Lab 5: Accessing Hardware Devices from User Space

Introduction

For many types of devices, there is a frequent need to handle an interrupt and provide access to the memory space of the device. The logic of controlling the device does not necessarily have to be within the kernel if the device does not need to take advantage of any of other resources that the kernel provides.

In this lab, you will explore two ways of achieving direct access to the hardware from user space: direct access via `/dev/mem`, and the User Space I/O (UIO) framework.

Writing a kernel driver is overkill for some devices, and the development process is more complicated because it requires writing kernel code. In this lab session, you will create your first very simple UIO driver and learn how to load a module in Linux.

Objectives

After completing this lab, you will be able to:

- Access a hardware device directly from user space
- Use the UIO framework to access a hardware device
- Experience loading and unloading kernel modules

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

Accessing the Device from User Space Step 1

1-1. Change the path to the project directory.

1-1-1. Run the following commands to create and change to the project directory path:

```
[host] $ mkdir ~/emblnx/labs/lab5

[host] $ cd ~/emblnx/labs/lab5
```

1-2. Use the `petalinux-create` command to create a new embedded Linux platform and choose the platform.

1-2-1. Run the following command to create a new Petalinux project:

```
[host] $ petalinux-create -t project -s /opt/pkg/ZYBO_petalinux_v2014_2-final.bsp
```

The command will create the software project directory: `ZYBO_petalinux_v2014_2` under `~/emblnx/labs/lab5`.

1-2-2. Change the directory to the Petalinux project: `~/emblnx/labs/lab5/ZYBO_petalinux_v2014_2`.

1-3. Create a new user application to access GPIO devices from user space.

The PetaLinux tools allow you to create user application templates for either C or C++. These templates include application source code and makefiles so that you can easily configure and compile applications for the target and install them into the root file system.

1-3-1. Enter the following command to create a new user application inside a PetaLinux project:

```
[host] $ petalinux-create -t apps --name gpio-dev-mem-test
```

The new application you have created can be found in the `<project-root>/components/apps/gpio-dev-mem-test` directory, where `<project-root>` is `~/emblnx/labs/lab5/ZYBO_petalinux_v2014_2`.

1-4. Copy the `gpio-dev-mem-test` source from the `sources/lab5/gpio-dev-mem-test` directory.

1-4-1. Change to the newly created application directory:

```
[host] $ cd <project-root>/components/apps/gpio-dev-mem-test
```

1-4-2. Replace the existing `gpio-dev-mem-test.c` file with the completed source from the `sources/lab5` directory:

```
[host] $ cp ~/emblnx/sources/lab5/gpio-dev-mem-test/*.c ./
```

1-4-3. Review the `gpio-dev-mem-test.c` source code to see how to access a device from user space.

`/dev/mem` is a virtual file representing the memory map of the whole system. To access the device from user space, you can open `/dev/mem`, and then use `mmap()` to map the device to memory. You can then access the device by using the pointer that points to the mapped memory.

1-5. Select the new application to be included in the build process. The application is not enabled by default.

1-5-1. Make sure that you are in the project directory; i.e., `~/emblinux/labs/lab5/ZYBO_petalinux_v2014_2`.

1-5-2. Launch the rootfs configuration menu by entering the following command:

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

1-5-3. Press the Down Arrow key (↓) to scroll down the menu to **Apps** (1).

1-5-4. Press <**Enter**> to go into the Apps sub-menu.

The new application **gpio-dev-mem-test** is listed in the menu.

1-5-5. Move to the **gpio-dev-mem-test** and click <**Y**> to select the application (2).

