Lab 8: Custom Driver Development – AXI Device

Introduction

The UIO framework allows you to quickly develop device drivers that can be controlled from user space. As the FIR AXI IP core you created in Lab 7 is a very simple device, writing a kernel-space driver for it is unnecessary. A UIO driver is a better choice with the functionality and complexity moved to user space.

In this lab, you will use UIO to access the IP core.

Objectives

After completing this lab, you will be able to:

- Write a UIO program to access the FIR AXI IP core
- Create the image after enabling the UIO drivers
- Generate the SD card image
- Verify the functionality after booting the board from the SD card

Preparation

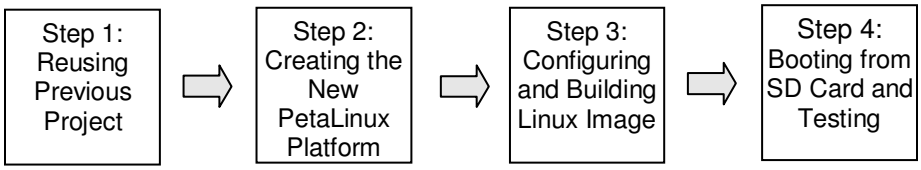
If this is the first lab that you are performing, then refer to the "Before You Start" section of Lab 1 for necessary preparatory information on how to set up the environment.

If your workstation has been restarted or logout, run the following command to start DHCP server on the host:

```
[host] $ sudo service isc-dhcp-server restart
```

Please refer to the "Initializing the Workshop Environment" section of Lab 1 for detailed information.

General Flow for this Lab



Reusing the Previous Vivado Design Suite Project — Step 1

You will use the design created in the Lab 7. You will save this previous project so that it can be used as the current lab project.

1-1. Create a `lab8` directory under the `labs` folder. Create a directory called `hardware` under the created `lab8` directory.

1-1-1. Create and change to the project directory by executing the following commands:

```
[host] $ mkdir ~/emblnx/labs/lab8
[host] $ mkdir ~/emblnx/labs/lab8/hardware
[host] $ cd ~/emblnx/labs/lab8/hardware
```

1-2. Launch the Vivado Design Suite. Open the Vivado Design Suite project of the previous lab (lab7) and save the project as lab8.

1-2-1. Launch the Vivado Design Suite, if it is not already open, by executing the following command:

```
[host] $ vivado
```

If the program does not start then run the following command on the host machine to source the setup script:

```
[host] $ source /opt/pkg/Xilinx/Vivado/2014.2/settings32.sh
```

1-2-2. Click the **Open Project** link in the Getting Started page.

1-2-3. Click **Browse Projects**, browse to the *~/emblnx/labs/lab7/hardware* directory, and select the **lab7.xpr** entry.

1-2-4. Select **File** > **Save Project As**.

1-2-5. Click the Browse button next to the Project location field and browse to the */home/petalinux/emblnx/labs/lab8/hardware* directory.

1-2-6. Enter **lab8** in the Project name field.

1-2-7. Verify that the **Create project subdirectory** option is not selected.

1-2-8. Click **OK**.

1-3. Set the project settings to point to the IPs in the sources/lab8 directory. Reset the output products.

1-3-1. Click **Project Settings** under Project Manager in the Flow Navigator pane.

1-3-2. Select **IP** in the left pane of the Project Settings window.

1-3-3. Select the entry in the IP Repositories section and click the  button on the right to remove it.

1-3-4. Click **Add IP Repository** and browse to the */home/petalinux/emblnx/sources/lab8* directory.

1-3-5. Click **Select**.

The Fir and zed_audio_ctrl entries will show up in the *IP in Selected Repository* section.

1-3-6. Click **OK**.

1-3-7. In the **Sources** > **Hierarchy** window, expand the *system_wrapper* hierarchy, right-click on *system_i* and select **Reset Output Products…**

1-3-8. Click **Reset**.

1-4. Generate the bitstream. Export the hardware making sure that the Include Bitstream option is selected.

1-4-1. Click **Generate Bitstream** in the Flow Navigator to run the synthesis, implementation, and bitstream generation processes.

1-4-2. Click **Yes** to re-run the synthesis process.

1-4-3. When the bitstream generation process completes, click **OK** to open the implemented design.

1-4-4. Select **File > Export > Export Hardware**.

1-4-5. In the *Export Hardware* form make sure that the Include Bitstream and Export to local project directory are selected.

