

# Embedded Linux Hands-on Tutorial -- ZedBoard



Revision: Mar. 12, 2013

1300 NE Henley Court, Suite 3  
Pullman, WA 99163  
(509) 334 6306 Voice | (509) 334 6300 Fax

This *Embedded Linux Hands-on Tutorial – ZedBoard* will provide step-by-step instructions for customizing your hardware, compiling 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 software development, and apply them to ZedBoard.

In this tutorial, we will start from *ZedBoard Linux Hardware Design* (available on Digilent website, ZedBoard product page). The system architecture for *ZedBoard Linux Hardware Design* is shown in Figure 1.

In the *ZedBoard Linux Hardware Design*, we connect UART1 to USB-UART, SD0 to SD Card Slot, USB0 to USB-OTG port, Enet0 to Gig-bit Ethernet Port, and Quad SPI to on-board QSPI Flash. These cores are hard IPs inside Processing System (PS) and connect to on-board peripherals via Multiplexed I/O (MIO) pins. The use of PS GPIO is a little bit more complicated. We connect the lower bits of PS GPIO via MIOs to Button 8, 9, LED 9, and Pmod JE, while the higher bits of PS-GPIO are connected via Extended MIOs (EMIO) to 5 push buttons, 8 LEDs, 8 slide switches, Pmods JA to JD, and on-board OLED module. In the Programmable Logic (PL), we have HDMI Tx Controller, VDMA, SPDIF, and IIC IP cores to talk to ADV7511 HDMI Transmitter Chip and I2S and IIC IP Cores for ADAU1761 Audio Codec. More detail of the hardware design can be found in the documentation inside the ZedBoard Linux Hardware Design package.

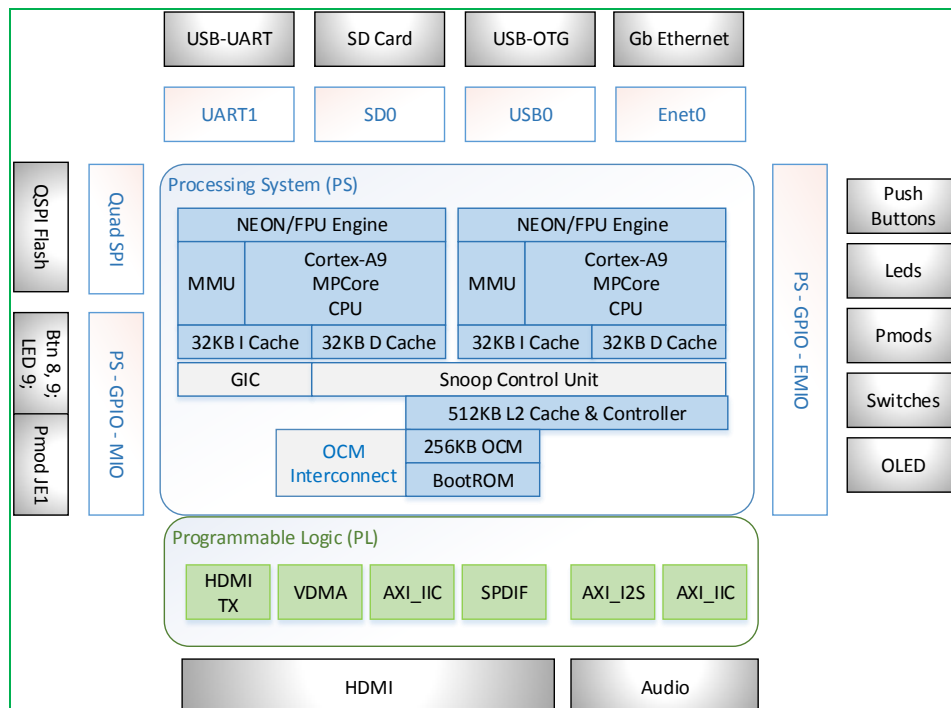
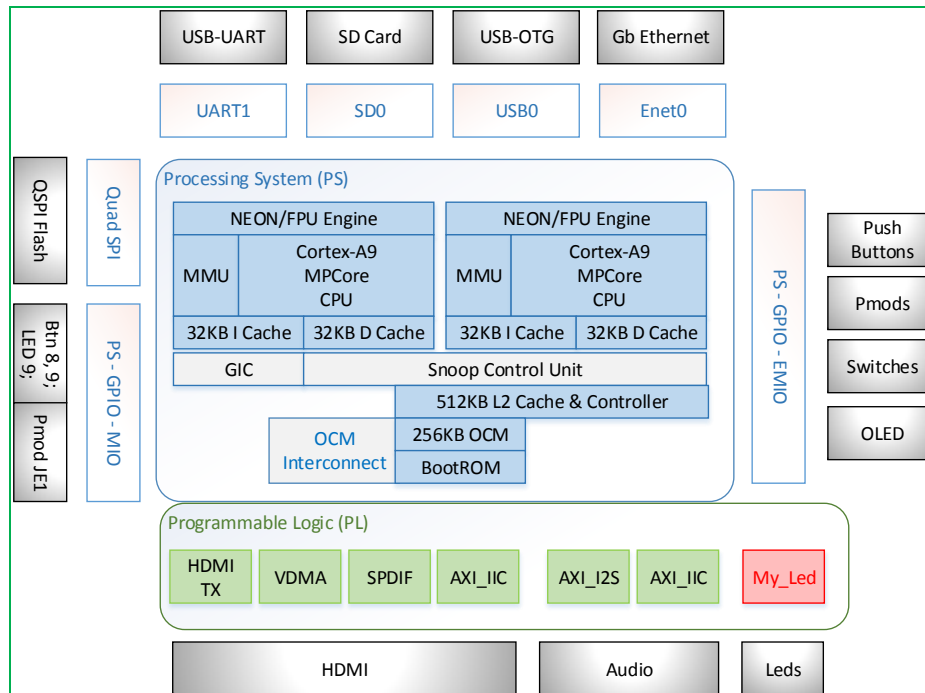


Figure 1. Reference Basic Hardware System Architecture for ZedBoard

In this tutorial, we are going to detach the LEDs from PS GPIO core, and implement our own *my\_led* core for it in PL, as shown in Figure 2. We will add our own LED controller into the device tree, write driver for it, and development user applications to control the status of LEDs.



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

Before going through this tutorial, we recommended that you read the *Getting Started with Embedded Linux -- ZedBoard* first. You can follow this tutorial with *Embedded Linux Development Guide* (available on 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 ISE DS 14.4** in Linux environment. All the screen shots and codes are done using **Xilinx ISE DS 14.4 x64 Edition** in **CentOS 6.2 x86\_64**.

So much for the background information for this tutorial, and we are about to get our hands dirty on real design!

## Section I: Hardware Customization

### Prerequisites

- **Xilinx ISE DS 14.4 (WebPack)**  
available at [Xilinx Website Download Page](#)
- *ZedBoard Linux Hardware Design*  
available at [Diligent Website ZedBoard Page](#)

### Instructions

- I-1** Download the *ZedBoard Linux Hardware Design* from the website and unzip it into our working directory. (Our working directory is named tutorial throughout this tutorial) See Example 1. For more information on the hardware design, please refer to *Project Guide* under `doc` folder.

#### Example 1.

```
[tinghui.wang@DIGILENT_LINUX Tutorial]$ unzip ~/Downloads/ZedBoard_Linux_Design.zip
Archive:  /home/tinghui.wang/Downloads/ZedBoard_Linux_Design.zip
...
  inflating: ZedBoard_Linux_Design/sw/hw_platform/ps7_init.html
  inflating: ZedBoard_Linux_Design/sw/hw_platform/ps7_init.tcl
  inflating: ZedBoard_Linux_Design/sw/hw_platform/system.xml
   creating: ZedBoard_Linux_Design/sw/zynq_fsbl/
   creating: ZedBoard_Linux_Design/sw/zynq_fsbl/src/
  inflating: ZedBoard_Linux_Design/sw/zynq_fsbl/src/main.c
[tinghui.wang@DIGILENT_LINUX Tutorial]$
```

- I-2** Source Xilinx ISE 14.4 settings and open the design with **Xilinx Platform Studio (XPS)**. See Example 2. You will see the **XPS** window pops up as shown in Figure 3.

**Note:** There are four setting files available in Xilinx tools for users to use: `settings64.sh` for users using 64-bit machine with bash; `settings32.sh` for users using 32-bit machine with bash; `settings32.csh` for users using 32-bit machine with bash; `settings64.csh` for users using 64-bit machine with C Shell.

#### Example 2.

```
[tinghui.wang@DIGILENT_LINUX Tutorial]$ source /opt/Xilinx/14.4/ISE_DS/settings64.sh
. /opt/Xilinx/14.4/ISE_DS/common/.settings64.sh /opt/Xilinx/14.4/ISE_DS/common
. /opt/Xilinx/14.4/ISE_DS/EDK/.settings64.sh /opt/Xilinx/14.4/ISE_DS/EDK
. /opt/Xilinx/14.4/ISE_DS/common/CodeSourcery/.settings64.sh
/opt/Xilinx/14.4/ISE_DS/common/CodeSourcery
. /opt/Xilinx/14.4/ISE_DS/PlanAhead/.settings64.sh /opt/Xilinx/14.4/ISE_DS/PlanAhead
. /opt/Xilinx/14.4/ISE_DS/../../Vivado/2012.4/.settings64.sh
/opt/Xilinx/14.4/ISE_DS/../../Vivado/2012.4
. /opt/Xilinx/14.4/ISE_DS/ISE/.settings64.sh /opt/Xilinx/14.4/ISE_DS/ISE
. /opt/Xilinx/14.4/ISE_DS/../../Vivado_HLS/2012.4/.settings64.sh
/opt/Xilinx/14.4/ISE_DS/../../Vivado_HLS/2012.4
[tinghui.wang@DIGILENT_LINUX Tutorial]$ xps ZedBoard_Linux_Design/hw/xps_proj/system.xmp &
[tinghui.wang@DIGILENT_LINUX Tutorial]$
```

I-3 We are going to detach LEDs from the GPIO core in PS system first. So, we need to click on the **I/O Peripherals** (red circle in Figure 3), and a **Zynq PS MIO Configuration** window will pop up as shown in Figure 4. We are going to change the width of GPIO on **EMIO** interface from **60** to **52** to remove 8 LED pins, as shown in the red circle in Figure 4. However, we also need to refresh the change on **External Port** in **Ports** tab. So, click **Ports** tab, expand `processing_system7_0`, expand **(IO\_IF) GPIO\_0**, disconnect it from **External Ports**, and connect it to **External Ports** again (as shown in Figure 5). We will leave the modification of external pin location configuration (`ucf` file) in later steps.

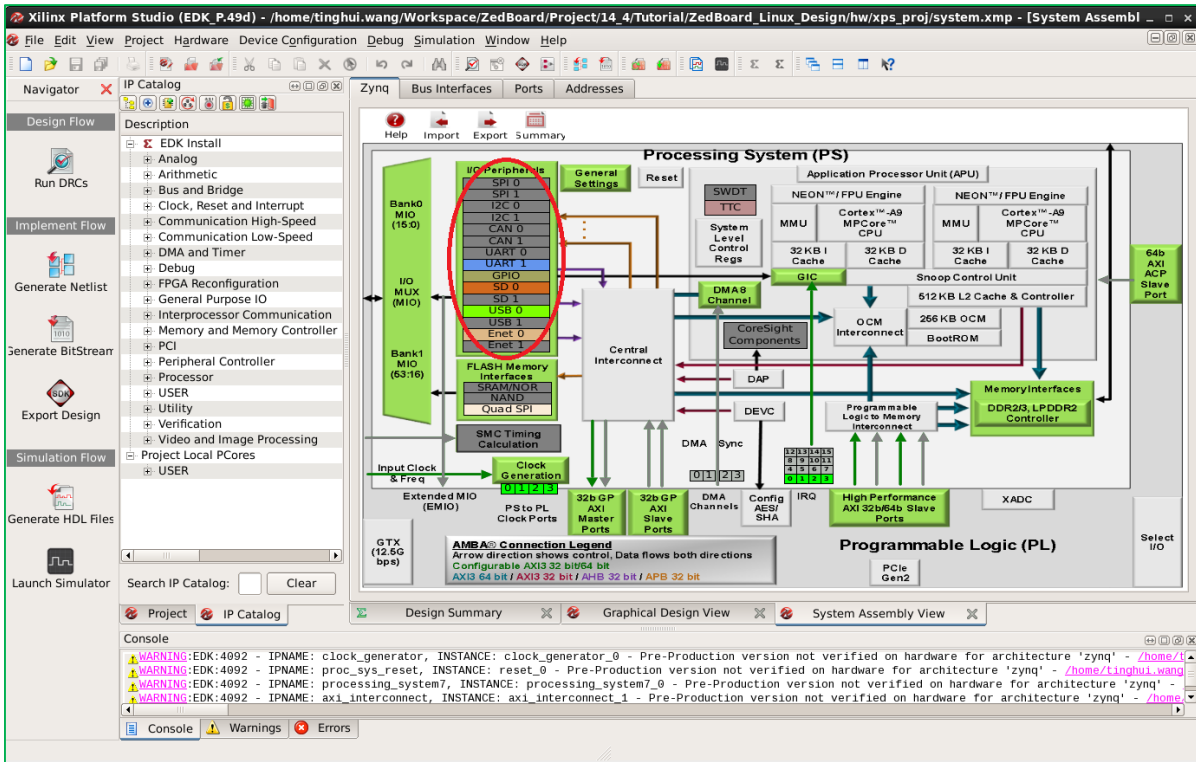


Figure 3. Xilinx Platform Studio GUI

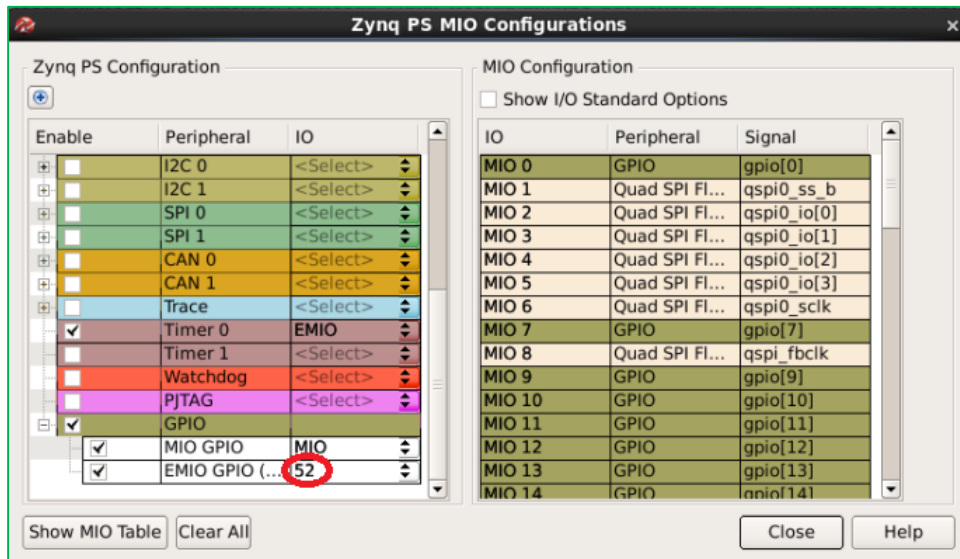


Figure 4. Zynq PS MIO Configurations

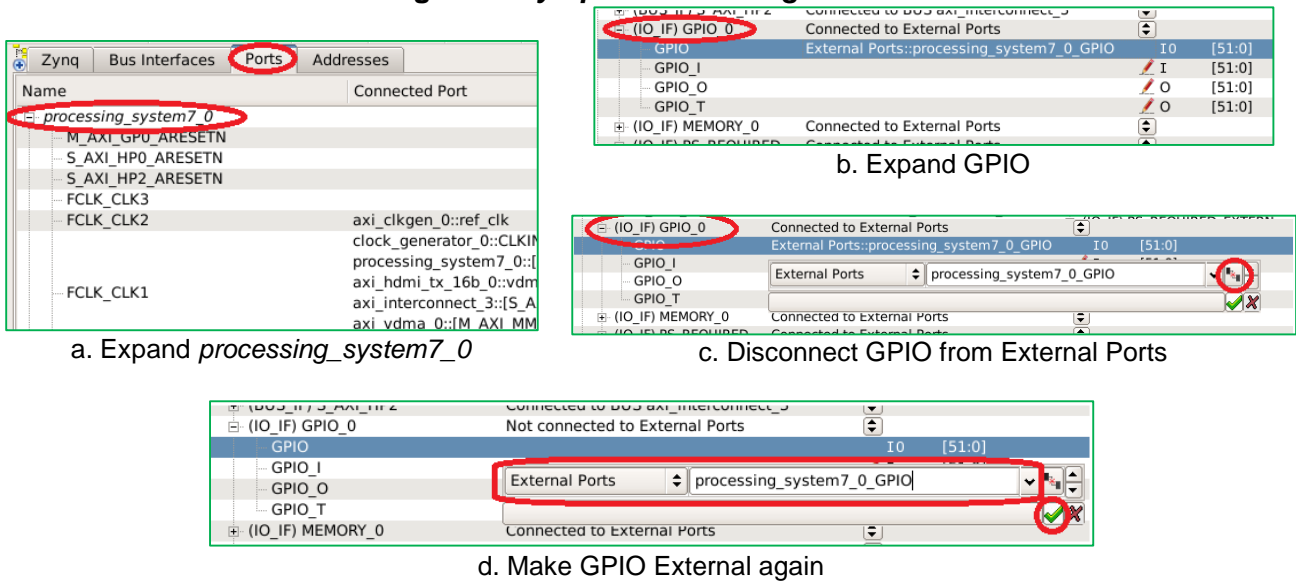


Figure 5. Reconnect GPIO to External ports

I-4 Now we can start implementing our myLED IP Core. Click **Hardware -> Create or Import Peripheral...** from the menu (as shown in Figure 6). The **Create and Import Peripheral Wizard** window will pop up (as shown in Figure 7). Click Next.

