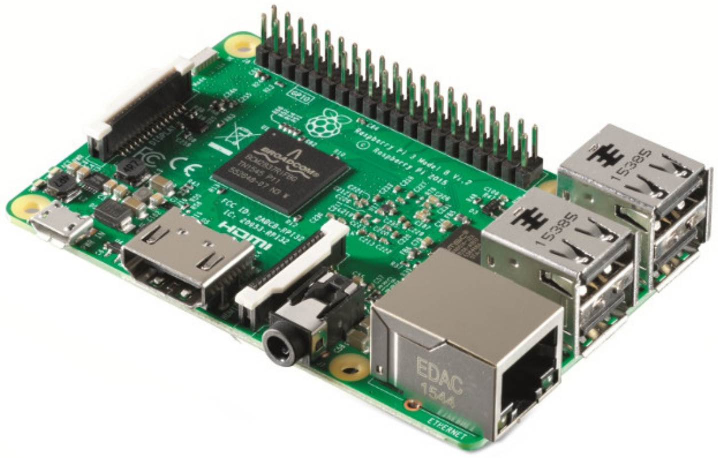
Lab 1 - Using Raspberry Pi 3 GPIO

Week 2 – Kernel Modules



ECE 4220/7220

Real Time Embedded Computing

Fall 2019

University of Missouri

What is a Kernel Module?

- Code that can be loaded and unloaded into the kernel.
- Extend the functionality of the kernel.
  - We do not have to rebuild an entire kernel to add new functions.
  - We do not have to build a large kernel that contains all functionality.
- Does not require the system to reboot after load.

Kernel Module Functions

- As root user you can install, uninstall, and list kernel modules.
  - lsmod – list all the modules currently installed
  - insmod mod_name.ko - installs the specified module into the kernel
  - rmmod mod_name – removes the specified mod from the kernel (notice there is no '.ko')

NOTE: You must use **sudo** in front of both the insmod and rmmod commands.

Writing a Module

- Modules differ from applications.
- Instead of having an 'int main(void)' section, modules have two sections:
  - int init_module(void)
    - Runs when the module is installed.
    - Must return 0 if there are no errors.
  - void cleanup_module(void)
    - Runs when the module is uninstalled.

Example Module Code

```
#ifndef MODULE
#define MODULE
#endif

#ifndef __KERNEL__
#define __KERNEL__
#endif

#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("GPL");

int init_module(void)
{
 // your code here
 return 0;
}

void cleanup_module(void)
{
 // your code here
}
```

Building a Module and Using I/O in a Module

- Modules are .c files that are compiled into object files (.ko)
- void* ioremap(unsigned long phys_addr, unsigned long size)
  - phys_addr – beginning of physical address range
  - size – size of physical address range
- By using this function we are returned a virtual address which we can use just as in our applications.
- Example:
  - ptr = (unsigned long *) ioremap(0x3F200000, 4096);

Bit Masking

- Many times we will want to change one I/O pin.
- This is difficult since we must write to all bits of the port.
- Use **bit masking** to change the state of the pin and **leave all others unchanged**.
- Bitwise operations: AND, OR, XOR.
- These can be used to set, clear, and toggle pins.

Turn on RED LED using Bit Masking

- We want to turn on the red LED of the small auxiliary board.
- Get a pointer to the Selection Register is named GPSET0.
- The bit of GPSET0 that is associated with the red LED is GPIO 2 which corresponds to bit 3.
- To turn on the light we want to set the bit to 1

  *GPSET0 = * GPSET0 | 0x04; or * GPSET0 |= 0x04;

  Where 0x04 = 0000 0100

  Bit 7 Bit 5 Bit 0
- The address of GPSET0 and other registers is could be found in Chapter-6 in “BCM2837 ARM Peripherals Manual”

printk() Command

- We can use printk() command as an alternative for printf() in a kernel module.
- Despite what you might think, printk() is not meant to communicate information to the user, even though we used it for exactly this purpose in our program! It happens to be a logging mechanism for the kernel, and is used to log information or give warnings.
- It doesn’t print on the screen. You can check the printed line using “dmesg” command in the terminal.

Setting Up Eclipse for Kernels

1. Create a new project for Lab 1 Week 2

2. *Right click project folder -> Properties -> C/C++ Build -> Uncheck "Generate Makefiles Automatically" -> Click Apply*

3. Download **Makefile** from canvas under "*Other Materials*"

4. The files should be in the Downloads directory, move into that directory using: **cd Downloads**

5. Copy the file to the directory of your **.c** file.

   - **rsync -av Makefile pi@PI_IP:/home/pi/workspace/PROJECT_NAME/src/**

Helpful Information

- System calls that we will use: (Chapter 9, Section 4 in http://www.makelinux.net/ldd3/)
  - ioremap()
  - iowrite32()
  - ioread32()
- “**BCM2837 ARM Peripherals Manual**” is available on Canvas under “*Other Materials*”
- “**General notes for Raspberry Pi 3 development**” is also available on Canvas under “*Other Materials*”