1-4-6. Click **OK**.

1-4-7. Select **File** > **Exit** in the Vivado Design Suite

Creating a New Platform with PetaLinux Configuration Options Step 2

The next step is to create a new PetaLinux SDK software platform, ready for building a Linux system customized to your new hardware platform.

2-1. Verify the PetaLinux working environment or set it by running the PetaLinux setup script.

2-1-1. Run the following command on the host machine to see if the PetaLinux tools are installed and sourced properly

```
[host] $ echo $PETALINUX

It should display

/opt/pkg/petalinux-v2014.2-final
```

2-1-2. If it doesn't then run the following command on the host machine to source the set up script:

```
[host] $ source /opt/pkg/petalinux-v2014.2-final/settings.sh
```

2-2. Use the `petalinux-create` command to create a new embedded Linux platform in the project directory.

2-2-1. Change to the project directory. Enter the following command:

```
cd ~/emblnx/labs/lab8
```

2-2-2. Run the following command to create a new Petalinux project:

```
[host] $ petalinux-create -t project --name software
```

Configuring and Building the Linux System Step 3

The next step is to customize the software platform template to precisely match your unique hardware system. This is done by copying and merging the platform configuration files generated during the hardware build phase into the newly created software platform.

3-1. Configure the PetaLinux software project based on the hardware and set the SD Card as the boot device.

3-1-1. Change the directory to the `<XSDK workspace directory>` directory:

```
[host] $ cd ~/emblnx/labs/lab8/hardware/lab8.sdk
```

3-1-2. Use the `petalinux-config` command to import the hardware configuration:

```
[host] $ petalinux-config --get-hw-description -p
~/emblnx/labs/lab8/software
```

It launches the top system configuration menu when petalinux-config --get-hw-description runs for the first time for the PetaLinux project or the tool detects there is a change in the system primary hardware candidates.

3-1-3. Select the **Subsystem AUTO Hardware Settings** (using down arrow key and pressing *Enter*) to go into the sub-menu.

3-1-4. Move down to *Advanced bootable images storage Settings* option and press **Y** or space bar to enable the setting.

This sub-menu allows users to specify where bootable images are. The settings of this menu are used by PetaLinux auto configured u-boot.

3-1-5. Press **Enter** to go into the **Advanced bootable images storage Settings** sub-menu.

3-1-6. Go into the **boot image storage > image storage media (primary flash)** and select the *primary sd* option.

3-1-7. Press **Enter.**

3-1-8. Select **<Exit>** to go one level up.

3-1-9. Similarly, select the **kernel image settings** and select the *image storage media* as **primary sd**.

3-1-10. Press **Enter,** exit the configuration and save the configuration.

The PetaLinux tools generate the hardware configuration files, if required, and copies the configuration files to the correct location in your project directory.

3-2. Launch the Linux kernel configuration menu and configure it to meet your requirements.

3-2-1. Change into the root directory of your PetaLinux project:

```
[host] $ cd ~/emblnx/labs/lab8/software
```

3-2-2. Run the following command in the terminal:

```
[host] $ petalinux-config -c kernel
```

3-2-3. Select **Device Drivers** from the Linux Kernel Configuration menu.

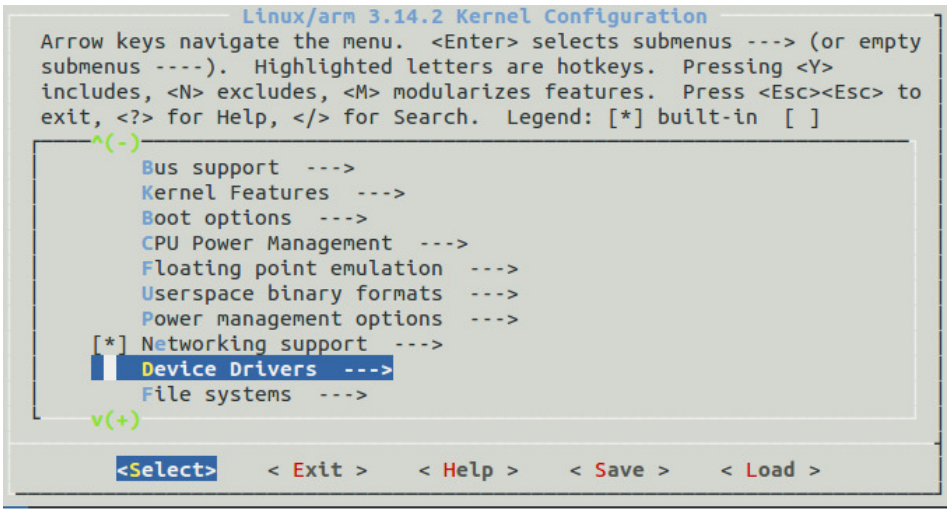


Figure 1. Accessing Device Drivers options

3-2-4. In the Device Drivers menu, scroll down to the **Userspace I/O drivers** select it as **"<*>".**