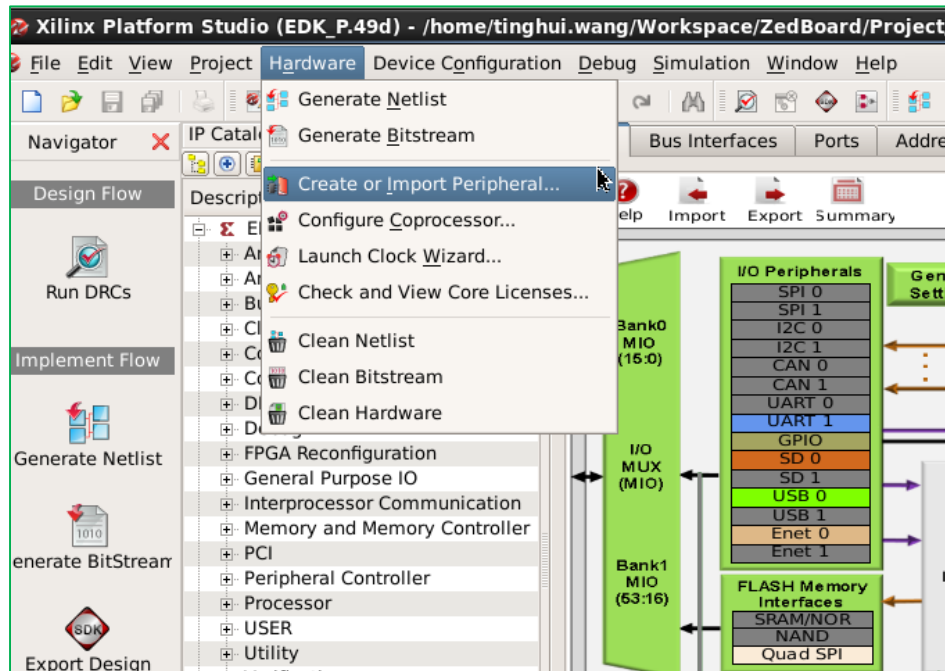


Figure 6. Start Create or Import Peripheral Wizard



Figure 7. The Create and Import Peripheral Wizard Window

I-5 In the Peripheral Flow, we select **Create Template for a New Peripheral** (as shown in Figure 8). Click Next.

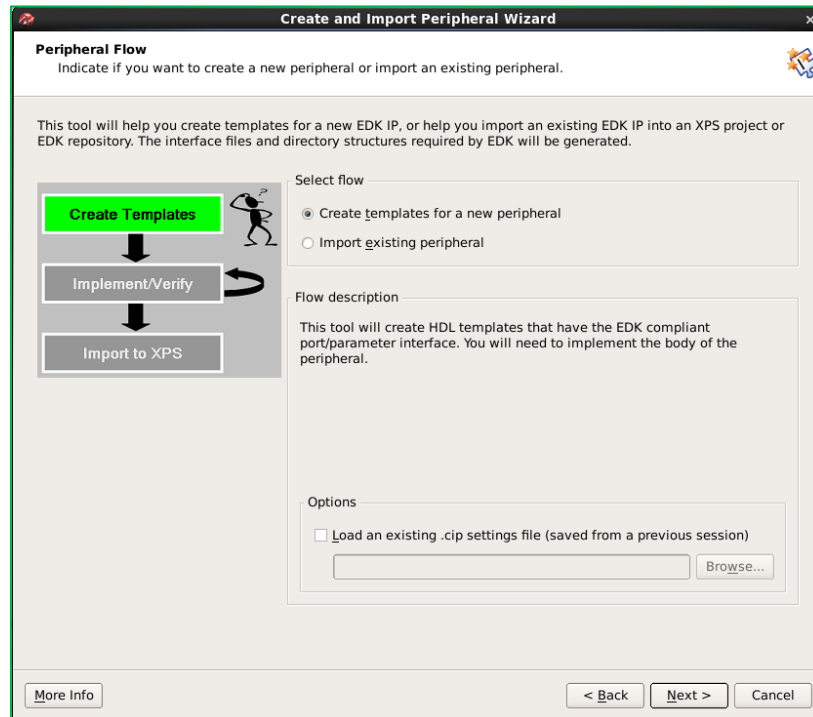


Figure 8. Select Peripheral Flow

I-6 We chose to store the IP core into this project folder (as shown in Figure 9). As a result, you will find `myled-1.00.a` folder under `pcores` folder. Click Next.

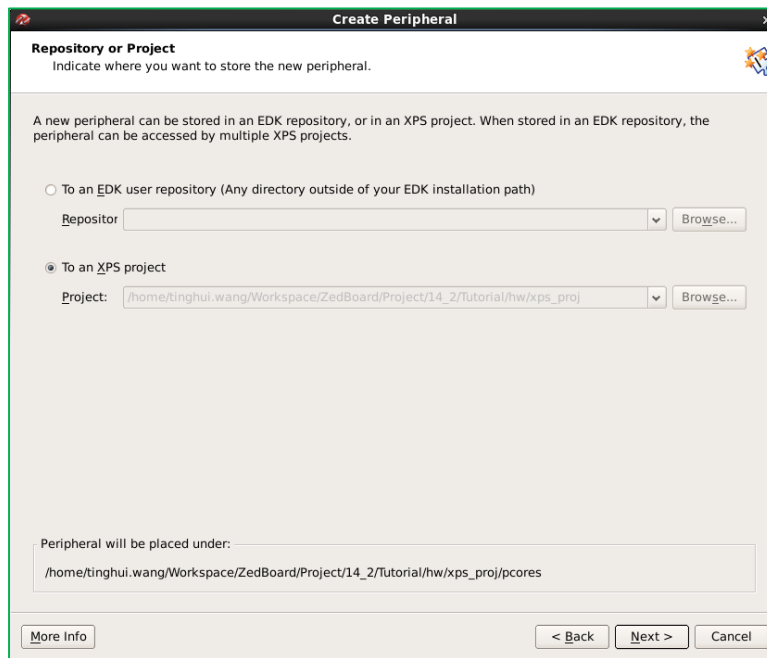


Figure 9. Store New IP Core in current XPS project

I-7 Fill in the name of the IP, and add your description of this IP core (as shown in Figure 10). Click Next.

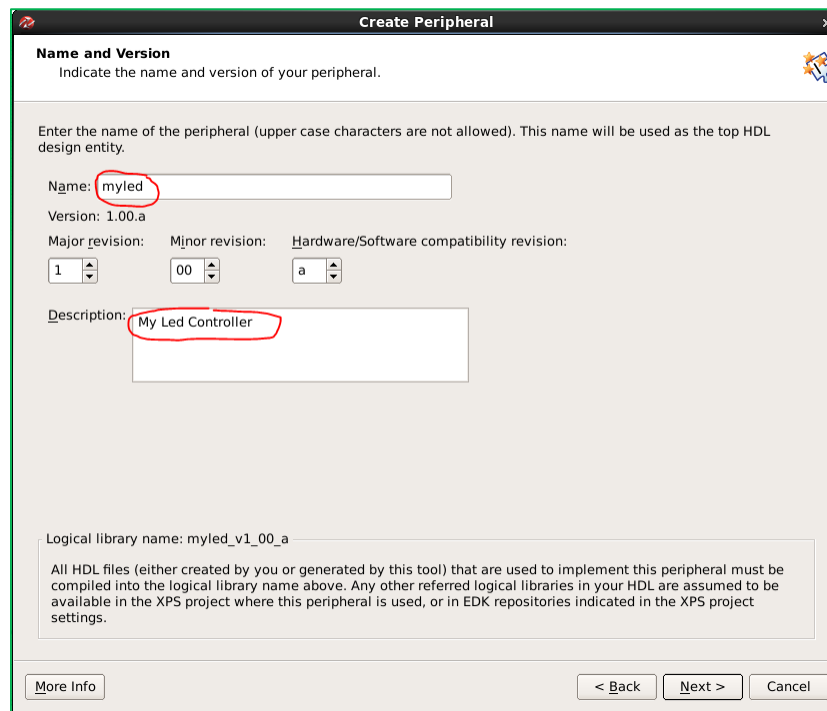


Figure 10. Fill in the name and description for the new IP Core



I-8 In Bus Interface, we will use AXI4-Lite. So, select **AXI4-Lite** and Click Next (As shown in Figure 11).

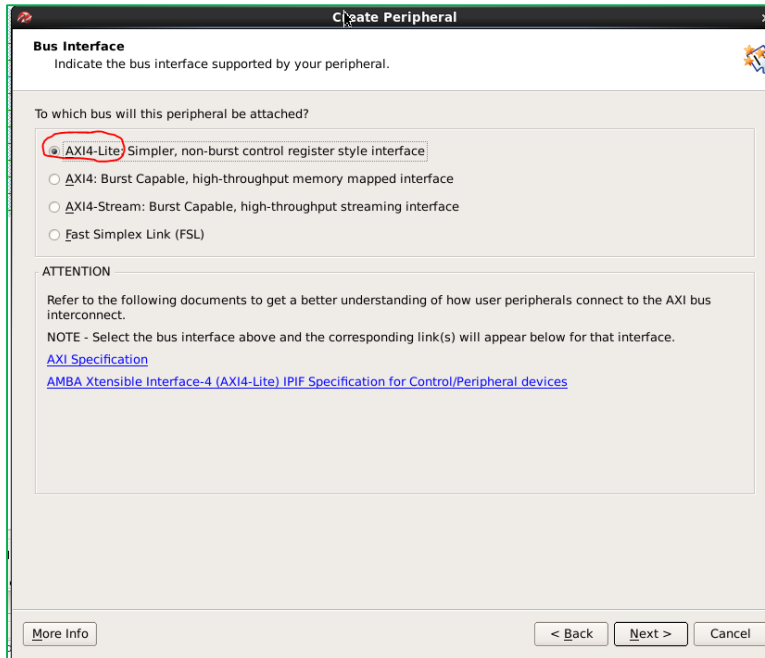


Figure 11. Select AXI4-Lite for Bus Interface

I-9 In IP Interface Service, the only thing we need is one software register. So, select **User Logic Software Register**, and deselect **User Logic master Support**, **Software Reset** and **Include Data Phase Timer** (as shown in Figure 12). Click Next.

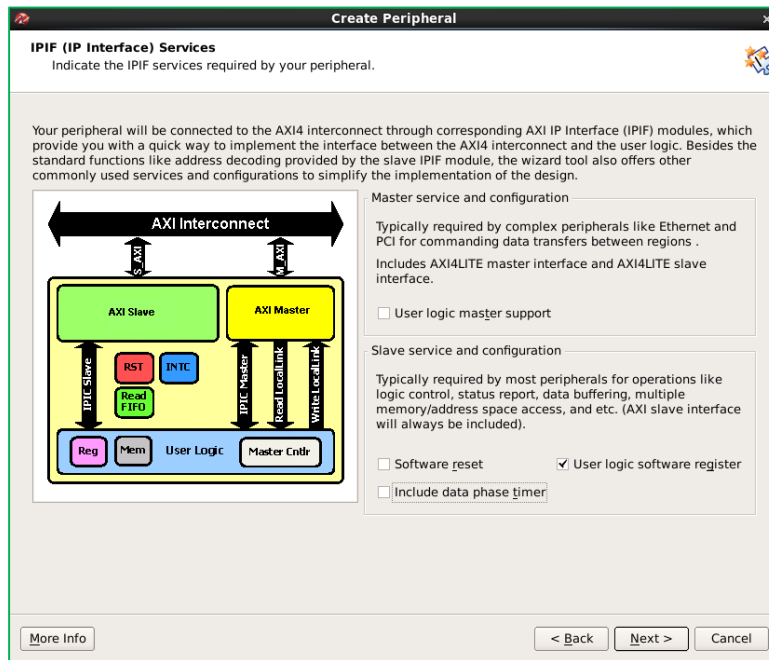


Figure 12. IP Interface Service Configuration



I-10 In our design, we only need 8 bits to control 8 LEDs. So, we only need **One** 32-bit register, and we will use the lower 8-bit in our design. In User S/W Register Configuration, type **1** in the box besides **Number of software accessible registers** (as shown in Figure 13). Click Next.

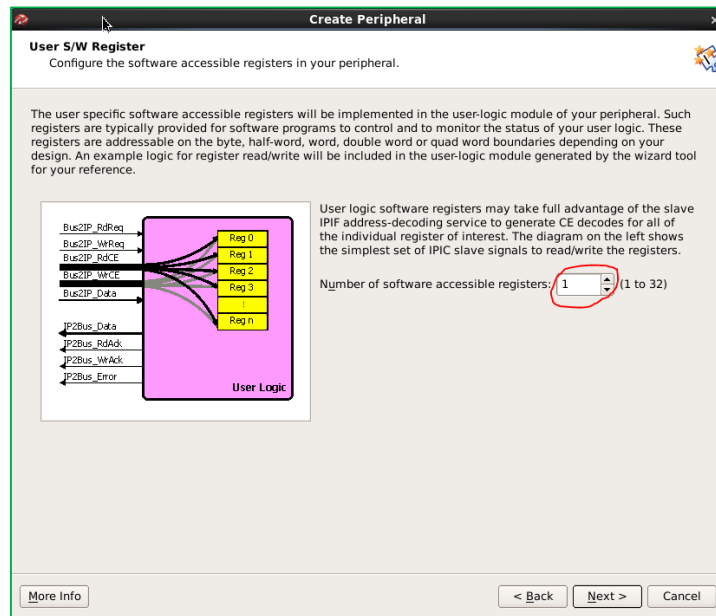


Figure 13. User S/W Register Configuration

I-11 Keep Default for **IP Interconnect Configuration** and **Peripheral Simulation Support**. If you want to use *Verilog* instead of *VHDL* (default), check the box beside **Generate stub 'user\_logic' template in Verilog instead of VHDL** in **Peripheral Implementation Support** (as shown in Figure 14). Click Next, and Click Finish.