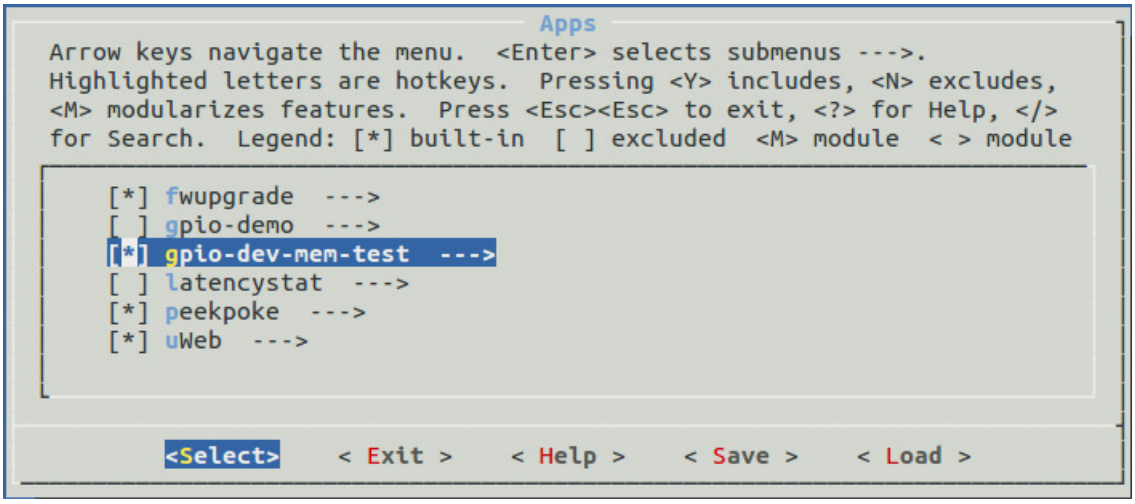

Figure 1. Selecting the gpio-dev-mem-test application

1-5-6. Exit the menu and select <**Yes**> to save the new configuration.

1-6. Build the image.

1-6-1. Enter the following command to build the image:

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

1-7. Copy the `BOOT.BIN` file from the pre-built directory to the SD card.

1-7-1. Make sure that the pre-built BOOT.BIN file is located in the SD card.

1-7-2. If you have done Lab 2, Lab 3, or Lab 4 as your last lab, there is no need to make any changes to the SD card.

1-7-3. If not, copy the BOOT.BIN from the `~/emblnx/labs/lab5/ZYBO_petalinux_v2014_2/pre-built/linux/images` directory to the SD card.

1-8. Run the DHCP server on the host to download the image.

1-8-1. Run the DHCP server:

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

1-9. Power up the board and set the serial port terminal.

1-9-1. Power ON the board.

1-9-2. Make sure that the `/dev/ttyUSB1` is set to read/write access.

```
# sudo chmod 666 /dev/ttyUSB1
```

1-9-3. Start the GtkTerm program.

You may reset the board (BTN7) to see the booting info once again.

1-10. Boot the new embedded Linux image over the network.

1-10-1. Watch the booting process in the GtkTerm window.

1-10-2. Press any key to stop auto-boot when you see message in the GtkTerm window:

1-10-3. If you did not see the “DHCP client bound to address” message during the uboot bootup, you will need to run `dhcp` to obtain the IP address:

```
U-Boot-PetaLinux> dhcp
```

1-10-4. Set the TFTP server IP to the host IP by running the following command in the u-boot console:

```
U-Boot-PetaLinux> set serverip 192.168.1.1
```

1-10-5. Download and boot the new image using TFTP by executing this command in the u-boot console:

```
U-Boot-PetaLinux> run netboot
```

This command will download the `image.ub` from `/tftpboot` on the host to the main memory of the ARM Cortex-A9 MPcore system and boot the system with the image.

1-10-6. Watch the Linux booting in the GtkTerm window.

1-11. Log in and run the `gpio-dev-mem-test` program.

1-11-1. After you log into Linux, we will try the `gpio-dev-mem-test` command to directly access the GPIO devices.

For this example, you will need to know the physical address of the GPIO peripheral. Because all the hardware information is in the DTS (device tree source) file, you can examine the DTS file to obtain the physical address information of the GPIO devices.