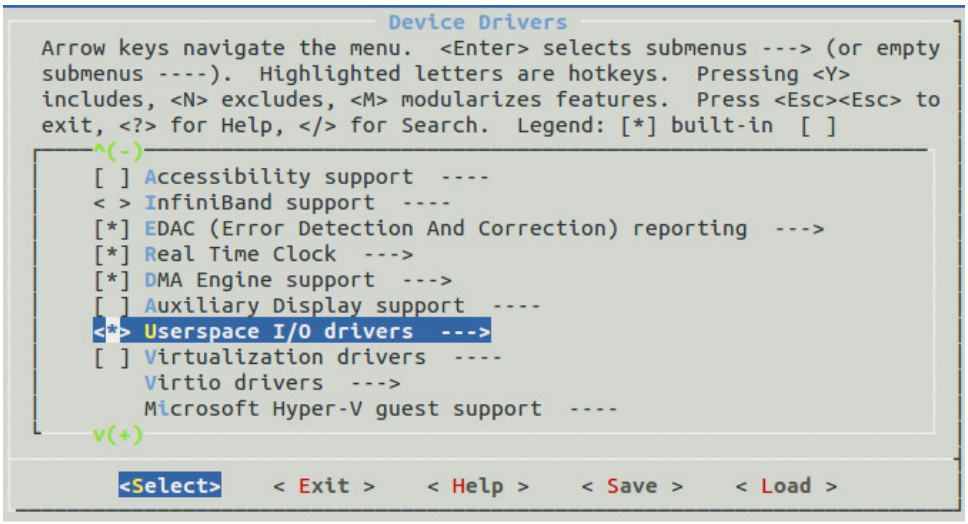


Figure 2. Selecting the UIO Driver as Built-in

Because the kernel is configured to support loadable modules by default, for those loadable device drivers, you can select it as built-it or module. "<*>" means built-in and "<M>" means

module. If a driver is selected as a module, it will not be loaded when booting Linux; you can load it after Linux boots by using the `modprobe` command (see below).

The UIO kernel driver will be selected as built-in.

3-2-5. Hit the **Enter** key to go to its sub-menu and observe that the *Userspace I/O platform driver with generic IRQ handling* is selected.

3-2-6. Exit to one-level back and scroll up until you see the *I2C support*.

3-2-7. Go to the **Device Drivers > I2C Support** menu and verify that it is selected since the CODEC will be configured from the PS (Processing System) using I2C (I2C channel 1).

3-2-8. Exit the configuration, saving the settings.

3-3. Mark the zed_audio_ctrl_0, axi_gpio_0, fir_left and fir_right instances as the UIO devices in the DTS file.

3-3-1. Browse to the directory `~/emblnx/labs/lab8/software/subsystems/linux/configs/device-tree`.

Notice that there are four files- pl.dtsi, ps.dtsi, system-conf.dtsi, and system-top.dts; you can make changes to system-top.dts file to customize your settings as it does not get updated by the PetaLinux tools.

3-3-2. Copy the provided **system-top.dts** file from the `~/emblnx/sources/lab8` and place it in the `~/emblnx/labs/lab8/software/subsystems/linux/configs/device-tree` directory replacing the existing file.

3-3-3. Review the `pl.dtsi` file in editor like gedit.