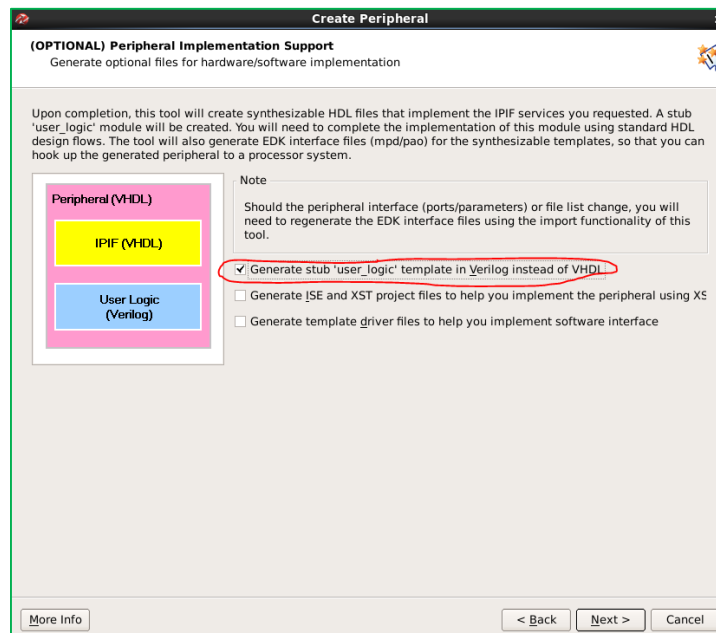


Figure 14. (OPTIONAL) Peripheral Implementation Support Configuration for Verilog Users

I-12 We are going to add our IP to our design. You will find the IP core in **IP Catalog** Panel under **Project Local PCores -> USER -> MYLED 1.00.a**. Right click on it and Click **Add IP** (as shown in Figure 15).

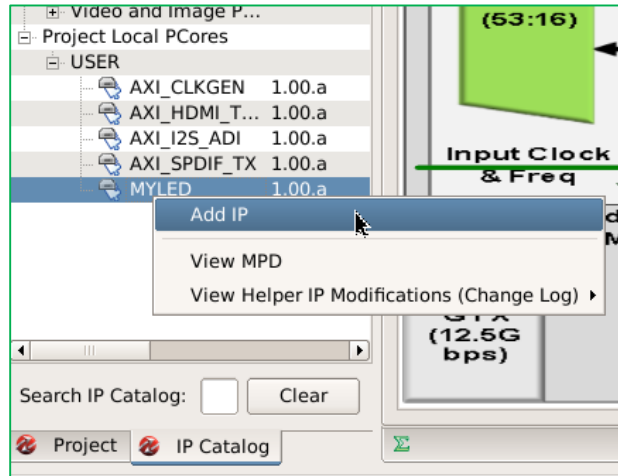


Figure 15. Add MYLED to your design

I-13 In the pop-up **XPS Core Configuration** window, keep everything as default (as shown in Figure 16), and Click OK.

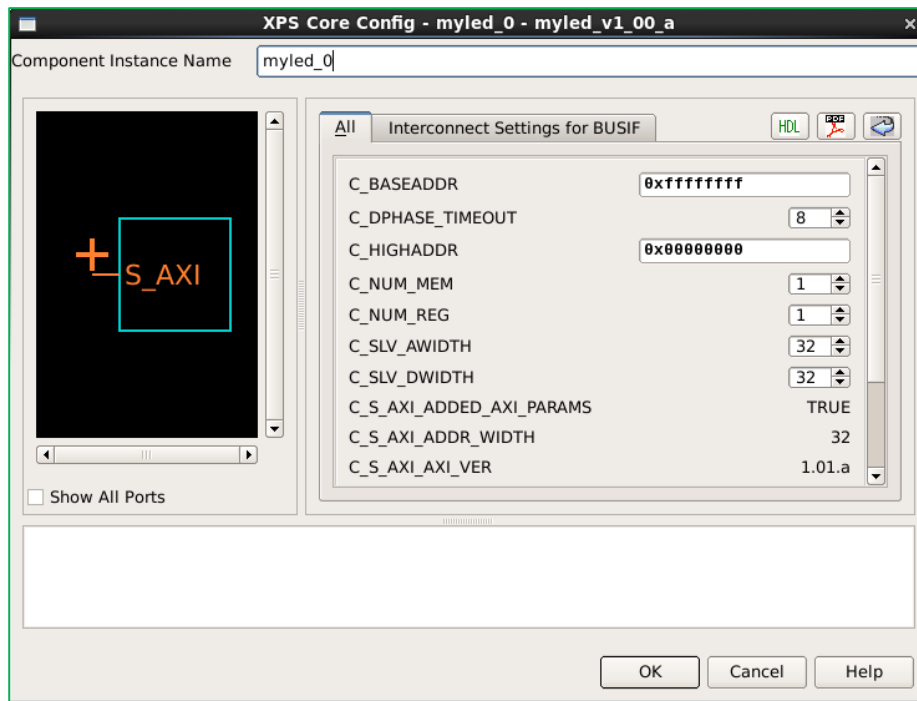


Figure 16. XPS Core Configuration for myled

I-14 The AXI4-Lite bus of myLED IP Core needs to be connected to processing system. So in **Instantiate and Connect IP** window, check **Select Processor Instance to Connect To**, and select **processing\_system7\_0** in the drop down box (as shown in Figure 17). Click OK.

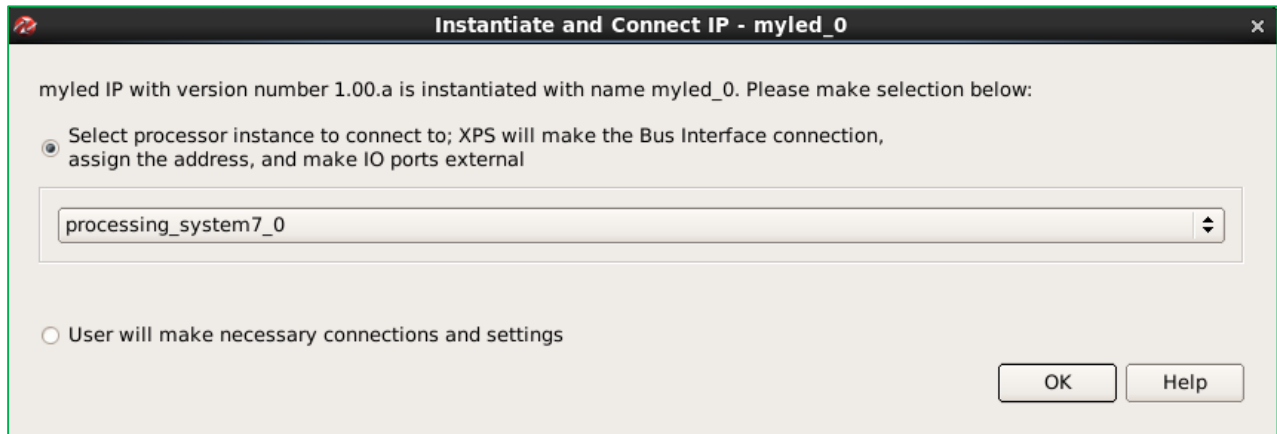


Figure 17. Instantiate and Connect MYLED IP

I-15 You will see an IP named myLED\_0 is added to your system with **S\_AXI** connected to **axi\_interconnect\_1** as shown in Figure 18.

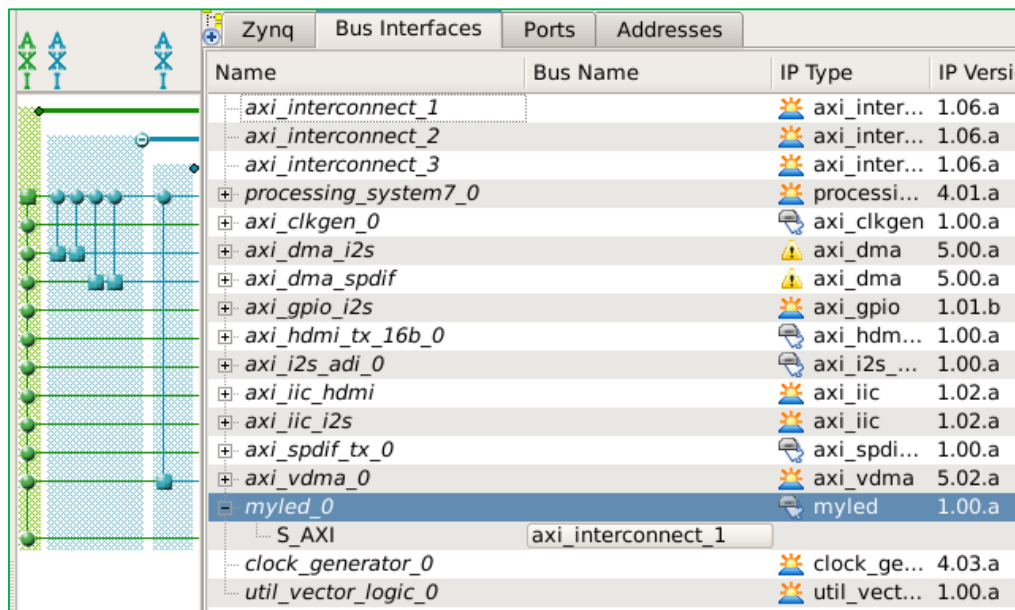
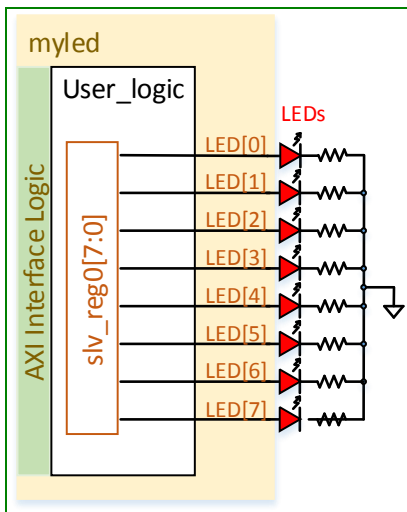
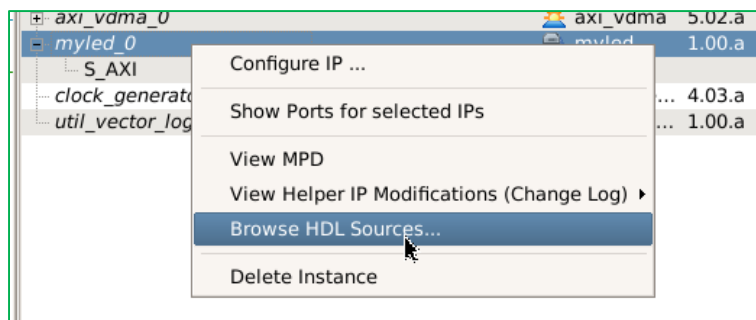


Figure 18. MYLED IP shows up in the design

**I-16** Now we are going to implement the circuit using HDL. The `myled` IP Core Template will generate two HDL files corresponding to two modules: `myled`, and `user_logic` (as shown in Figure 19). Module `myled` will always be in *VHDL*, and `user_logic` module will be in the language you select in step I-13. This tutorial will present you with solutions in both *VHDL* and *Verilog*.



**Figure 19. myled IP Block Diagram**



**Figure 20. Open HDL Source File of myled**

**I-17** (For *VHDL* Users) Right click on `myled_0` and click **Browse HDL Sources** (as shown in Figure 20). In the pop-up window locate source files `user_logic.vhd` under `xps_proj/pcores/myled_v1_00_a/hdl/vhdl/`. In declaration `user_logic` of entity, add a user port named `LED`, define it as output, `std_logic_vector(7 downto 0)`. In logic implementation, connect the lower 8 bit of `slv_reg0` to `LED`, as shown in Example 3 - *VHDL*.

**I-17** (For *Verilog* Users) Right click on `myled_0` and click **Browse HDL Sources** (as shown in Figure 20). In the pop-up window locate source files `user_logic.v` under `xps_proj/pcores/myled_v1_00_a/hdl/verilog/`. In declaration of `user_logic` module, add a user port named `LED`, define it as `output [7:0]`. In logic implementation, connect the lower 8 bit of `slv_reg0` to `LED`, as shown in Example 3 - *Verilog*.

**Example 3 – VHDL.**

```

84 entity user_logic is
85   generic
86   (
87     ...
88   );
89   port
90   (
91     -- ADD USER PORTS BELOW THIS LINE -----
92     --USER ports added here
93     LED : out std_logic_vector(7 downto 0);
94     -- ADD USER PORTS ABOVE THIS LINE -----
95     ...
96   );
97   ...
98   --USER signal declarations added here, as needed for user logic
99   LED <= slv_reg0(7 downto 0);

```

### Example 3 – Verilog.

```

84 entity user_logic is
85   generic
86   (
87     ...
96   );
97   port
98   (
99     -- ADD USER PORTS BELOW THIS LINE -----
100    --USER ports added here
101    LED                : out std_logic_vector(7 downto 0);
102    -- ADD USER PORTS ABOVE THIS LINE -----
103    ...
117  );
118  ...
147  --USER signal declarations added here, as needed for user logic
148  LED <= slv_reg0(7 downto 0);

```

**I-18** (Both *Verilog* & *VHDL*) Right click on **myled\_0** and click **Browse HDL Sources** (as shown in Figure 20). In the pop-up window locate source files `myled.vhd` under `xps_proj/pcores/myled_v1_00_a/hdl/vhdl/`. In declaration of `myled` entity, add a user port named `LED`, define it as `output, std_logic_vector(7 downto 0)`. Connect `LED` of `user_logic` to `LED` of `myled`, as shown in **Example 4**.

### Example 4.

```

114 entity myled is
115   generic
116   (
117     ...
136   );
137   port
138   (
139     -- ADD USER PORTS BELOW THIS LINE -----
140    --USER ports added here
141    LED                : out std_logic_vector(7 downto 0);
142    -- ADD USER PORTS ABOVE THIS LINE -----
143    ...
166  );
167  ...
292  USER_LOGIC_I : entity myled_v1_00_a.user_logic
293    generic map
294    (
295      ...
301    )
302    port map
303    (
304      -- MAP USER PORTS BELOW THIS LINE -----
305      --USER ports mapped here
306      LED                => LED,
307      -- MAP USER PORTS ABOVE THIS LINE -----
308      ...
319    );

```

I-19 Right click on **myled\_0** and click **View MPD**, as shown in Figure 21. In the MPD (Microprocessor Peripheral Definition), add port declaration for LED (as shown in **Example 5**).

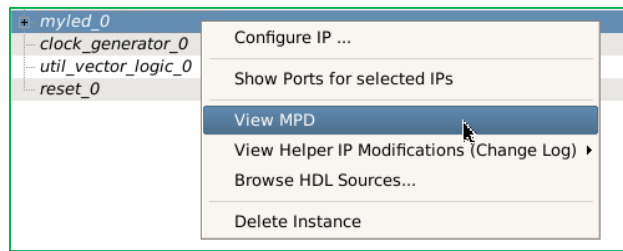


Figure 21. Open MPD of myled

**Example 5.**

```
...
38 ## Ports
39 PORT LED = "", DIR = O, VEC = [7:0]
40 PORT S_AXI_ACLK = "", DIR = I, SIGIS = CLK, BUS = S_AXI
...
```

I-20 Click **Project->Rescan User Repositories** to let port LED shown up on **Ports** tab (as shown in Figure 22). Switch to **Ports** tab, expand **myled\_0**, and connect **LED** to **External Ports** (as shown in Figure 23). Its default name is `myled_0_LED_pin`.