1-11-2. Browse to the directory `~/emblnx/labs/lab5/ZYBO_petalinux_v2014_2/subsystems/linux/configs/device-tree`.

1-11-3. Open the `pl.dtsi` file using `gedit`.

1-11-4. Obtain the physical address of the LEDs GPIO by searching for "LEDs_4bits" in the DTS file.

The first argument of the "reg" property of the device node is the physical start address of the device. For example:
`reg = < 0x41220000 0x10000 >;`
"0x41220000" is the physical start address of the LEDs GPIO device and the "0x10000" is the address range.

1-11-5. After you obtain the physical address, you can now run the `gpio-dev-mem-test` command to access the LEDs GPIO.

```
root@ZYBO_petalinux_v2014_2:~# gpio-dev-mem-test
GPIO access through /dev/mem.
GPIO physical address is required.
*argv[0] -g <GPIO_ADDRESS> -i|-o <VALUE>
    -g <GPIO_ADDR>   GPIO physical address
    -i               Input from GPIO
    -o <VALUE>       Output to GPIO
```

Figure 2. Running gpio-dev-mem-test

The Xilinx GPIO peripheral has two registers:

- offset 0x0 is the data register used to read or write the GPIO ports.
- offset 0x4 is the direction register, which is used to indicate whether each GPIO bit is to be configured as an input (corresponding direction bit is '1') or an output (direction bit is '0').

For example, assuming that the LEDs GPIO's physical start address is 0x41220000, to write to the LED GPIO, you need to:

```
# gpio-dev-mem-test -g 0x41220000 -o 0
# gpio-dev-mem-test -g 0x41220000 -o 15
```

Assuming the GPIO's physical start address of the DIP switches is 0x41200000 and Push buttons is 0x41210000, to read from the GPIO of the Push buttons and DIP switches:

```
# gpio-dev-mem-test -g 0x41200000 -i
```

```
# gpio-dev-mem-test -g 0x41210000 -i

root@ZYBO_petalinux_v2014_2:~# gpio-dev-mem-test  -g 0x41220000 -o 0
GPIO access through /dev/mem.
root@ZYBO_petalinux_v2014_2:~# gpio-dev-mem-test  -g 0x41220000 -o 15
GPIO access through /dev/mem.
root@ZYBO_petalinux_v2014_2:~# gpio-dev-mem-test  -g 0x41200000 -i
GPIO access through /dev/mem.
gpio dev-mem test: input: 0000000d
root@ZYBO_petalinux_v2014_2:~# gpio-dev-mem-test  -g 0x41210000 -i
GPIO access through /dev/mem.
gpio dev-mem test: input: 00000002
```

Figure 3. Running gpio-dev-mem-test for GPIO peripherals

1-11-6. Adjust the GPIO's physical start address based on your actual system.

You can also try reading from the GPIOs that connects to the pushbuttons—just use the appropriate physical address obtained from the DTS file.

Experimenting with the UIO Driver Step 2

In this section, you will create a UIO driver. In this example, you are using the enhanced, generic UIO framework, which requires no custom kernel code at all.

2-1. Create a new user application to access GPIO devices from user space.

2-1-1. Make sure that you are in the Petalinux project directory; i.e., `~/emblnx/labs/lab5/ZYBO_petalinux_v2014_2`.

2-1-2. Enter the following command to create a new user application inside a PetaLinux project:

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

The new application you have created can be found in the `<project-root>/components/apps/gpio-uio-test` directory, where `<project-root>` is `~/emblnx/labs/lab5/ZYBO_petalinux_v2014_2`.

2-2. Copy the `gpio-uio-test` source from the `sources/lab5/gpio-uio-test` directory.

2-2-1. Change to the newly created application directory.

```
[host] $ cd <project-root>/components/apps/gpio-uio-test
```

2-2-2. Replace the existing `gpio-uio-test.c` with the completed source from the `sources` directory.

```
[host] $ cp ~/emblnx/sources/lab5/gpio-uio-test/*.c ./
```

2-2-3. Review the `gpio-uio-test.c` file to see how to access device from user space.

