Lab1

Using the Raspberry Pi 3 – Kernel Module and Interrupt Service Routine

Objective

In this part, you will learn how to access the General-Purpose Input/Output (GPIO) ports on the RPi3 through a Kernel Module.

Prelab

- Before coming to this lab, look up and write about the purpose and the arguments of the following functions:

gpio_is_valid, gpio_request, gpio_direction_output, gpio_direction_input, gpio_set_value, gpio_free, gpio_to_irq, request_irq, free_irq, enable_irq, and disable_irq

- Write about the use of printk and dmesg

Background

A **kernel module** is a code that can be added to the kernel at run-time. It can then be removed, as well. Modules extend the kernel's functionality while the system is up and running. That is, we do not need to rebuild an entire kernel to add new functions or build a large kernel that contains all functionality. There are different types of modules, including device drivers. You can find a sample code on how a write a kernel module at this link.

Check the file **Lab1.c**. It has the libraries you will need and the main structure of your lab programs.

WEEK-1

Lab Procedure

1. Turn ON/OFF LEDs using the Kernel Module

You will create a kernel module that turns on the four LEDs of the auxiliary board sequentially several times (ex., Two times) when the kernel is installed.

- In Kernel space, you cannot access the *wiringPi* library. However, recently coding in the kernel space became similar to the wiringPi library, as you do not have to access and configure the Raspberry Pi's GPIO through registers.
- Something to note about kernel modules, they DO NOT contain a main(). Instead, they contain two functions:

```
int init_module(void)

void  cleanup_module(void)
```

The init module function contains code that is run when the module is installed. This function should return 0 unless an error has occurred. cleanup_module is the code that is run when the module is removed.

Alternatively, you can name your functions differently, i.e.,

```
int when_installed(void) void
when_removed(void)
```

Your code should contain the lines:

```
// function runs when module is installed module_init(when_installed);

// function runs when module is removed module_exit(when_removed);
```

- To avoid warnings when installing the module, add the following line to your source code MODULE LICENSE("GPL");

  Important: a Kernel module is compiled into object code but not linked into a complete executable. Please, find the provided Makefile to compile a Kernel module.

- There are three helpful shell commands to use when dealing with modules:

1. lsmod: lists all of the modules currently installed in the kernel
2. insmod NAME.ko: installs the module whose filename is NAME.ko

3. rmmod NAME: removes the module with name: NAME (notice the .ko is not included)
4. modinfo NAME.ko: Can get the information about the loaded module

You need **sudo** to use 2 and 3.

- In order to debug your module code, you cannot use the printf() function. Instead, you can use the printk function and then check the printed lines using dmesg command in the terminal. Modify your kernel module so that the message "MODULE INSTALLED" is printed when the module is installed, and the message "MODULE REMOVED" is printed when the module is removed. Install and remove your module several times. What do you see when you run the *dmesg* command?

WEEK-2

Objective

In the second part of the lab, you will learn how to trigger an interrupt through a push button. Also, write an interrupt service routine (ISR) to handle the interrupt.

Lab Procedure

Detect Push-Button using an Interrupt

You will add an *Interrupt Service Routine (ISR)* to your modules from Week-1. The purpose of the *ISR* is to handle one of five events, which correspond to the five push buttons on the auxiliary board. Pushing any of the buttons causes an interrupt. The handler will change the state of the LEDs based on the button pressed. You will associate the first four buttons with the four LEDs on the board, respectively, to turn them on individually. And link the last button to turn OFF all the LEDs in one go.

Note *WiringPi* provides a way to configure GPIO interrupts in user space on the RPi. You may explore that approach for your future work.

References

- Wiring Pi
- Raspberry Pi GPIO Pinout