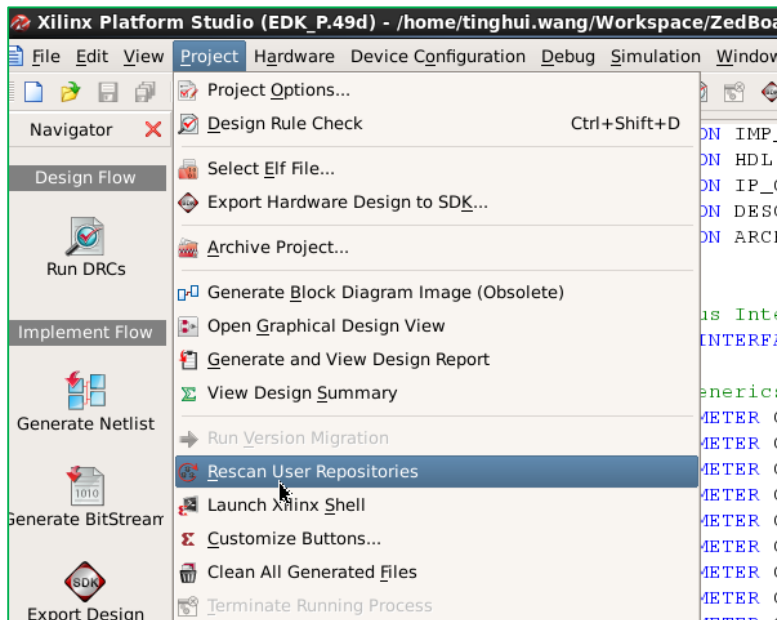


Figure 22. Rescan User Repositories

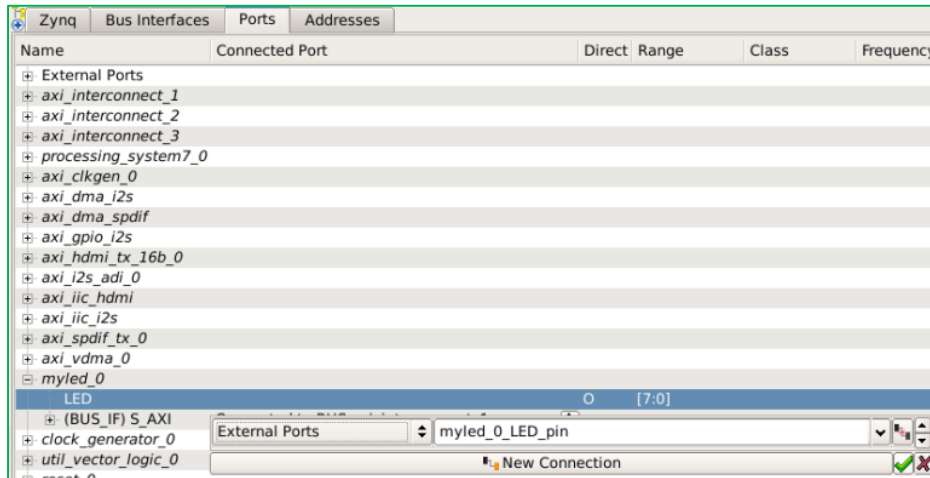


Figure 23. Connect LED to external ports

I-20 The final step is to specify the pin numbers for `myled_0_LED_pin` to connect our customized IP core to on-board LEDs physically. Double click UCF File: `data/system.ucf` as shown in Figure 24. Change the pin mapping according to Example 6.

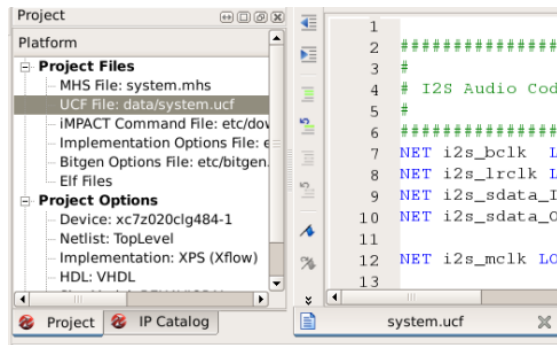


Figure 24. Open System UCF File

Example 6.

```

79
80 #####
81 # #
82 # On-board LED's #
83 # #
84 #####
85 net myled_0_LED_pin<0> LOC = T22 | IOSTANDARD = LVCMOS33; # LD0
86 net myled_0_LED_pin<1> LOC = T21 | IOSTANDARD = LVCMOS33; # LD1
87 net myled_0_LED_pin<2> LOC = U22 | IOSTANDARD = LVCMOS33; # LD2
88 net myled_0_LED_pin<3> LOC = U21 | IOSTANDARD = LVCMOS33; # LD3
89 net myled_0_LED_pin<4> LOC = V22 | IOSTANDARD = LVCMOS33; # LD4
90 net myled_0_LED_pin<5> LOC = W22 | IOSTANDARD = LVCMOS33; # LD5
91 net myled_0_LED_pin<6> LOC = U19 | IOSTANDARD = LVCMOS33; # LD6
92 net myled_0_LED_pin<7> LOC = U14 | IOSTANDARD = LVCMOS33; # LD7
93

```



### Example 6 (Cont.)

```
94 #####
95 # #
96 # On-board Slide Switches #
97 # #
98 #####
99 net processing_system7_0_GPIO<7> LOC = F22 | IOSTANDARD = LVCMOS33; # SW0
100 net processing_system7_0_GPIO<8> LOC = G22 | IOSTANDARD = LVCMOS33; # SW1
101 net processing_system7_0_GPIO<9> LOC = H22 | IOSTANDARD = LVCMOS33; # SW2
102 net processing_system7_0_GPIO<10> LOC = F21 | IOSTANDARD = LVCMOS33; # SW3
103 net processing_system7_0_GPIO<11> LOC = H19 | IOSTANDARD = LVCMOS33; # SW4
104 net processing_system7_0_GPIO<12> LOC = H18 | IOSTANDARD = LVCMOS33; # SW5
105 net processing_system7_0_GPIO<13> LOC = H17 | IOSTANDARD = LVCMOS33; # SW6
106 net processing_system7_0_GPIO<14> LOC = M15 | IOSTANDARD = LVCMOS33; # SW7
107
108 #####
109 # #
110 # On-board Left, Right, #
111 # Up, Down, and Select #
112 # Pushbuttons #
113 # #
114 #####
115 net processing_system7_0_GPIO<15> LOC = N15 | IOSTANDARD = LVCMOS33; # BTNL
116 net processing_system7_0_GPIO<16> LOC = R18 | IOSTANDARD = LVCMOS33; # BTNR
117 net processing_system7_0_GPIO<17> LOC = T18 | IOSTANDARD = LVCMOS33; # BTNU
118 net processing_system7_0_GPIO<18> LOC = R16 | IOSTANDARD = LVCMOS33; # BTND
119 net processing_system7_0_GPIO<19> LOC = P16 | IOSTANDARD = LVCMOS33; # BTNS
120
121 #####
122 # #
123 # Pmod JA #
124 # #
125 #####
126 net processing_system7_0_GPIO<20> LOC = Y11 | IOSTANDARD = LVCMOS33; # JA1
127 net processing_system7_0_GPIO<21> LOC = AA11 | IOSTANDARD = LVCMOS33; # JA2
128 net processing_system7_0_GPIO<22> LOC = Y10 | IOSTANDARD = LVCMOS33; # JA3
129 net processing_system7_0_GPIO<23> LOC = AA9 | IOSTANDARD = LVCMOS33; # JA4
130 net processing_system7_0_GPIO<24> LOC = AB11 | IOSTANDARD = LVCMOS33; # JA7
131 net processing_system7_0_GPIO<25> LOC = AB10 | IOSTANDARD = LVCMOS33; # JA8
132 net processing_system7_0_GPIO<26> LOC = AB9 | IOSTANDARD = LVCMOS33; # JA9
133 net processing_system7_0_GPIO<27> LOC = AA8 | IOSTANDARD = LVCMOS33; # JA10
134
135 #####
136 # #
137 # Pmod JB #
138 # #
139 #####
140 net processing_system7_0_GPIO<28> LOC = W12 | IOSTANDARD = LVCMOS33; # JB1
141 net processing_system7_0_GPIO<29> LOC = W11 | IOSTANDARD = LVCMOS33; # JB2
142 net processing_system7_0_GPIO<30> LOC = V10 | IOSTANDARD = LVCMOS33; # JB3
143 net processing_system7_0_GPIO<31> LOC = W8 | IOSTANDARD = LVCMOS33; # JB4
144 net processing_system7_0_GPIO<32> LOC = V12 | IOSTANDARD = LVCMOS33; # JB7
145 net processing_system7_0_GPIO<33> LOC = W10 | IOSTANDARD = LVCMOS33; # JB8
146 net processing_system7_0_GPIO<34> LOC = V9 | IOSTANDARD = LVCMOS33; # JB9
147 net processing_system7_0_GPIO<35> LOC = V8 | IOSTANDARD = LVCMOS33; # JB10
```

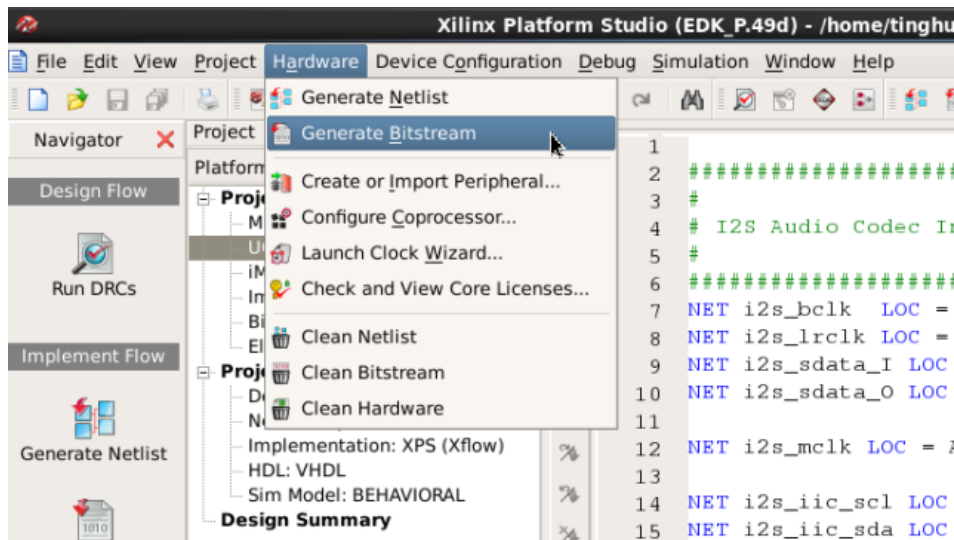
**Example 6 (Cont.)**

```

148
149 #####
150 #
151 # Pmod JC
152 #
153 #####
154 net processing_system7_0_GPIO<36> LOC = AB7 | IOSTANDARD = LVCMOS33; # JC1_P (JC1)
155 net processing_system7_0_GPIO<37> LOC = AB6 | IOSTANDARD = LVCMOS33; # JC1_N (JC2)
156 net processing_system7_0_GPIO<38> LOC = Y4 | IOSTANDARD = LVCMOS33; # JC2_P (JC3)
157 net processing_system7_0_GPIO<39> LOC = AA4 | IOSTANDARD = LVCMOS33; # JC2_N (JC4)
158 net processing_system7_0_GPIO<40> LOC = R6 | IOSTANDARD = LVCMOS33; # JC3_P (JC7)
159 net processing_system7_0_GPIO<41> LOC = T6 | IOSTANDARD = LVCMOS33; # JC3_N (JC8)
160 net processing_system7_0_GPIO<42> LOC = T4 | IOSTANDARD = LVCMOS33; # JC4_P (JC9)
161 net processing_system7_0_GPIO<43> LOC = U4 | IOSTANDARD = LVCMOS33; # JC4_N (JC10)
162
163 #####
164 #
165 # Pmod JD
166 #
167 #####
168 net processing_system7_0_GPIO<44> LOC = V7 | IOSTANDARD = LVCMOS33; # JD1_P (JD1)
169 net processing_system7_0_GPIO<45> LOC = W7 | IOSTANDARD = LVCMOS33; # JD1_N (JD2)
170 net processing_system7_0_GPIO<46> LOC = V5 | IOSTANDARD = LVCMOS33; # JD2_P (JD3)
171 net processing_system7_0_GPIO<47> LOC = V4 | IOSTANDARD = LVCMOS33; # JD2_N (JD4)
172 net processing_system7_0_GPIO<48> LOC = W6 | IOSTANDARD = LVCMOS33; # JD3_P (JD7)
173 net processing_system7_0_GPIO<49> LOC = W5 | IOSTANDARD = LVCMOS33; # JD3_N (JD8)
174 net processing_system7_0_GPIO<50> LOC = U6 | IOSTANDARD = LVCMOS33; # JD4_P (JD9)
175 net processing_system7_0_GPIO<51> LOC = U5 | IOSTANDARD = LVCMOS33; # JD4_N (JD10)

```

**I-21** Regenerate the bit stream for hardware design by clicking on **Hardware -> Generate Bitstream**, as shown in Figure 25.



**Figure 25. Generate Bitstream**

## Section II: Compile U-Boot (Optional)

### Prerequisites

- **Xilinx ISE DS 14.4 (WebPack)**  
available at Xilinx Website Download Page
- *ZedBoard Linux Hardware Design*  
available at Digilent Website ZedBoard Page
- *U-Boot Source Code*  
available at Digilent github repository: <https://github.com/Digilent/u-boot-digilent>

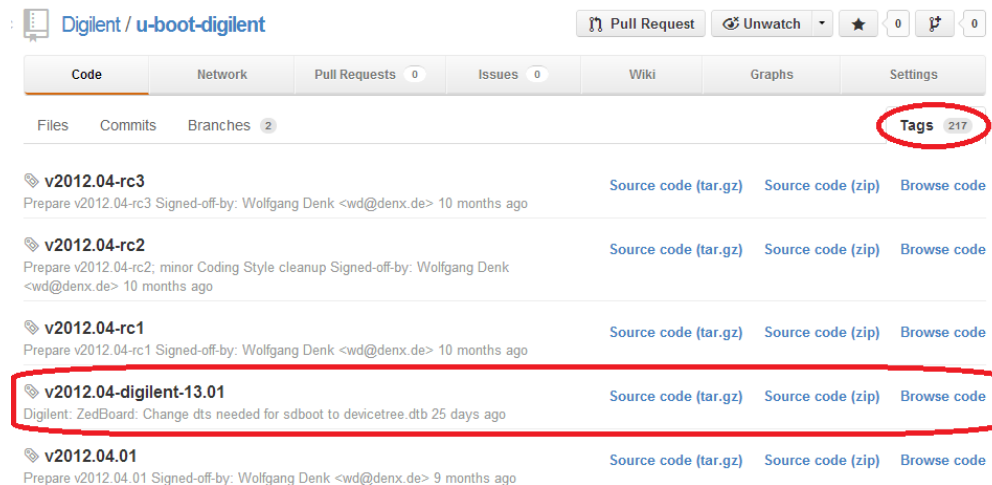
### Instructions

- II-1** Get the source code for U-Boot 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/Digilent/u-boot-digilent`. The whole Git Repository is around 55MB, as shown in Example 7.

#### Example 7.

```
[tinghui.wang@DIGILENT_LINUX Tutorial]$ git clone https://github.com/Digilent/u-boot-digilent
Initialized empty Git repository in
/home/tinghui.wang/Workspace/ZedBoard/Project/14_4/Tutorial/u-boot-digilent/.git/
remote: Counting objects: 190139, done.
remote: Compressing objects: 100% (36339/36339), done.
remote: Total 190139 (delta 153023), reused 188409 (delta 151293)
Receiving objects: 100% (190139/190139), 54.94 MiB | 158 KiB/s, done.
Resolving deltas: 100% (153023/153023), done.
[tinghui.wang@DIGILENT_LINUX Tutorial]$
```

- Download a compressed package:** If you only want to use u-boot once and do not want to track the updates, you can also download a compressed package from github.com website: <https://github.com/Digilent/u-boot-digilent>. Click **Tags** on the top right corner of the page, and the most recent tag is **v2012.04-digilent-13.01**.