A UIO device is presented as `/dev/uioX` in the file system. To access the device through UIO, you can open `/dev/uioX`, 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.

2-3. Select the new application to be included in the build process. The application is not enabled by default.

2-3-1. Make sure your into project directory; i.e.., `~/emblnx/labs/lab5/ZYBO_petalinux_v2014_2`.

2-3-2. Launch the rootfs configuration menu by entering the following command:

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

2-3-3. Press the Down Arrow key (↓) to scroll down the menu to **Apps**.

2-3-4. Press <**Enter**> to go into the Apps sub-menu.

2-3-5. The new application **gpio-uio-test** is listed in the menu.

2-3-6. Move to **gpio-uio-test** and click <**Y**> to select the application.

2-3-7. Exit the menu and select **<Yes>** to save the new configuration.

It will take a few seconds for the configuration change to be applied. Wait until you return to the shell prompt on the command console.

Configuring the Kernel to Support UIO — Step 3

Although, you can implement the control logic of the devices totally in user space, you need to enable the UIO framework inthe kernel. You will configure the UIO subsystem to be built as a loadable module. However, it is also possible to build it directly into the kernel if you prefer.

3-1. Configure the kernel to support UIO.

3-1-1. Make sure that you are in the project directory.

3-1-2. Enter the following command in the terminal.

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

3-1-3. Select **Device Drivers** from the Linux Kernel Configuration menu.

3-1-4. In the Device Drivers menu, scroll down to **Userspace I/O drivers** and select it as **"<M>"**:

```
<M> Userspace I/O drivers  --->
```

Because the kernel is configured to support loadable modules by default, for those loadable device drivers, you can select it as built-in 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 can be either selected as built-in or module. You select it as a module here to experiment with loading a module in Linux later in this lab.

3-1-5. Go into the Userspace I/O Drivers menu and select **Userspace I/O platform driver with generic IRQ handling** as a module:

```
--- Userspace I/O drivers
<M>   Userspace I/O platform driver with generic IRQ handling
<M>   Userspace I/O platform driver with generic irq and dynamic memory
```

