GPP Device Drivers & I/O Testing

Lorenzo Javier (005979623)
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 95192
E-mail: loj90@sbcglobal.net

Abstract

General Purpose Input/Output (GPIO) pins/registers are a critical part of embedded systems for controlling peripheral devices. This paper will discuss the implementation of a kernel module to drive an LED's power level, which are typically connected to unique pin headers called GPIOs.

1. Introduction

The embedded system used for this assignment was a Raspberry Pi 1 Model B+, which is based off an ARM architecture. It has integrated GPIO pins which are configurable for custom kernel drivers, and certain user applications that can set the "power" level of an LED.

2. Code Design

In order to control a specific GPIO pin from the kernel module level, the specific code design must first initialize, or request it into memory just once. By doing so, it can read write, and control the new device at any time during the kernel's operations.

2.1. Kernel Module

This low level module has to reference the physical hardware registers in order to control the GPIO pins. The Linux GPIO library was used in this instance by defining the GPIO pin connected to the LED so that it could be referenced in memory. The physical Broadcom GPIO pin number was 21, and with that, a char device was created upon initialization, setting the direction as an output.

If this kernel module needed to be uninstalled, then the GPIO pin and LED level reference would all have to be "released" so that future kernel modules could lock and use these resources.

2.2. User Application

Building the user application does require the kernel module to be installed and running first. Once the char device was created, the user application can open (access) the device through, i.e. /dev/led0, which is much simpler. From there, a user could use the function, ioctl, to immediately set the output levels of the LED driver. [1]

3. Implementation

First, the kernel module must be compiled using the cross-compile toolchain for the ARM platform. Secondly, the user application must be built on top of the kernel module with the appropriate operation calls to interact with the main kernel module to drive the devices.

3.1. Kernel Module

From the main directory of the kernel module, the following commands should be executed:

1. sudo make
2. sudo insmod led.ko
3. sudo rmmod led (*Removing the led module)

For debugging, run either command while running the kernel module to see any printk messages:

1. dmesg
2. tail –f /var/log/kern.log &

3.2. User Application

From the main directory of the user application, the following commands should be executed:

1. sudo make
2. sudo ./led_test

This particular LED driver test runs in a while loop, allowing the user to switch "on" and "off" the LED power level.

4. Testing and Verification

The main goal of this assignment was to be able to control an LED driver device by sending "HIGH" and "LOW" signals to turn it "ON" and "OFF". It was successful by following the below commands to test both the kernel module and user application.

4.1. Kernel Module

From the main directory of the kernel module, the following commands should be executed:

1. sudo make
2. sudo insmod led.ko
3. sudo rmmod led (*Removing the led module)

For debugging, run either command while running the kernel module to see any printk messages:

1. dmesg
2. tail –f /var/log/kern.log &

4.2. User Application

From the main directory of the user application, the following commands should be executed:

1. sudo make
2. sudo ./led_test

This particular LED driver test runs in a while loop so the user can switch "ON" and "OFF" the LED power level. In

this instance, ON = 1 and OFF = 0, and by sending the numeric values in any combination allows the user to directly interact with the LED.

5. Conclusion

By understanding the CPU's datasheet, where all the physical addresses are listed out, any number of external peripheral devices can be attached to the physical GPIO pins and controlled through software, by referencing them in virtual addresses. By using the kernel modules, any user doesn't need to know the low level specifications to design software for any physical peripheral.

Using a development kit allows any user to work on a project with specific hardware and software before designing a unique integrated embedded system device.