The screenshot shows the GitHub repository page for `Digilent / u-boot-digilent`. The `Tags` tab is selected and circled in red. The list of tags is as follows:

| Tag                     | Source code (tar.gz)                 | Source code (zip)                 | Browse code                 |
|-------------------------|--------------------------------------|-----------------------------------|-----------------------------|
| v2012.04-rc3            | <a href="#">Source code (tar.gz)</a> | <a href="#">Source code (zip)</a> | <a href="#">Browse code</a> |
| v2012.04-rc2            | <a href="#">Source code (tar.gz)</a> | <a href="#">Source code (zip)</a> | <a href="#">Browse code</a> |
| v2012.04-rc1            | <a href="#">Source code (tar.gz)</a> | <a href="#">Source code (zip)</a> | <a href="#">Browse code</a> |
| v2012.04-digilent-13.01 | <a href="#">Source code (tar.gz)</a> | <a href="#">Source code (zip)</a> | <a href="#">Browse code</a> |
| v2012.04.01             | <a href="#">Source code (tar.gz)</a> | <a href="#">Source code (zip)</a> | <a href="#">Browse code</a> |

**Figure 26. Download Source code from github website**

If you downloaded the tar.gz, you can decompress it using command  
tar zxvf u-boot-digilent-2012.04-digilent-13.01.tar.gz

If you downloaded the zip file, you can decompress it using command  
unzip u-boot-digilent-2012.04-digilent-13.01.zip

**II-2** In this part, we are going to set a different MAC address and static IP address so that the board can work in local network. All the settings for ZedBoard are in the file located at include/configs/zynq\_zed.h. We will edit it using **vim** (as shown in Example 8), and change the MAC address to 00:0a:35:00:01:45, IP address to 192.168.10.250, and gateway IP to 192.168.10.1, (as shown in Example 9). Save the file using command **:wq**.

### Example 8.

```
[tinghui.wang@DIGILENT_LINUX Tutorial]$ cd u-boot-digilent/  
[tinghui.wang@DIGILENT_LINUX u-boot-digilent]$ vim include/configs/zynq_zed.h
```

### Example 9.

```
30 /* Default environment */  
31 #define CONFIG_IPADDR 192.168.10.250  
32 #define CONFIG_SERVERIP 192.168.10.1  
33  
34 #undef CONFIG_ZYNQ_XIL_LQSPI  
...  
40 #undef CONFIG_EXTRA_ENV_SETTINGS  
41 #define CONFIG_EXTRA_ENV_SETTINGS \\  
42     "ethaddr=00:0a:35:00:01:45\0" \\  
43     "kernel_size=0x140000\0" \\  
44     "ramdisk_size=0x200000\0" \
```

**II-3** To compile U-Boot, we need cross-compile tools which are provided by **Xilinx ISE 14.4**. Those tools have a prefix **arm-xilinx-linux-gnueabi-** to the standard names for **GCC** tool chain. In order to use the cross-platform compilers, please make sure Xilinx ISE 14.4 settings have been sourced. If not, please refer to step **I-2**. To configure and build U-Boot for ZedBoard, follow Example 10.

### Example 10.

```
[tinghui.wang@DIGILENT_LINUX u-boot-digilent]$ make CROSS_COMPILE=arm-xilinx-linux-gnueabi-  
zynq_zed_config  
Configuring for zynq_zed board...  
[tinghui.wang@DIGILENT_LINUX u-boot-digilent]$ make CROSS_COMPILE=arm-xilinx-linux-gnueabi-  
Generating include/autoconf.mk  
Generating include/autoconf.mk.dep  
arm-xilinx-linux-gnueabi-gcc -DDO_DEPS_ONLY \  
...  
make -C examples/api all  
make[1]: Entering directory `/home/tinghui.wang/Workspace/ZedBoard/Project/14_4/Tutorial/u-  
boot-digilent/examples/api'  
make[1]: Nothing to be done for `all'.  
make[1]: Leaving directory `/home/tinghui.wang/Workspace/ZedBoard/Project/14_4/Tutorial/u-  
boot-digilent/examples/api'  
[tinghui.wang@DIGILENT_LINUX u-boot-digilent]$
```

**II-4** After the compilation, the **ELF (Executable and Linkable File)** generated is named `u-boot`. We need to add `.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 put the `u-boot.elf` to `boot_image` folder and substitute the `u-boot.elf` that comes along with ZedBoard Embedded Linux Design Package, as shown in Example 11.

**Example 11.**

```
[tinghui.wang@DIGILENT_LINUX u-boot-digilent]$ cp u-boot ../ZedBoard_Linux_Design/boot_image/u-  
boot.elf  
[tinghui.wang@DIGILENT_LINUX u-boot-digilent]$
```

## Section III: Generate BOOT.BIN

### Prerequisites

- **Xilinx ISE DS 14.4 (WebPack)**  
available at [Xilinx Website Download Page](#)
- *ZedBoard Linux Hardware Design*  
available at [Digilent Website ZedBoard Page](#)
- *Finished Hardware Design from Section I & u-boot.elf from Section II*

### Instructions

**III-1** Export the hardware design (after step I-21) to Xilinx SDK by clicking on **Project -> Export Hardware Design to SDK...**, as shown in Figure 27. In the pop-up window, click **Export and Launch SDK**, as shown in Figure 28.

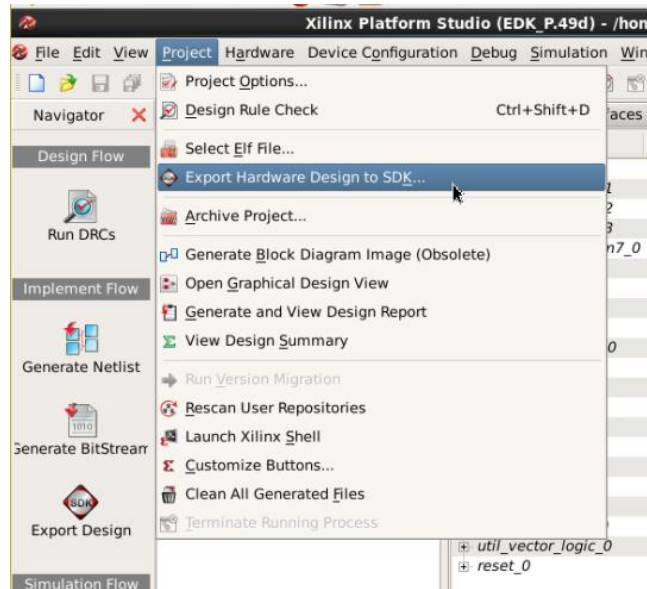


Figure 27. Export Hardware Design to SDK

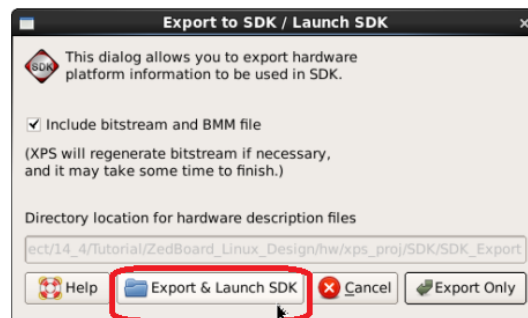


Figure 28. Export and Launch SDK

III-2 Set Workspace to `ZedBoard_Linux_Design/hw/xps_proj/SDK/SDK_Export` and click **OK**, as shown in Figure 29.

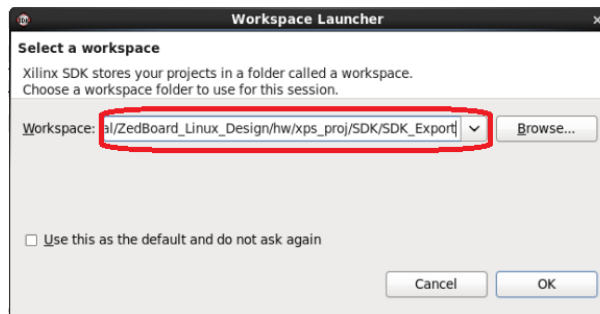


Figure 29. Set SDK Workspace Path

III-3 After SDK launches, the hardware platform project is already shown up in Project Explorer on the left of SDK main window, as shown in Figure 30. We need to create FSBL first. Click **File->New->Project...**, as shown in Figure 31.

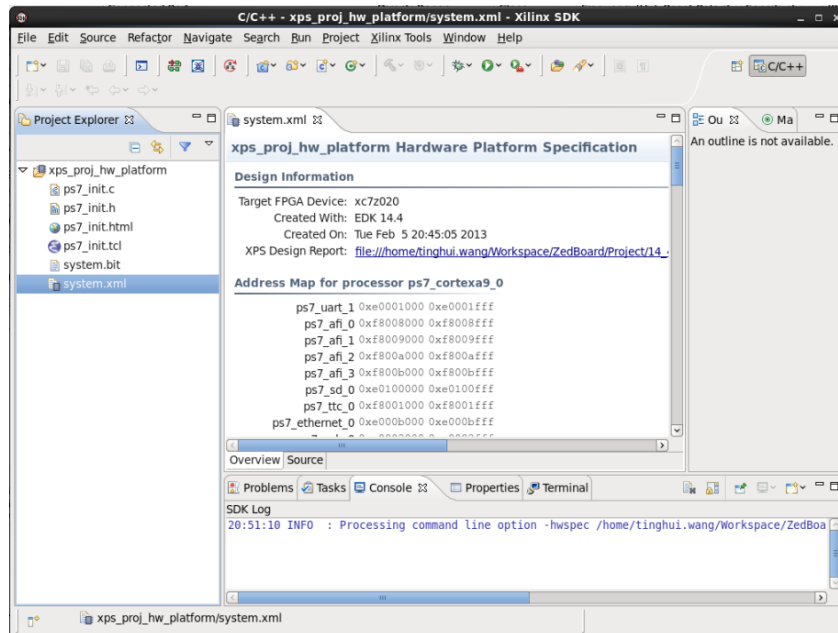


Figure 30. Xilinx SDK Window

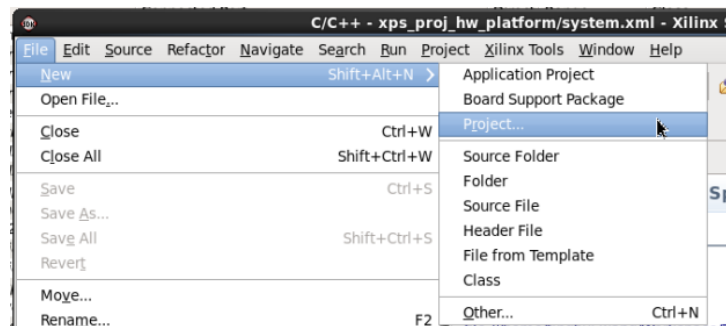


Figure 31. Create New Project in SDK



III-4 In New Project window, select **Xilinx->Application Project**, and then Click **Next** (Figure 32).

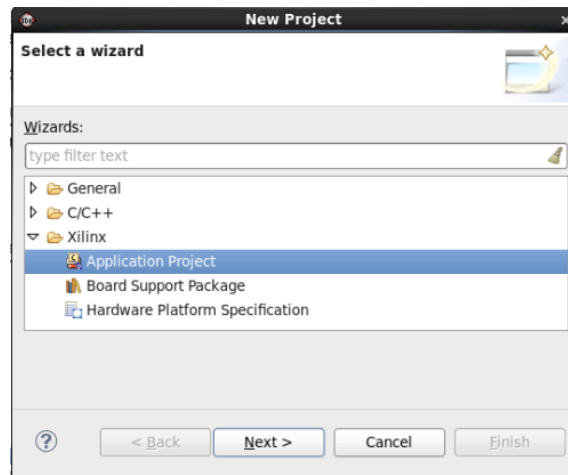


Figure 32. Select Application Project Wizard

III-5 We name the project FSBL. Select **xps\_proj\_hw\_platform** for **Target Hardware** because it is the hardware project we just exported. Select **standalone** for **OS Platform**. And Click **Next**. (as shown in Figure 33).

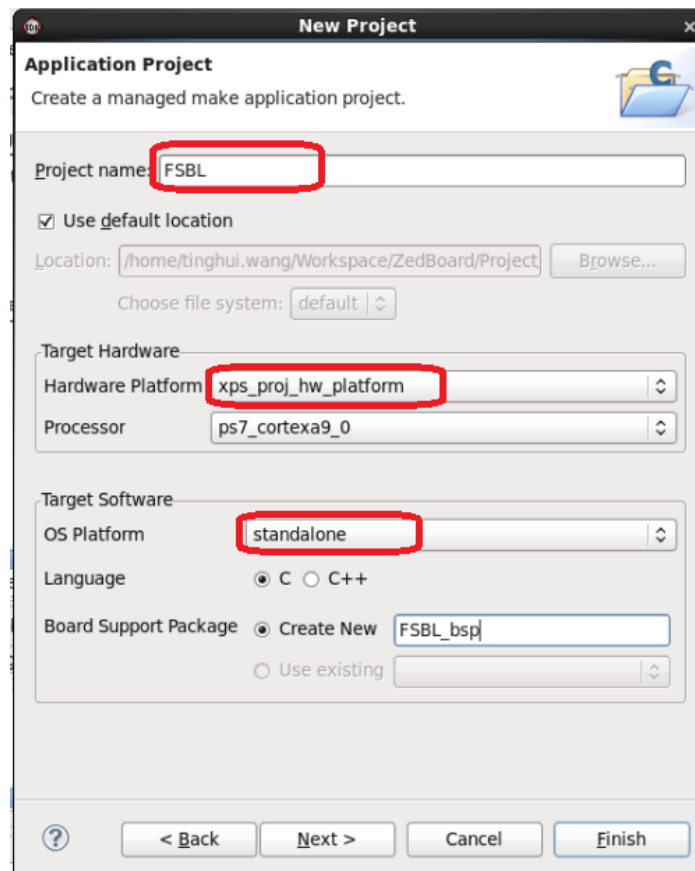


Figure 33. New Application Project

III-6 Select Zynq FSBL as template, and click Finish. (as shown in Figure 34).

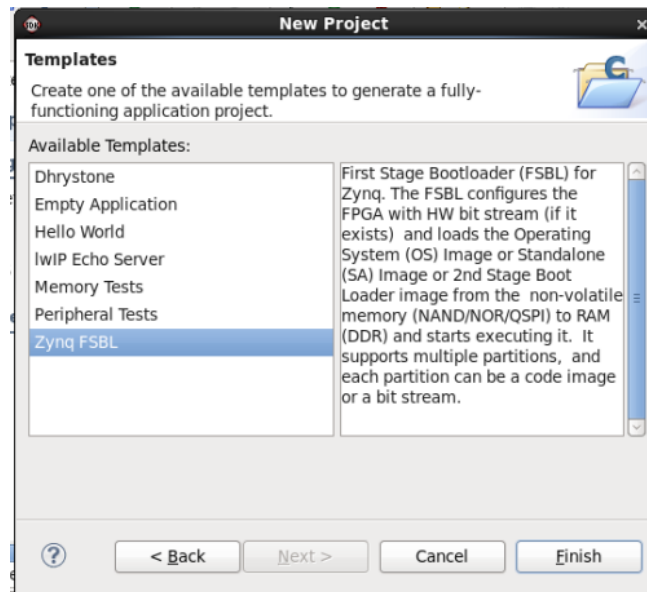


Figure 34. Select Zynq FSBL as Template

III-7 For ZedBoard, we need to reset USB PHY chip by toggling the USB-Reset pin in FSBL. You can swap the `main.c` file in FSBL project with the `main.c` under `sw/zynq_fsbl/src` folder in ZedBoard Linux Design Package. Or you can add the following code in Example 12 into `main.c` file of FSBL project.