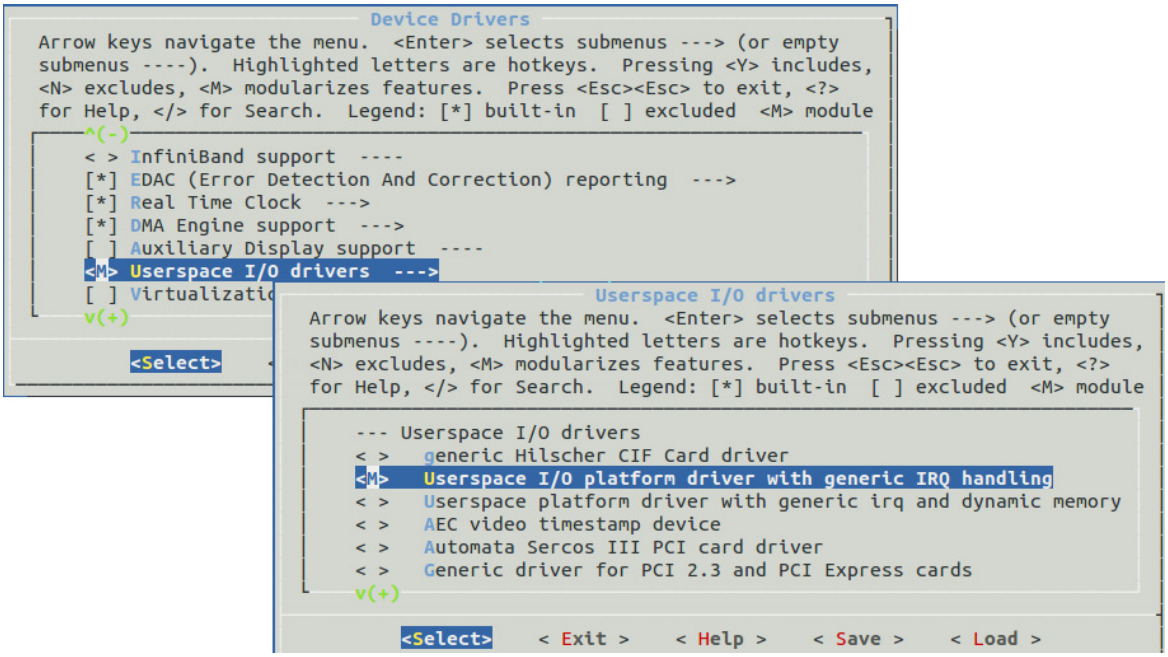

Figure 4. Selecting UserIO drivers as module

3-1-6. Exit the kernel and save the configuration changes.

Identifying the Device to be Controlled by UIO — Step 4

As you have learned in Lab 2, the `compatible` property on a device entry in the device tree (DTS) links the device to a kernel driver. You are going to mark the LEDs GPIO to be controlled as the UIO device, instead of the normal Xilinx GPIO device driver.

4-1. Mark the LEDs GPIO to be controlled as the UIO device.

4-1-1. Change to the directory `~/emblnx/labs/lab5/ZYBO_petalinux_v2014_2/subsystems/linux/configs/device-tree`.

4-1-2. Review the `pl.dtsi` file.

You can open the `pl.dtsi` file by using `gedit`.

4-1-3. You will find the entries for the GPIO peripherals BTNs_4bits, LEDs_4Bits, and SWs_4Bits. They have “xlnx, xps-gpio-1.00.a” as the compatible string. This string indicates that the driver being used is from Xilinx and the driver used is xps-gpio-1.00.a.

4-1-4. You are going to use the UIO driver and not the GPIO driver for this device; thus, you need to replace the `compatible` parameter line with `compatible="generic-uio"`. You will do this in the system-top.dts file.

4-1-5. Open the `system-top.dts` file in the gedit editor and add the marked lines.

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

&clkc {
        ps-clk-frequency = <50000000>;
};

&flash0 {
   compatible = "s25fl128s1";
};

&ps7_usb_0 {
        dr_mode = "otg";
} ;

&ps7_ethernet_0 {
        phy-handle = <&phy0>;
        mdio {
                #address-cells = <1>;
                #size-cells = <0>;
                phy0: phy@1 {
                        compatible = "realtek,RTL8211E";
                        device_type = "ethernet-phy";
                        reg = <1>;
                } ;
        } ;
} ;

&BTNs_4Bits {
        compatible = "generic-uio";
} ;

&LEDs_4Bits {
        compatible = "generic-uio";
} ;

&SWs_4Bits {
        compatible = "generic-uio";
} ;
```

Figure 5. Assigning generic-uio driver to the PL peripherals.

4-1-6. Save the file and close the editor.

Rebuilding Linux and Testing UIO Step 5

You have configured the kernel, created the UIO driver, and made a UIO device node. Now, you need to rebuild the Linux image and test the UIO.

5-1. Rebuild the Linux image.

5-1-1. Change to the PetaLinux project directory `~/emblnx/labs/lab5/ZYBO_petalinux_v2014_2`.

5-1-2. Enter the following command to build the image:

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

5-2. Power up the board and set the serial port terminal.

5-2-1. Power ON the board.

5-2-2. Make sure that `/dev/ttyUSB1` is set to read/write access:

```
# sudo chmod 666 /dev/ttyUSB1
```

5-2-3. Start the GtkTerm program.

You can reset the board (BTN7) to see the booting info once again.

5-3. Boot the new embedded Linux image over the network.

5-3-1. Watch the booting process in the GtkTerm window.

5-3-2. Press any key to stop auto-boot in the GtkTerm window:

5-3-3. If you did not see the “DHCP client bound to address” message during the uboot bootup, you will need to run `dhcp` to obtain the IP address:

