
                         config.gz kallsyms
14
             596
                                                    stat
15
             6
                         consoles
                                      kmsq
                                                     swaps
2
             608
                                       kpagecount
                          сри
                                                     sys
                                    kpageflags sysvipc
loadavg timer_l
3
             614
                         cpuinfo
317
             615
                         crypto
                                                   timer list
318
             621
                         device-tree locks
                                                   tty
333
             641
                          devices meminfo
                                                     uptime
4
             642
                          diskstats misc
                                                     version
                          dma
429
             643
                                      modules
                                                     vmallocinfo
             647
440
                          dri
                                       mounts
                                                     vmstat
441
             652
                          driver
                                       mtd
                                                     zoneinfo
                          execdomains myled
5
             653
515
             7
                          fb
                                       net
548
             8
                           filesystems pagetypeinfo
zynq> echo 0x0F > /proc/myled
zynq> cat /proc/myled
0x0f
zynq> mkdir -p /lib/modules/`uname -r`
zynq> cp myled.ko /lib/modules/`uname -r`
zynq> rmmod myled
```

Figure 67. RAMDISK



7 User Application

7.1 Prerequisites

Vivado 2014.1 WebPACK: available at the Xilinx Website <u>Download Page</u>.

7.2 Instructions

 In this section, we will write a user application that makes the LEDs blink by writing to /proc/myled. Create a directory named user_app in the Tutorial folder, as shown in Fig. 68. Inside the user_app directory, we will compose the led_blink.c, as shown in Fig. 69.

```
[kfranz@DIGILENT_LINUX Tutorial]$ mkdir user_app
[kfranz@DIGILENT_LINUX Tutorial]$ ls
devicetree.dtb drivers linux-digilent u-boot-digilent user_app ZYBO_base_system
[kfranz@DIGILENT_LINUX Tutorial]$
```

Figure 68. User_app

[kfranz@DIGILENT_LINUX user_app]\$ vim led_blink.c

Figure 69. led_blink

```
0 #include <stdio.h>
 1 #include <stdlib.h>
 2 #include <unistd.h>
 3
 4 int main()
5 {
       FILE* fp;
 6
7
       while(1) {
 8
           fp = fopen("/proc/myled", "w");
           if(fp == NULL) {
9
               printf("Cannot open /proc/myled for write\n");
10
11
               return -1;
12
           }
           fputs("0x0F\n", fp);
13
14
           fclose(fp);
15
           sleep(1);
           fp = fopen("/proc/myled", "w");
16
17
           if(fp == NULL) {
18
               printf("Cannot open /proc/myled for write\n");
19
               return -1;
20
           }
21
           fputs("0x00\n", fp);
           fclose(fp);
22
23
           sleep(1);
24
       }
25
       return 0;
26 }
```

Figure 70. led_blink.c

2. Compose a Makefile and compile led_blink.c into led_blink.o, as shown in Figs. 71-73.



[kfranz@DIGILENT LINUX user app]\$ vim Makefile

Figure 71. Makefile.

```
1 cc = arm-xilinx-linux-gnueabi-gcc
2 CFLAGS = -g
3
4 all : led_blink
5
6 led blink : led_blink.o
7
      ${CC} ${CFLAGS} $^ -0 $@
8
9 clean :
     rm -rfv *.o
10
11
      rm -rfv led_blink
12
13 .PHONY : clean
```

Figure 72. Makefile.

```
[kfranz@DIGILENT_LINUX user_app]$ make
arm-xilinx-linux-gnueabi-gcc -g -c -o led_blink.o led_blink.c
arm-xilinx-linux-gnueabi-gcc -g -o led_blink led_blink.o
[kfranz@DIGILENT_LINUX user_app]$ ls
led_blink led_blink.c led_blink.o Makefile
[kfranz@DIGILENT_LINUX user_app]$
```

Figure 73. Compile led blink.

3. Insert the SD card into the computer, and copy the binary file led_blink onto the first partition of SD card, as shown in Fig. 74.

[kfranz@DIGILENT LINUX user app]\$ cp led blink /media/ZYBO_BOOT/

Figure 74. Move led_blink.

```
. . .
rcS Complete
zynq> mount /dev/mmcblk0p1 /mnt/
zynq> cd /mnt/
zynq> ls
BOOT.BIN
                                         led blink
                    devicetree.dtb
myled.ko
                    ramdisk8M.image.gz
                                          zImage
zynq> insmod myled.ko
[ 122.160000] myled probed at va 0x8000
zynq> ./led_blink
^C
zynq> mkdir -p /lib/modules/`uname -r`
zynq> cp myled.ko /lib/modules/`uname -r`
zynq> rmmod myled
```

Figure 75. RAMDISK.