**Example 12.**

```

470     FsblMeasurePerfTime (tCur, tEnd);
471 #endif
472
473     /* Reset the USB */
474     {
475         fsbl_printf(DEBUG_GENERAL, "Reset USB...\r\n");
476
477         /* Set data dir */
478         *(unsigned int *)0xe000a284 = 0x00000001;
479
480         /* Set OEN */
481         *(unsigned int *)0xe000a288 = 0x00000001;
482         Xil_DCacheFlush();
483         /* For REVB Set data value low for reset, then back high */
484 #ifndef ZED_REV_A
485         *(unsigned int *)0xe000a048 = 0x00000001;
486         Xil_DCacheFlush();
487         *(unsigned int *)0xe000a048 = 0x00000000;
488         Xil_DCacheFlush();
489 #else
490         *(unsigned int *)0xe000a048 = 0x00000000;
491         Xil_DCacheFlush();
492         *(unsigned int *)0xe000a048 = 0x00000001;
493         Xil_DCacheFlush();
494 #endif
495     }

```

III-8 After you have save the changes to `main.c` file, 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 `ZedBoard_Linux_Design/hw/xps_proj/SDK/SDK_Export/FSBL/Debug/FSBL.elf`

III-9 Now, we have all the files ready to create `BOOT.BIN`. Click **Xilinx Tools -> Create Zynq Boot Image**, as shown in Figure 35.

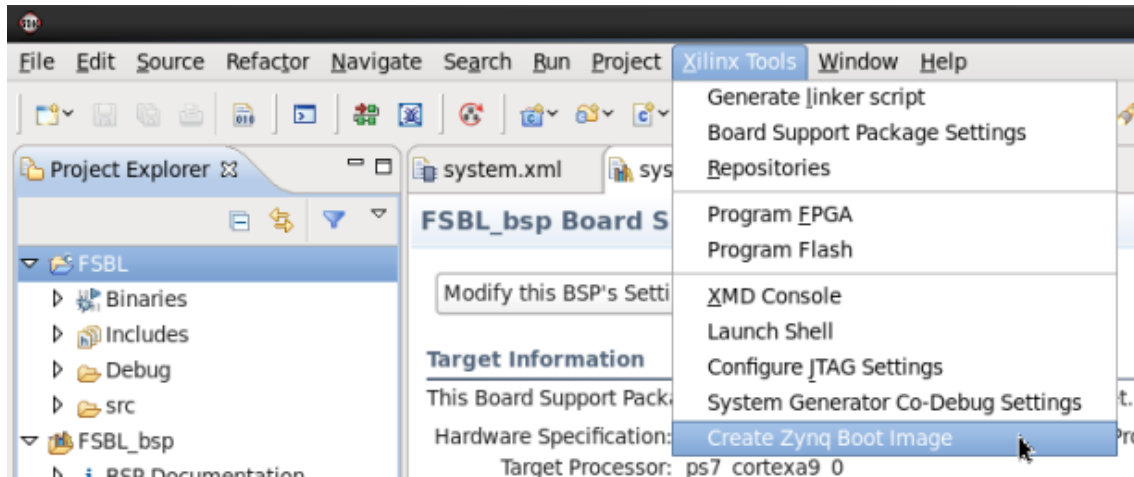


Figure 35. Create Zynq Boot Image

III-10 In the Create Zynq Boot Image window (as shown in Figure 36), Click **Browse** to set the path for **FSBL elf**. Click **Add** to add `u-boot.elf` that we compiled from Section II. In this tutorial, the `boot_image` folder under ZedBoard Linux Design is set as output folder for BIN file. Click **Create Image**.

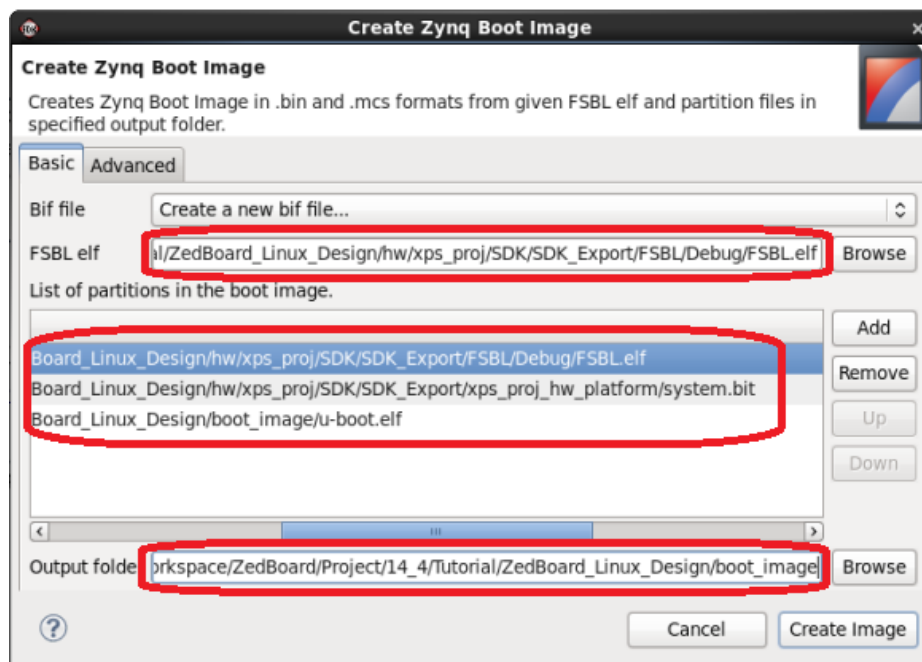


Figure 36. Create Zynq Boot Image Configuration

**III-11** The created BIN file was named u-boot.bin. We need to rename it to BOOT.BIN (all in capital letters) so that ZedBoard can find and load it after power up.

## Section IV: Compile Linux Kernel

### Prerequisites

➤ **Xilinx ISE DS 14.4 (WebPack)**

available at Xilinx Website Download Page

➤ *ZedBoard Linux Hardware Design*

available at Digilent Website ZedBoard Page

➤ *Linux Kernel Source Code*

available at Digilent github repository: <https://github.com/Digilent/linux-digilent>

### Instructions

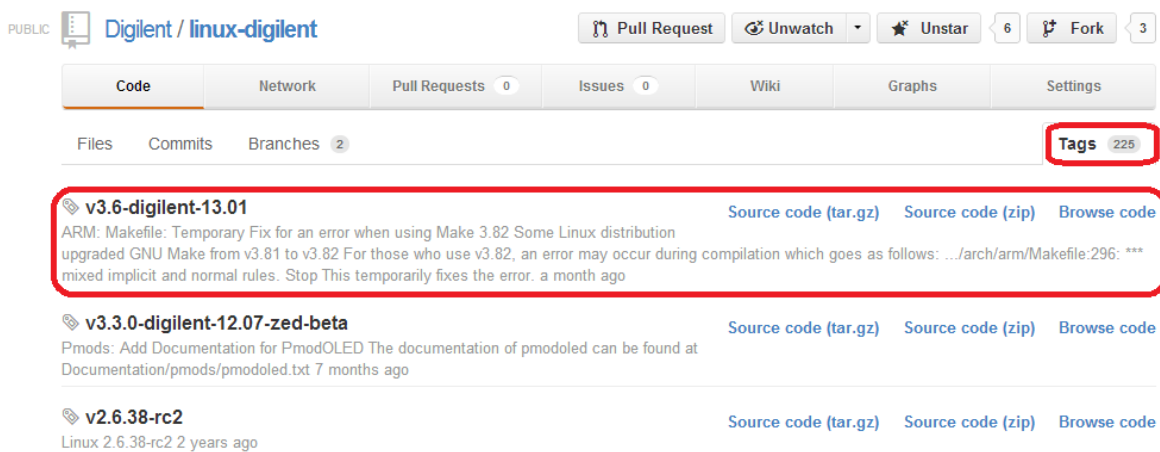
**IV-1** Get the Linux kernel source code from Digilent git repository. There are two ways to retrieve the source code:

- a. **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/Digilent/linux-digilent`. The whole Git Repository is around 550MB, as shown in Example 13.

#### Example 13.

```
[tinghui.wang@DIGILENT_LINUX Tutorial]$ git clone https://github.com/Digilent/linux-digilent
Initialized empty Git repository in
/home/tinghui.wang/Workspace/ZedBoard/Project/14_4/Tutorial/linux-digilent/.git/
remote: Counting objects: 2640848, done.
remote: Compressing objects: 100% (425292/425292), done.
remote: Total 2640848 (delta 2210020), reused 2620443 (delta 2189623)
Receiving objects: 100% (2640848/2640848), 565.52 MiB | 641 KiB/s, done.
Resolving deltas: 100% (2210020/2210020), done.
Checking out files: 100% (40001/40001), done.
[tinghui.wang@DIGILENT_LINUX Tutorial]$
```

- b. **Download a compressed package:** If you only want to use u-boot once and do not want to track the updates, you can also download a compressed package from github.com website: <https://github.com/Digilent/linux-digilent>. Click **Tags** on the top right corner of the page, and the most recent tag is **v3.6-digilent-13.01**.



PUBLIC Digilent / linux-digilent

Pull Request Unwatch Unstar 6 Fork 3

Code Network Pull Requests 0 Issues 0 Wiki Graphs Settings

Files Commits Branches 2 **Tags 225**

**v3.6-digilent-13.01** Source code (tar.gz) Source code (zip) Browse code

ARM: Makefile: Temporary Fix for an error when using Make 3.82 Some Linux distribution upgraded GNU Make from v3.81 to v3.82 For those who use v3.82, an error may occur during compilation which goes as follows: .../arch/arm/Makefile:296: \*\*\* mixed implicit and normal rules. Stop This temporarily fixes the error. a month ago

v3.3.0-digilent-12.07-zed-beta Source code (tar.gz) Source code (zip) Browse code

Pmmods: Add Documentation for PmodOLED The documentation of pmodoled can be found at Documentation/pmmods/pmodoled.txt 7 months ago

v2.6.38-rc2 Source code (tar.gz) Source code (zip) Browse code

Linux 2.6.38-rc2 2 years ago

Figure 37. Download Source code from github website

If you downloaded the tar.gz, you can decompress it using command  
tar zxvf linux-digilent-v3.6-digilent-13.01.tar.gz

If you downloaded the zip file, you can decompress it using command  
unzip linux-digilent-v3.6-digilent-13.01.zip

**IV-2** We will start to configure the kernel with the default configuration for ZedBoard. The configuration locates at arch/arm/configs/digilent\_zed\_defconfig. To use the default configure, you can follow Example 14.

**Example 14.**

```
[tinghui.wang@DIGILENT_LINUX Tutorial]$ cd linux-digilent/  
[tinghui.wang@DIGILENT_LINUX linux-digilent]$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-  
gnull- digilent_zed_defconfig  
HOSTCC scripts/basic/fixdep  
HOSTCC scripts/kconfig/conf.o  
SHIPPED scripts/kconfig/zconf.tab.c  
SHIPPED scripts/kconfig/zconf.lex.c  
SHIPPED scripts/kconfig/zconf.hash.c  
HOSTCC scripts/kconfig/zconf.tab.o  
HOSTLD scripts/kconfig/conf  
#  
# configuration written to .config  
#  
[tinghui.wang@DIGILENT_LINUX linux-digilent]$
```

**IV-3** We will change the configuration for PmodOLED driver. Change it from built-in driver to a loadable kernel module. To do this, we will start configuration menu for Linux kernel following Example 15. And configuration screen will show up in your terminal as Figure 37.

**Example 15.**

```
[tinghui.wang@DIGILENT_LINUX linux-digilent]$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-  
gnull- menuconfig
```

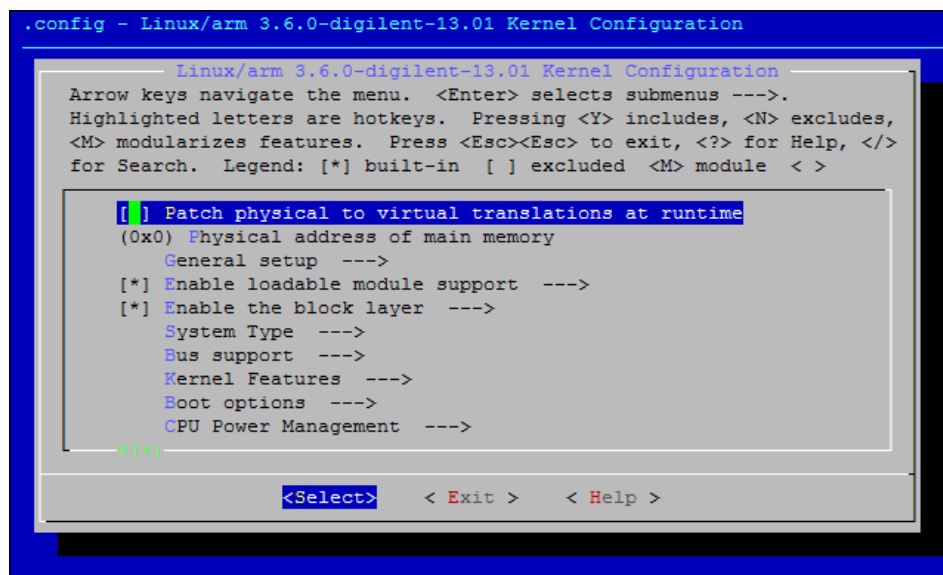


Figure 37. Linux Kernel Configuration Menu

IV-4 Browse to Device Driver-> PMOD Support -> PmodOLED1, press M to modularize the feature, as shown in Figure 38, 39, 40.

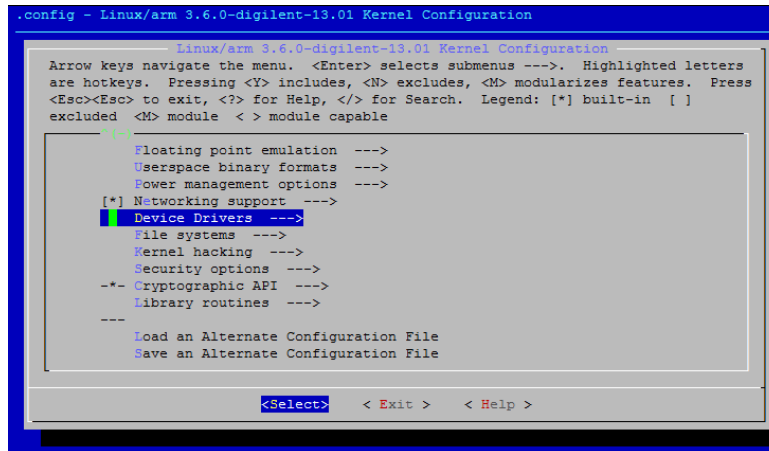


Figure 38. Browse to Device Drivers

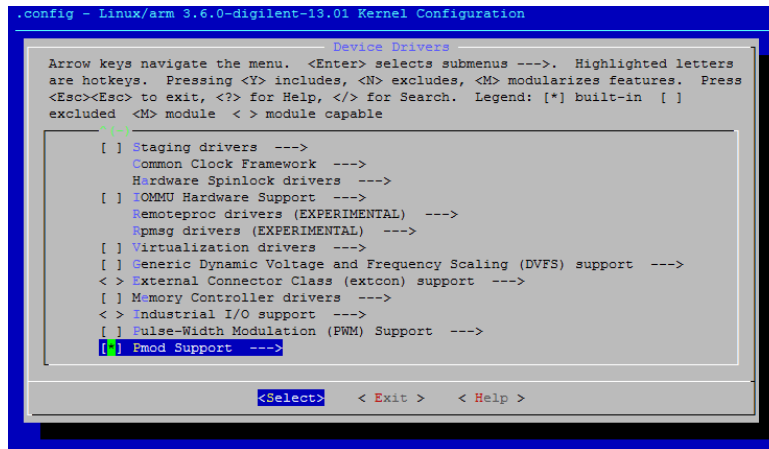


Figure 39. Browse to Pmod Support

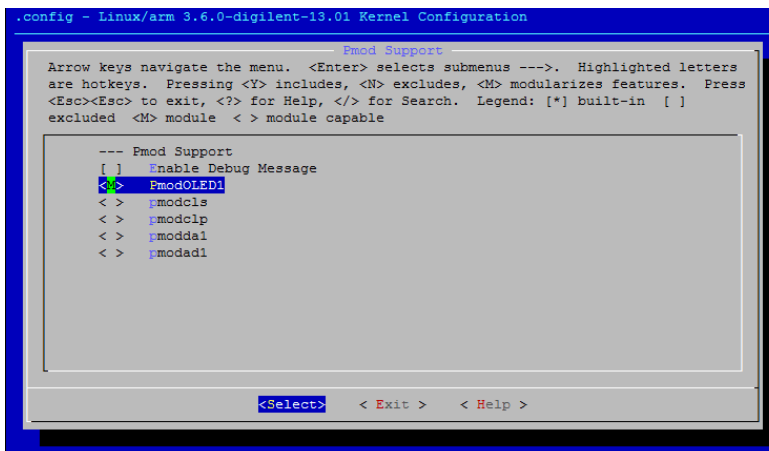
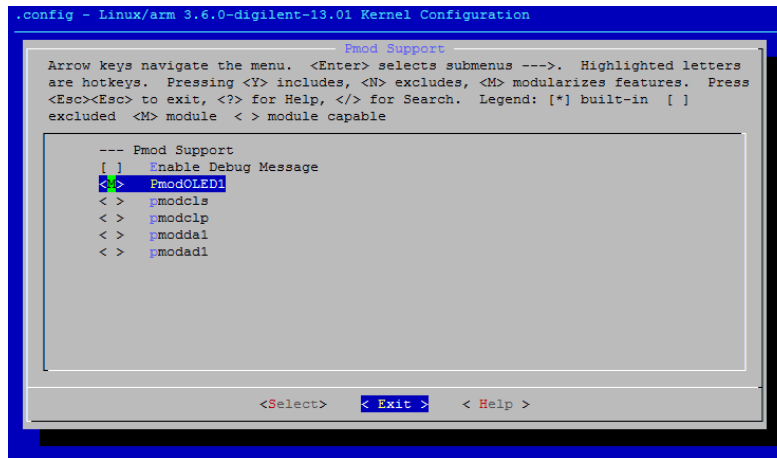