```
U-Boot-PetaLinux> dhcp
```

5-3-4. Set the TFTP server IP to the host IP by running the following command in the u-boot console:

```
U-Boot-PetaLinux> set serverip 192.168.1.1
```

5-3-5. Download and boot the new image using TFTP by executing this command in the u-boot console:

```
U-Boot-PetaLinux> run netboot
```

This command will download the `image.ub` from `/tftpboot` on the host to the main memory of the ARM Cortex-A9 MPcore system and boot the system with the image.

5-3-6. Watch the Linux booting in the GtkTerm window.

5-4. Test the UIO.

5-4-1. Log into the system.

5-4-2. Load the UIO modules because you have made the UIO as modules when you configured the kernel previously:

```
# modprobe uio
# modprobe uio_pdrv_genirq
```

5-4-3. List the active, loaded modules with the `lsmod` command:

```
# lsmod
```

```
Built with PetaLinux v2014.2 (Yocto 1.6) ZYBO_petalinux_v2014_2 /dev/ttyPS0
ZYBO_petalinux_v2014_2 login: root
Password:
login[791]: root login on 'ttyPS0'
root@ZYBO_petalinux_v2014_2:~# modprobe uio
root@ZYBO_petalinux_v2014_2:~# modprobe uio_pdrv_genirq
root@ZYBO_petalinux_v2014_2:~# lsmod
    Not tainted
uio_pdrv_genirq 3432 0 - Live 0xbf081000
uio 9315 1 uio_pdrv_genirq, Live 0xbf07a000
ipv6 391307 13 [permanent], Live 0xbf000000
```

Figure 6. Entering the modprobe command

You can find the information of the loaded UIO modules from this place:

```
# ls /sys/class/uio/
uio0 uio1 uio2
```

The name of the UIO and the address information of the UIO can be found in `/sys/class/uio/uioX`.

5-4-4. Run `mdev -s` to make sure that `/dev/uioX` correctly represents the UIO device:

```
# mdev -s
```

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

5-4-5. Try the `gpio-uio-test` command to directly access the LEDs GPIO device.

For example, write 15 to the LEDs GPIO.

```
# gpio-uio-test -d /dev/uio1 -o 15
```

5-4-6. Read from the DIP switches GPIO:

```
# gpio-uio-test -d /dev/uio2 -i
```

```
root@ZYBO_petalinux_v2014_2:~# ls /sys/class/uio/
uio0  uio1  uio2
root@ZYBO_petalinux_v2014_2:~# mdev -s
root@ZYBO_petalinux_v2014_2:~# gpio-uio-test -d /dev/uio1 -o 15
GPIO UIO test.
root@ZYBO_petalinux_v2014_2:~# gpio-uio-test -d /dev/uio2 -i
GPIO UIO test.
gpio-uio-test: input: 0000000d
```

Figure 7. Using gpio-uio-test to turn on all LEDs

5-4-7. When done, power OFF the board and close the GtkTerm window.

Conclusion

In this lab, you have learned how to:

- Access devices from user space
- Create a UIO driver
- Load a module
- Find module information

Completed Solution

If you want to run the solution then copy **BOOT.bin** from the *labsolution\lab5\SDCard* directory onto a SD card. Place the SD card in the Zybo. Set Zybo in the SD Card boot mode. Connect the Zybo to the host machine using Ethernet cable.

Run the following command to start DHCP server on the host:

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

Copy the **image.ub** file from the *labsolution\lab5\tftpboot* directory into */tftpboot* directory.

Power ON the board. Set the terminal session. Interrupt the boot process when autoboot message is shown. Set the serverip address using the following command in the target board terminal window:

```
#set serverip 192.168.1.1
```

Run the netboot command:

```
#run netboot
```

Login into the system and test the lab.