You will see that it lists two instances of FIR filters, one instance of the GPIO, and one instance of the audio controller. These are the only IPs in the design which are in PL section. Observe that the instances are assigned generic drivers.

```
/*
 * CAUTION: This file is automatically generated by Xilinx.
 * Version: HSM 2014.2
 * Today is: Mon Sep 29 14:05:49 2014
*/

        &ps7_axi_interconnect_0 {
                ranges;
                axi_gpio_0: gpio@41200000 {
                        #gpio-cells = <2>;
                        compatible = "xlnx,xps-gpio-1.00.a";
                        gpio-controller ;
                        reg = <0x41200000 0x10000>;
                        xlnx,all-inputs = <0x0>;
                        xlnx,all-inputs-2 = <0x1>;
                        xlnx,all-outputs = <0x1>;
                        xlnx,all-outputs-2 = <0x0>;
                        xlnx,dout-default = <0x00000000>;
                        xlnx,dout-default-2 = <0x00000000>;
                        xlnx,gpio-width = <0x2>;
                        xlnx,gpio2-width = <0x1>;
                        xlnx,interrupt-present = <0x0>;
                        xlnx,is-dual = <0x1>;
                        xlnx,tri-default = <0xFFFFFFFF>;
                        xlnx,tri-default-2 = <0xFFFFFFFF>;
                } ;
                fir_left: fir@43c10000 {
                        compatible = "xlnx,fir-1.0";
                        interrupt-parent = <&ps7_scugic_0>;
                        interrupts = <0 30 4>;
                        reg = <0x43c10000 0x10000>;
                        xlnx,s-axi-fir-io-addr-width = <0x5>;
                        xlnx,s-axi-fir-io-data-width = <0x20>;
                } ;
                fir_right: fir@43c20000 {
                        compatible = "xlnx,fir-1.0";
                        interrupt-parent = <&ps7_scugic_0>;
                        interrupts = <0 29 4>;
                        reg = <0x43c20000 0x10000>;
                        xlnx,s-axi-fir-io-addr-width = <0x5>;
                        xlnx,s-axi-fir-io-data-width = <0x20>;
                } ;
                zed_audio_ctrl_0: zed-audio-ctrl@43c00000 {
                        compatible = "xlnx,zed-audio-ctrl-1.0";
                        reg = <0x43c00000 0x10000>;
                        xlnx,dphase-timeout = <0x8>;
                        xlnx,num-mem = <0x1>;
                        xlnx,num-reg = <0x1>;
                        xlnx,s-axi-min-size = <0x000001FF>;
                        xlnx,slv-awidth = <0x20>;
                        xlnx,slv-dwidth = <0x20>;
                        xlnx,use-wstrb = <0x0>;
                } ;
        } ;
```

Figure 3. The pl.dtsi file content

3-3-4. Open the **system-top.dts** file in an editor and modify it so it has UIO drivers for the zed_audio_ctrl_0, axi_gpio_0, fir_left and fir_right instances.

```
/dts-v1/;
/include/ "system-conf.dtsi"
/ {
};

&ps7_ethernet_0 {
        phy-handle = <&phy0>;
        mdio {
                #address-cells = <1>;
                #size-cells = <0>;
                phy0: phy@0 {
                        compatible = "marvell,88e1510";
                        device-type = "ethernet-phy";
                        reg = <0>;
                };
        };
};

&fir_left {
        compatible = "generic-uio";
} ;

&fir_right {
        compatible = "generic-uio";
} ;

&zed_audio_ctrl_0 {
        compatible = "generic-uio";
} ;

&axi_gpio_0 {
        compatible = "generic-uio";
} ;
```

Figure 4. Assigning UIO drivers

3-3-5. Save the file and close the editor.

3-4. Create a new user application, called fir-uio-test, to test the FIR IP. Copy the `fir-uio-test.c` and `Makefile` source files from the `sources/lab8/fir-uio-test` directory.

3-4-1. Make sure that you are at the software directory of the current project, i.e. `~/emblnx/labs/lab8/software`

3-4-2. Enter the following command to create a new user application inside a PetaLinux project:

```
[host] $ petalinux-create -t apps --name fir-uio-test
```

The new application will be created in the `~/emblnx/labs/lab8/software/components/apps/fir-uio-test` directory.

3-4-3. Change to the newly created application directory:

```
[host] $ cd ~/emblnx/labs/lab8/software/components/apps/fir-uio-test
```

3-4-4. Replace the existing `fir-uio-test.c` file with the provided source files from the `sources/lab8` directory:

```
[host] $ cp ~/emblnx/sources/lab8/fir-uio-test/* ./
```

This will copy the fir-uio-test.c and Makefile. The Makefile adds the compiler flag, CFLAG, with optimization turned OFF. This is required so the compiler does not optimize the code that refers to the same memory mapped I/O register for successive write.

3-4-5. Review the `fir-uio-test.c` source code to see how to access a device using UIO.