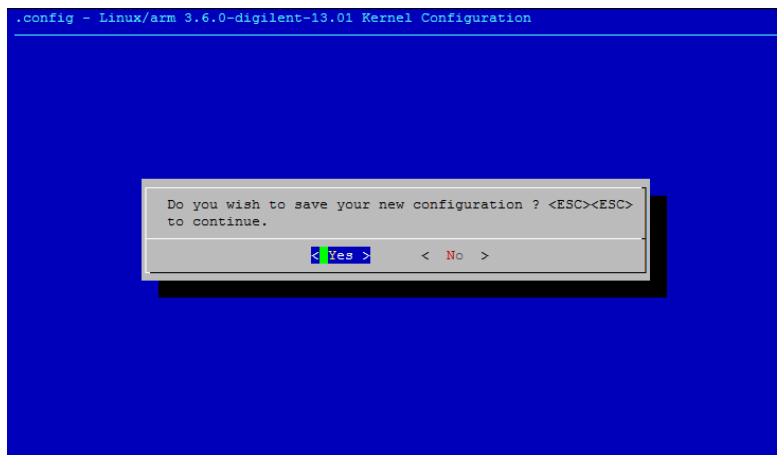
Figure 40. Modularize PmodOLED1 Feature



**IV-5** Move cursor to Exit to exit Linux Kernel Configuration menu, as shown in Figure 41. Chose YES to save the new configuration, as shown in Figure 42.



**Figure 41. Exit Linux Kernel Configuration Menu**



**Figure 42. Chose Yes to save the new Linux Kernel Configuration**

**IV-6** Follow Example 16 to compile Linux Kernel.

**Example 16.**

```
[tinghui.wang@DIGILENT_LINUX linux-digilent]$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-  
scripts/kconfig/conf --silentoldconfig Kconfig  
WRAP arch/arm/include/generated/asm/auxvec.h  
WRAP arch/arm/include/generated/asm/bitsperlong.h  
WRAP arch/arm/include/generated/asm/cputime.h  
WRAP arch/arm/include/generated/asm/emergency-restart.h  
...  
GEN .version  
CHK include/generated/compile.h  
UPD include/generated/compile.h  
CC init/version.o  
LD init/built-in.o  
KSYM .tmp_kallsyms1.o  
KSYM .tmp_kallsyms2.o  
LD vmlinux
```

### Example 16 (Cont.).

```
SYSMAP System.map
OBJCOPY arch/arm/boot/Image
Kernel: arch/arm/boot/Image is ready
AS arch/arm/boot/compressed/head.o
GZIP arch/arm/boot/compressed/piggy.gzip
AS arch/arm/boot/compressed/piggy.gzip.o
CC arch/arm/boot/compressed/misc.o
CC arch/arm/boot/compressed/decompress.o
CC arch/arm/boot/compressed/string.o
SHIPPED arch/arm/boot/compressed/liblfuncs.S
AS arch/arm/boot/compressed/liblfuncs.o
SHIPPED arch/arm/boot/compressed/ashldi3.S
AS arch/arm/boot/compressed/ashldi3.o
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
...
CC drivers/pmods/pmodoled-gpio.mod.o
LD [M] drivers/pmods/pmodoled-gpio.ko
...
[tinghui.wang@DIGILENT_LINUX linux-digilent]$
```

**IV-7** After the compilation, the kernel image is located at arch/arm/boot/zImage.

## Section V (Optional): Test Kernel Image with Pre-built File System

### Prerequisites

- **Xilinx ISE DS 14.4 (WebPack)**  
available at Xilinx Website Download Page
- *Linux Kernel Source Code*  
available at Digilent github repository: <https://github.com/Digilent/linux-digilent>
- **Pre-built File System Image**  
ramdisk Image is available in ZedBoard Linux Reference Design  
linaro File System can be obtained from <http://release.linaro.org>
- *BOOT.BIN from Section III, zImage from Section IV.*

### Instructions

**V-1** To boot Linux Operating System on ZedBoard, you need BOOT.BIN, Linux kernel image (zImage), device tree blob (DTB file), and file system. BOOT.BIN has been created in Section III, zImage has been compiled in Section IV. We will now compile DTB file. The default device tree source file is located at arch/arm/boot/dts/digilent-zed.dts.

**RAMDISK:** modify the device tree source file according to Example 17.

**LINARO FS:** use the default device tree source as it is.

#### Example 17.

```
48     chosen {
49         /* bootargs = "console=ttyPS0,115200 root=/dev/mmcbk0p2 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     };
```

**V-2** Generate DTB file, as shown in Example 18.

#### Example 18.

```
[tinghui.wang@DIGILENT_LINUX linux-digilent]$ ./scripts/dtc/dtc -I dts -O dtb
-o ../devicetree.dtb arch/arm/boot/dts/digilent-zed.dts
DTC: dts->dtb on file "arch/arm/boot/dts/digilent-zed.dts"
[tinghui.wang@DIGILENT_LINUX linux-digilent]$
```

**V-3 (RAMDISK)** Copy BOOT.BIN, devicetree.dtb, zImage and ramdisk8M.tar.gz to first partition of SD card, as shown in Example 19 – RamDisk.

**(LINARO FS)** Follow **Formatting the SD Card** part in *Getting Started with Embedded Linux – ZedBoard* to create Linaro File System on the second partition of SD Card. Copy BOOT.BIN, devicetree.dtb, zImage to the first partition of SD card, as shown in Example 19 – Linaro.

NOTE: In Example, the first partition of SD card is mounted to /media/ZED\_BOOT

### Example 19 – RamDisk.

```
[tinghui.wang@DIGILENT_LINUX Tutorial]$ ls
devicetree.dtb  linux-digilent  u-boot-digilent  ZedBoard_Linux_Design
[tinghui.wang@DIGILENT_LINUX Tutorial]$ cp ZedBoard_Linux_Design/boot_image/BOOT.BIN
/media/ZED_BOOT/
[tinghui.wang@DIGILENT_LINUX Tutorial]$ cp ZedBoard_Linux_Design/sd_image/ramdisk8M.image.gz
/BOOT.BIN /media/ZED_BOOT/
[tinghui.wang@DIGILENT_LINUX Tutorial]$ cp ./devicetree.dtb /media/ZED_BOOT/
[tinghui.wang@DIGILENT_LINUX Tutorial]$ cp linux-digilent/arch/arm/boot/zImage
/media/ZED_BOOT/
[tinghui.wang@DIGILENT_LINUX Tutorial]$
```

### Example 19 – Linaro.

```
[tinghui.wang@DIGILENT_LINUX Tutorial]$ ls
devicetree.dtb  linux-digilent  u-boot-digilent  ZedBoard_Linux_Design
[tinghui.wang@DIGILENT_LINUX Tutorial]$ cp ZedBoard_Linux_Design/boot_image/BOOT.BIN
/media/ZED_BOOT
[tinghui.wang@DIGILENT_LINUX Tutorial]$ cp ./devicetree.dtb /media/ZED_BOOT
[tinghui.wang@DIGILENT_LINUX Tutorial]$ cp linux-digilent/arch/arm/boot/zImage
/media/ZED_BOOT
[tinghui.wang@DIGILENT_LINUX Tutorial]$
```

- V-4** Plug SD Card into ZedBoard. To boot from SD card, jumpers for MIO2, MIO3, MIO6 are connected to GND, and jumpers for MIO4 and MIO5 are connected to 3V3. Jumper for JP6 needs to be connected. Connect UART port to PC with 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 power on the board, the console for RamDisk is shown in Figure 43 – RamDisk, and the console for Linaro File System is shown in Figure 43 – Linaro.

## Section VI: Modify Device Tree and Compose Kernel Driver

### Prerequisites

➤ **Xilinx ISE DS 14.4 (WebPack)**

available at Xilinx Website Download Page

➤ *Linux Kernel Source Code*

available at Digilent github repository: <https://github.com/Digilent/linux-digilent>

### Instructions

**VI-1** Create drivers directory under Tutorial folder, as shown in Example 20. Inside the drivers directory, we will compose the driver for myled controller.

#### Example 20.

```
[tinghui.wang@DIGILENT_LINUX Tutorial]$ mkdir drivers
[tinghui.wang@DIGILENT_LINUX Tutorial]$ ls
devicetree.dtb  drivers  inux-digilent  u-boot-digilent  ZedBoard_Linux_Design
[tinghui.wang@DIGILENT_LINUX Tutorial]$
```

**VI-2** We need a Makefile so that we can compile the driver. The Makefile goes as Example 20.

#### Example 21 – Command Line.

```
[tinghui.wang@DIGILENT_LINUX Tutorial]$ cd drivers
[tinghui.wang@DIGILENT_LINUX drivers]$ vim Makefile
```

#### Example 21 – Makefile.

```
1 obj-m := myled.o
2
3 all:
4     make -C ../linux-digilent/ M=$(PWD) modules
5
6 clean:
7     make -C ../linux-digilent/ M=$(PWD) clean
```

**VI-3** We will start with a simple driver that create a file named myled under Linux /proc file system. The status of on-board LEDs can be changed by write a number to the file. The driver goes as Example 22.

#### Example 22 – Command Line

```
[tinghui.wang@DIGILENT_LINUX drivers]$ vim myled.c
```

#### Example 22 – myled.c

```
1 #include <linux/kernel.h>
2 #include <linux/module.h>
3 #include <asm/uaccess.h> /* Needed for copy_from_user */
4 #include <asm/io.h> /* Needed for IO Read/Write Functions */
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 */
```

## Example 22 – myled.c (Cont.)

```
8
9 /* Define Driver Name */
10 #define DRIVER_NAME "myled"
11
12 unsigned long *base_addr; /* Virtual Base Address */
13 struct resource *res; /* Device Resource Structure */
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.
21 * It will then write the value to the register of myled controller,
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,
48 * and seq_read function will print whatever in seq_file to the terminal.
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
```

## Example 22 – myled.c (Cont.)

```
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;
68     struct seq_file *m;
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->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,
91     .read = seq_read,
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
```

## Example 22 – 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 */
116     remove_proc_entry(DRIVER_NAME, NULL);
117
118     /* Release mapped virtual address */
119     iounmap(base_addr);
120
121     /* Release the region */
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 region 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 __devinit 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);
152     if (base_addr == NULL) {
153         dev_err(&pdev->dev, "Couldn't ioremap memory at 0x%08lx\n",
154             (unsigned long)res->start);
155         ret = -ENOMEM;
156         goto err_release_region;
157     }
158
```



## Example 22 – myled.c (Cont.)

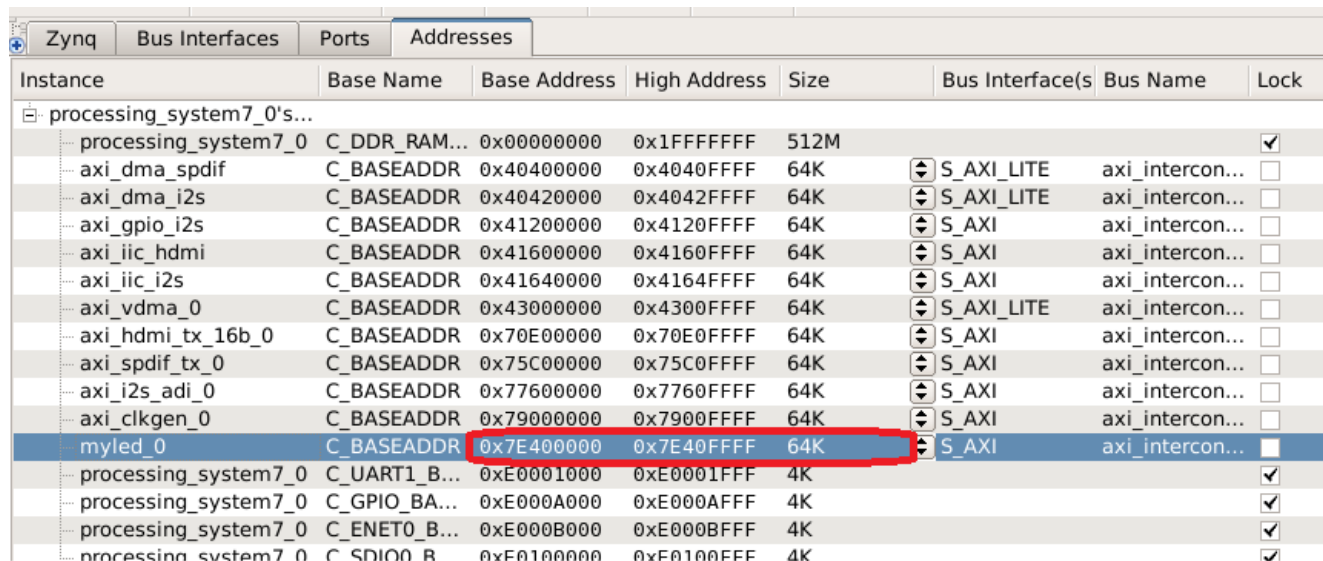
```
159     myled_proc_entry = proc_create(DRIVER_NAME, 0, NULL,
160                                   &proc_myled_operations);
161     if (myled_proc_entry == NULL) {
162         dev_err(&pdev->dev, "Couldn't create proc entry\n");
163         ret = -ENOMEM;
164         goto err_create_proc_entry;
165     }
166
167     printk(KERN_INFO DRIVER_NAME " probed at VA 0x%08lx\n",
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[] __devinitconst = {
182     {.compatible = "dglnt,myled-1.00.a"},
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 = __devexit_p(myled_remove),
196     .shutdown = __devexit_p(myled_shutdown)
197 };
198
199 /* Register myled platform driver */
200 module_platform_driver(myled_driver);
201
202 /* Module Informations */
203 MODULE_AUTHOR("Diligent, Inc.");
204 MODULE_LICENSE("GPL");
205 MODULE_DESCRIPTION(DRIVER_NAME ": MYLED driver (Simple Version)");
206 MODULE_ALIAS(DRIVER_NAME);
207
```

## VI-4 Compile and generate the driver module by make as shown in Example 23.

### Example 23

```
[tinghui.wang@DIGILENT_LINUX drivers]$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-
make -C ../linux-digilent/
M=/home/tinghui.wang/Workspace/ZedBoard/Project/14_4/Tutorial/drivers modules
make[1]: Entering directory
`/home/tinghui.wang/Workspace/ZedBoard/Project/14_4/Tutorial/linux-digilent'
CC [M] /home/tinghui.wang/Workspace/ZedBoard/Project/14_4/Tutorial/drivers/myled.o
Building modules, stage 2.
MODPOST 1 modules
CC /home/tinghui.wang/Workspace/ZedBoard/Project/14_4/Tutorial/drivers/myled.mod.o
LD [M] /home/tinghui.wang/Workspace/ZedBoard/Project/14_4/Tutorial/drivers/myled.ko
make[1]: Leaving directory
`/home/tinghui.wang/Workspace/ZedBoard/Project/14_4/Tutorial/linux-digilent'
[tinghui.wang@DIGILENT_LINUX drivers]$
```

**VI-5** We need to add the **myled** device node into the device tree. Make a copy of default device tree source in **drivers** folder, and modify it according to Example 24. The compatibility string of the node is the same as we define in driver source code (myled.c: line 182). The **reg** property defines the physical address space of the node. The address here should match with the address of myled IP core in address tab in EDK design, as shown in Figure 44.



| Instance                  | Base Name    | Base Address | High Address | Size | Bus Interface(s) | Bus Name        | Lock                                |
|---------------------------|--------------|--------------|--------------|------|------------------|-----------------|-------------------------------------|
| processing_system7_0's... |              |              |              |      |                  |                 |                                     |
| ... processing_system7_0  | C_DDR_RAM... | 0x00000000   | 0x1FFFFFFF   | 512M |                  |                 | <input checked="" type="checkbox"/> |
| ... axi_dma_spdif         | C_BASEADDR   | 0x40400000   | 0x4040FFFF   | 64K  | S_AXI_LITE       | axi_intercon... | <input type="checkbox"/>            |
| ... axi_dma_i2s           | C_BASEADDR   | 0x40420000   | 0x4042FFFF   | 64K  | S_AXI_LITE       | axi_intercon... | <input type="checkbox"/>            |
| ... axi_gpio_i2s          | C_BASEADDR   | 0x41200000   | 0x4120FFFF   | 64K  | S_AXI            | axi_intercon... | <input type="checkbox"/>            |
| ... axi_iic_hdmi          | C_BASEADDR   | 0x41600000   | 0x4160FFFF   | 64K  | S_AXI            | axi_intercon... | <input type="checkbox"/>            |
| ... axi_iic_i2s           | C_BASEADDR   | 0x41640000   | 0x4164FFFF   | 64K  | S_AXI            | axi_intercon... | <input type="checkbox"/>            |
| ... axi_vdma_0            | C_BASEADDR   | 0x43000000   | 0x4300FFFF   | 64K  | S_AXI_LITE       | axi_intercon... | <input type="checkbox"/>            |
| ... axi_hdmi_tx_16b_0     | C_BASEADDR   | 0x70E00000   | 0x70E0FFFF   | 64K  | S_AXI            | axi_intercon... | <input type="checkbox"/>            |
| ... axi_spdif_tx_0        | C_BASEADDR   | 0x75C00000   | 0x75C0FFFF   | 64K  | S_AXI            | axi_intercon... | <input type="checkbox"/>            |
| ... axi_i2s_adi_0         | C_BASEADDR   | 0x77600000   | 0x7760FFFF   | 64K  | S_AXI            | axi_intercon... | <input type="checkbox"/>            |
| ... axi_clkgen_0          | C_BASEADDR   | 0x79000000   | 0x7900FFFF   | 64K  | S_AXI            | axi_intercon... | <input type="checkbox"/>            |
| ... myled_0               | C_BASEADDR   | 0x7E400000   | 0x7E40FFFF   | 64K  | S_AXI            | axi_intercon... | <input type="checkbox"/>            |
| ... processing_system7_0  | C_UART1_B... | 0xE0001000   | 0xE0001FFF   | 4K   |                  |                 | <input checked="" type="checkbox"/> |
| ... processing_system7_0  | C_GPIO_BA... | 0xE000A000   | 0xE000AFFF   | 4K   |                  |                 | <input checked="" type="checkbox"/> |
| ... processing_system7_0  | C_ENET0_B... | 0xE000B000   | 0xE000BFFF   | 4K   |                  |                 | <input checked="" type="checkbox"/> |
| ... processing_system7_0  | C_SDIO0_B... | 0xF0100000   | 0xF0100FFF   | 4K   |                  |                 | <input checked="" type="checkbox"/> |

Figure 44. Physical Address for myled IP Core

### Example 24 – Command Line

```
[tinghui.wang@DIGILENT_LINUX drivers]$ cp ../linux-digilent/arch/arm/boot/dts/digilent-
zed.dts ./
[tinghui.wang@DIGILENT_LINUX drivers]$ vim digilent-zed.dts
```

### Example 24 – diligent-zed.dts

```
549         spi-speed-hz = <4000000>;
550         spi-sclk-gpio = <gpio7_gpio_0 59 0>;
551         spi-sdin-gpio = <gpio7_gpio_0 60 0>;
552     };
553
554     myled {
555         compatible = "dglnt,myled-1.00.a";
556         reg = <0x7e400000 0x10000>;
557     };
558 };
559 };
```

**VI-6** Recompile the device tree blob as shown in Example 25.

### Example 25

```
[tinghui.wang@DIGILENT_LINUX drivers]$ ../linux-digilent/scripts/dtc/dtc -I dts -O dtb -o
devicetree.dtb diligent-zed.dts
DTC: dts->dtb on file "diligent-zed.dts"
[tinghui.wang@DIGILENT_LINUX drivers]$
```

**VI-7** Copy these two files to the first partition of SD card and we are ready to test our driver on-board now, as shown in Example 26.

### Example 26.

```
[tinghui.wang@DIGILENT_LINUX drivers]$ ls
devicetree.dtb  Makefile      Module.symvers  myled.ko      myled.mod.o
diligent-zed.dts  modules.order  myled.c         myled.mod.c  myled.o
[tinghui.wang@DIGILENT_LINUX drivers]$ cp myled.ko /media/ZED_BOOT/d
[tinghui.wang@DIGILENT_LINUX drivers]$ cp devicetree.dtb /media/ZED_BOOT/
[tinghui.wang@DIGILENT_LINUX drivers]$
```

**VI-8** Plug the SD card into ZedBoard, and we can start testing our driver now. Use **insmod** command to install driver module into the kernel. After the driver is installed, **myled** entry will be created under **/proc** file system. Writing **0x0F** to the **/proc/myled** will light up **LED 0~3**, while writing **0xF0** to **myled** entry in **/proc** file system will light up **LED 4~7**. You can either remove the driver with command **rmmmod** or power off the system by command **poweroff**. In both case, all the LEDs will be turned off, as shown in Example 27. (How to use terminal with ZedBoard, please refer to Step V-4 or Section **Boot from SD** in **Getting Started with Embedded Linux – ZedBoard**).

## Example 27 – RAMDISK

```
U-Boot 2012.04.01-dirty (Feb 01 2013 - 12:52:36)

DRAM: 512 MiB
WARNING: Caches not enabled
MMC: SDHCI: 0
Using default environment
...
reading zImage

2457328 bytes read
reading devicetree.dtb

9728 bytes read
reading ramdisk8M.image.gz

3694108 bytes read
## Starting application at 0x00008000 ...
Uncompressing Linux... done, booting the kernel.
[ 0.000000] Booting Linux on physical CPU 0
[ 0.000000] Linux version 3.6.0-digilent-13.01-00002-g06b3889
(tinghui.wang@DIGILENT_LINUX) (gcc version 4.6.3 (Sourcery CodeBench Lite 2012.03-79) ) #1
SMP PREEMPT Sun Feb 10 23:54:12 PST 2013
...
rcS Complete
zynq> mount /dev/mmcblk0p1 /mnt/
zynq> cd /mnt/
zynq> ls
BOOT.BIN          devicetree.dtb    ramdisk8M.image.gz
myled.ko          zImage
zynq> insmod myled.ko
[ 122.160000] myled probed at va 0xe0d20000
zynq> ls /proc
1          567          9           fs           partitions
10         582          asound      interrupts   scsi
11         588          buddyinfo   iomem        self
12         594          bus          ioports      slabinfo
13         595          cmdline     irq           softirqs
14         596          config.gz   kallsyms     stat
15         6           consoles    kmsg         swaps
2          608          cpu         kpagecount   sys
3          614          cpuinfo     kpageflags   sysvipc
317        615          crypto      loadavg       timer_list
318        621          device-tree locks         tty
333        641          devices     meminfo       uptime
4          642          diskstats   misc          version
429        643          dma         modules       vmallocinfo
440        647          dri         mounts        vmstat
441        652          driver      mtd           zoneinfo
5          653          execdomains myled
515        7           fb          net
548        8           filesystems pagetypeinfo
zynq> echo 0x0F > /proc/myled
zynq> cat /proc/myled
0x0f
zynq> echo 0xF0 > /proc/myled
zynq> cat /proc/myled
0xf0
zynq> mkdir -p /lib/modules/`uname -r`
zynq> cp myled.ko /lib/modules/`uname -r`
zynq> rmmod myled
```

## Example 27 – Linaro FS

```
U-Boot 2012.04.01-dirty (Feb 01 2013 - 12:52:36)

DRAM: 512 MiB
WARNING: Caches not enabled
MMC: SDHCI: 0
...
reading zImage

2457328 bytes read
reading devicetree.dtb

9728 bytes read
reading ramdisk8M.image.gz

3694108 bytes read
## Starting application at 0x00008000 ...
Uncompressing Linux... done, booting the kernel.
[ 0.000000] Booting Linux on physical CPU 0
[ 0.000000] Linux version 3.6.0-digilent-13.01-00002-g06b3889
(tinghui.wang@DIGILENT_LINUX) (gcc version 4.6.3 (Sourcery CodeBench Lite 2012.03-79) ) #1
SMP PREEMPT Sun Feb 10 23:54:12 PST 2013
...
Last login: Thu Jan 1 00:00:10 UTC 1970 on tty1
cat: /var/lib/update-notifier/fsck-at-reboot: No such file or directory
run-parts: /etc/update-motd.d/98-fsck-at-reboot exited with return code 1
Welcome to Linaro 12.11 (GNU/Linux 3.6.0-digilent-13.01-00002-g06b3889 armv7l)

* Documentation: https://wiki.linaro.org/

0 packages can be updated.
0 updates are security updates.

root@linaro-ubuntu-desktop:~# mount /dev/mmcbk0p1 /mnt/
root@linaro-ubuntu-desktop:~# cd /mnt/
root@linaro-ubuntu-desktop:/mnt# insmod myled.ko
root@linaro-ubuntu-desktop:/mnt# ls /proc
1      1357  1981  2088  2189  2345  514  asound          fs           pagetypeinfo
10     1367  1983  2101  2194  2440  547  buddyinfo      interrupts  partitions
1054   1369  1993  2104  2200  2452  548  bus            iomem       scsi
11     1372  1995  2105  2211  2464  565  cmdline        ioports     self
1112   1378  1999  2113  2223  2488  568  config.gz      irq         slabinfo
1128   1379  2     2116  2225  2536  589  consoles       kallsyms    softirqs
1129   1380  2011  2119  2226  2547  591  cpu            kmsg        stat
1177   14    2017  2125  2228  2562  594  cpuinfo        kpagecount  swaps
1178   1408  2036  2133  2246  2574  597  crypto         kpageflags  sys
1199   1450  2064  2135  2260  2594  598  device-tree    loadavg     sysvipc
12     15    2066  2137  2262  3     6    devices        locks        timer_list
1228   1625  2072  2149  2268  317   603  diskstats     meminfo     tty
1236   1637  2073  2152  2271  318   669  dma           misc         uptime
1240   1730  2078  2153  2277  333   676  dri           modules      version
1243   1837  2079  2155  2279  4     7    driver        mounts       vmallocinfo
1284   1966  2080  2157  2294  439   737  execdomains   mtd          vmstat
13     1969  2085  2159  2320  440   8    fb            myled       zoneinfo
1354   1970  2086  2161  2340  5     9    filesystems   net

root@linaro-ubuntu-desktop:/mnt# echo 0x0F > /proc/myled
root@linaro-ubuntu-desktop:/mnt# echo 0xF0 > /proc/myled
root@linaro-ubuntu-desktop:/mnt# rmmod myled
root@linaro-ubuntu-desktop:/mnt#
```

## Section VII: User Application

### Prerequisites

- **Xilinx ISE DS 14.4 (WebPack)**  
available at [Xilinx Website Download Page](#)

### Instructions

**VII-1** In this section, we will write a user application that makes LED blinking by writing to file `/proc/myled`. Create **user\_app** directory under Tutorial folder, as shown in Example 28. Inside the **user\_app** directory, we will compose the `led_blink.c`, as shown in Example 29.

#### Example 28.

```
[tinghui.wang@DIGILENT_LINUX Tutorial]$ mkdir user_app
[tinghui.wang@DIGILENT_LINUX Tutorial]$ ls
devicetree.dtb  drivers  inux-digilent  u-boot-digilent  user_app  ZedBoard_Linux_Design
[tinghui.wang@DIGILENT_LINUX Tutorial]$
```

#### Example 29 – Command Line.

```
[tinghui.wang@DIGILENT_LINUX user_app]$ vim led_blink.c
```

#### Example 29 – led\_blink.c.

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

VII-2 Compose a Makefile and compile led\_blink.c into led\_blink.elf. As shown in Example 30.

### Example 30 – Command Line.

```
[tinghui.wang@DIGILENT_LINUX user_app]$ vim Makefile
```

### Example 30 – 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} -o $^ $@
8
9 clean :
10     rm -rfv *.o
11     rm -rfv led_blink
12
13 .PHONY : clean
```

### Example 30 – Compile led\_blink.

```
[tinghui.wang@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
[tinghui.wang@DIGILENT_LINUX user_app]$ ls
led_blink led_blink.c led_blink.o Makefile
[tinghui.wang@DIGILENT_LINUX user_app]$
```

VII-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 Example 31.

### Example 31.

```
[tinghui.wang@DIGILENT_LINUX user_app]$ cp led_blink /media/ZED_BOOT/
```

VII-4 Plug the SD card into ZedBoard. Use **insmod** command to install driver module into the kernel. Run led\_blink and the LED will start blinking.

### Example 32 – RAMDISK

```
...
rcS Complete
zynq> mount /dev/mmcblk0p1 /mnt/
zynq> cd /mnt/
zynq> ls
BOOT.BIN          devicetree.dtb    led_blink
myled.ko          ramdisk8M.image.gz zImage
zynq> insmod myled.ko
[ 122.160000] myled probed at va 0xe0d20000
zynq> ./led_blink
^C
zynq> mkdir -p /lib/modules/`uname -r`
zynq> cp myled.ko /lib/modules/`uname -r`
zynq> rmmmod myled
```

## Example 32 – Linaro FS

```
...
root@linaro-ubuntu-desktop:~# mount /dev/mmcblk0p1 /mnt/
root@linaro-ubuntu-desktop:~# cd /mnt/
root@linaro-ubuntu-desktop:/mnt# insmod myled.ko
root@linaro-ubuntu-desktop:/mnt# ./led_blink
^C
root@linaro-ubuntu-desktop:/mnt# rmmod myled
root@linaro-ubuntu-desktop:/mnt#
```