As can be seen, you need to open the `/dev/uioX` file and then use `mmap()` to map the device to the application's address space. Then you can access the device by using the pointer returned from the `mmap()` call.

3-5. Create another user application, called codec-test, to test the zed-audio-ctrl IP. Copy the `codec-test.c`, `audio.h` and `Makefile` source files from the `sources/lab8/codec-test` directory.

3-5-1. Make sure that you are at the software directory of the current project, i.e. `~/emblnx/labs/lab8/software`

3-5-2. Enter the following command to create a new user application inside a PetaLinux project:

```
[host] $ petalinux-create -t apps --name codec-test
```

The new application will be created in the `~/emblnx/labs/lab8/software/components/apps/codec-test` directory.

3-5-3. Change to the newly created application directory:

```
[host] $ cd ~/emblnx/labs/lab8/software/components/apps/codec-test
```

3-5-4. Replace the existing `codec-test.c` file with the provided source, add `audio.h` and `Makefile` files from the `sources/lab8/codec-test` directory:

```
[host] $ cp ~/emblnx/sources/lab8/codec-test/* ./
```

The Makefile adds the compiler flag CFLAG with optimization turned OFF.

3-5-5. Review the `codec-test.c` source code to see how to access a device using UIO and I2C device using the I2C driver.

As can be seen, you need to open the `/dev/uioX` file and then use `mmap()` to map the device to the application's address space. Then you can access the device by using the pointer returned from the `mmap()` call. Similarly, you need to open the `/dev/i2c-dev` device and then use `mmap()`.

3-6. Create a new user application, called zed-audio-test, to test the CODEC and FIR cores. Copy the `zed-audio-test.c`, `audio.h` and `MakeFile` source files from the `sources/lab8/zed-audio-test` directory.

3-6-1. Make sure that you are at the software directory of the current project, i.e. `~/emblnx/labs/lab8/software`

3-6-2. Enter the following command to create a new user application inside a PetaLinux project:

```
[host] $ petalinux-create -t apps --name zed-audio-test
```

The new application will be created in the `~/emblnx/labs/lab8/software/components/apps/zed-audio-test` directory.

3-6-3. Change to the newly created application directory:

```
[host] $ cd ~/emblnx/labs/lab8/software/components/apps/zed-audio-test
```

3-6-4. Replace the existing `zed-audio-test.c` file with the provided source from the `sources/lab8/zed-audio-test` directory and also add the provided `audio.h` and `Makefile` files:

```
[host] $ cp ~/emblnx/sources/lab8/zed-audio-test/* ./
```

The Makefile adds the compiler flag CFLAG with optimization turned OFF.

3-6-5. Review the `zed-audio-test.c` source code to see how to access various devices using UIO.

3-7. Launch the rootfs configuration menu and include the `fir-uio-test`, `code-test`, and `zed-audio-test` applications

3-7-1. Run the following command in the terminal:

```
[host] $ petalinux-config -c rootfs
```

3-7-2. Select **Apps**.

3-7-3. Enable **fir-uio-test, codec-test,** and **zed-audio-test** by clicking <**Y**>.

3-7-4. Select **Exit** two times and save the configuration.

3-8. Build the system image.

3-8-1. Run the `petalinux-build` command again to build the system image:

```
[host] $ petalinux-build
```

Booting the PetaLinux Image with SD Card and Testing Step 4

Having configured to use the SD card as the primary boot device, you will create a boot image file that contains the Zynq All Programmable SoC FSBL, the BIT file for the programmable logic (PL) configuration, u-boot, and the Linux image for the SD card boot.

4-1. Create a BOOT.BIN file to boot from the SD card.

4-1-1. Change the directory to where linux image is generated by executing the following command.

```
[host] cd ~/emblnx/labs/lab8/software/images/linux
```

4-1-2. Enter the following command in the console:

```
petalinux-package --boot --fsbl zynq_fsbl.elf --fpga
~/emblnx/labs/lab8/hardware/lab8.runs/impl_1/system_wrapper.bit --uboot
```

The `BOOT.BIN` file will be generated and the system_wrapper.bit file will be copied in the linux image directory; i.e., `~/emblnx/labs/lab8/software/images/linux`.

4-1-3. Copy the BOOT.BIN and image.ub files from the `~/emblnx/labs/lab8/software/images/linux` directory to the SD card.

4-2. Set the jumper settings for booting from the SD card. Connect the board and power it ON and set the serial port terminal.

4-2-1. Make sure that the jumper settings for booting from SD card are set as shown below.

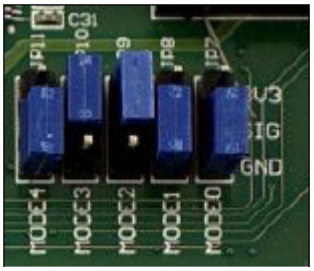


Figure 5. Jumper setting for SD card boot

4-2-2. Power ON the board.

4-2-3. Make sure that `/dev/ttyACM0` is set to read/write access:

```
[host] $ sudo chmod 666 /dev/ttyACM0
```

4-2-4. In the dashboard, in the Search field, enter the serial port.

4-2-5. Select the **Serial port terminal** application and select the appropriate port and 115200 baud.

4-3. Log in and test various applications.

4-3-1. In the terminal, press the <**Enter**> key.

4-3-2. Log into the system using `root` as the user name and password.

4-3-3. Run `mdev -s` to make sure `/dev/uioX` correctly represents the UIO device:

```
# mdev -s
```

The `mdev` commands automatically create device files in `/dev` for the devices found in `/sys/*`.

4-3-4. Go to the `/sys/class/uio` directory in the GtkTerm console (Serial port):

```
# cd /sys/class/uio
```

4-3-5. Check which UIO represents the FIR devices by checking the `uioX/name` file:

```
# cat uio<X>/name
```

In this example, `uio1/name` is the `FIR_LEFT`, which matches the first part of the FIR device label before "@" in the DTS and means that uio1 links to the FIR left channel device.
The other UIO: uio2 links to the FIR right channel device, UIO: uio0 links to the GPIO and UIO:uio3 links to the audio controller.

To verify which UIOx maps to which address, you can use the following command:

```
# cat uio<X>/maps/map0/addr
```

4-3-6. Try `fir-uio-test` to send the impulse input to the filter and get the response back displaying the coefficients:

```
# fir-uio-test
```

You will see the following output:

```
FIR UIO test.
FIR filters Enabled

... Left Filter Output...
17a,49,ffe5,ff56,fed6,fea0,fed2,ff58,fff2,50,40,ffcb,ff46,ff28,ffd8,
164,363,503,556,3ba,33,fb94,f74d,f4f3,f5a9,f99f,ffe7,6b0,be2,8dd3,be
2,6b0,ffe7,f99f,f5a9,f4f3,f74d,fb94,33,3ba,556,503,363,164,ffd8,ff28
,ff46,ffcb,40,50,fff2,ff58,fed2,fea0,fed6,ff56,ffe5,49,17a,0,
... Right Filter Output...
17a,49,ffe5,ff56,fed6,fea0,fed2,ff58,fff2,50,40,ffcb,ff46,ff28,ffd8,
164,363,503,556,3ba,33,fb94,f74d,f4f3,f5a9,f99f,ffe7,6b0,be2,8dd3,be
2,6b0,ffe7,f99f,f5a9,f4f3,f74d,fb94,33,3ba,556,503,363,164,ffd8,ff28
,ff46,ffcb,40,50,fff2,ff58,fed2,fea0,fed6,ff56,ffe5,49,17a,0,
Prog ended
```

Figure 6. The fir-uio-test output

4-3-7. Try `codec-test` to play or mute the music. Connect an audio patch cable between the PC speaker out jack and LINE IN of the ZedBoard. Play music using an audio player.

```
# codec-test
```

Place SW0 to the UP position then you will hear the music, down to mute. Press any key to terminate the program.

4-3-8. Try `zed-audio-test` to play or filter the music.

```
# zed-audio-test
```

Place SW0 to the UP position to listen the music but about 4 KHz signal filtered, down to listen unfiltered. Press any key to terminate the program.

4-3-9. Power OFF the board.

Conclusion

In this lab, you have learned how to:

- Write a user application to access a device on UIO
- Edit the DTS file to link devices to the generic-uio driver
- Generate a boot image and boot the board using SD card

Completed Solution

Copy **`BOOT.bin`** and **`image.ub`** from the *`labsolution\lab8\SDCard`* directory onto a SD card. Place the SD card in the ZedBoard. Set ZedBoard in the SD Card boot mode. Connect the ZedBoard's Line In jack to the speaker out of the host machine, and connect a headphone to the Line Out jack of the board.

Power ON the board. Set the terminal session.

Login into the system and test the lab.