FALL 2023
MOBILE AND EMBEDDED SYSTEM DESIGN AND APPLICATION
(COMP/ELEC 424/553)

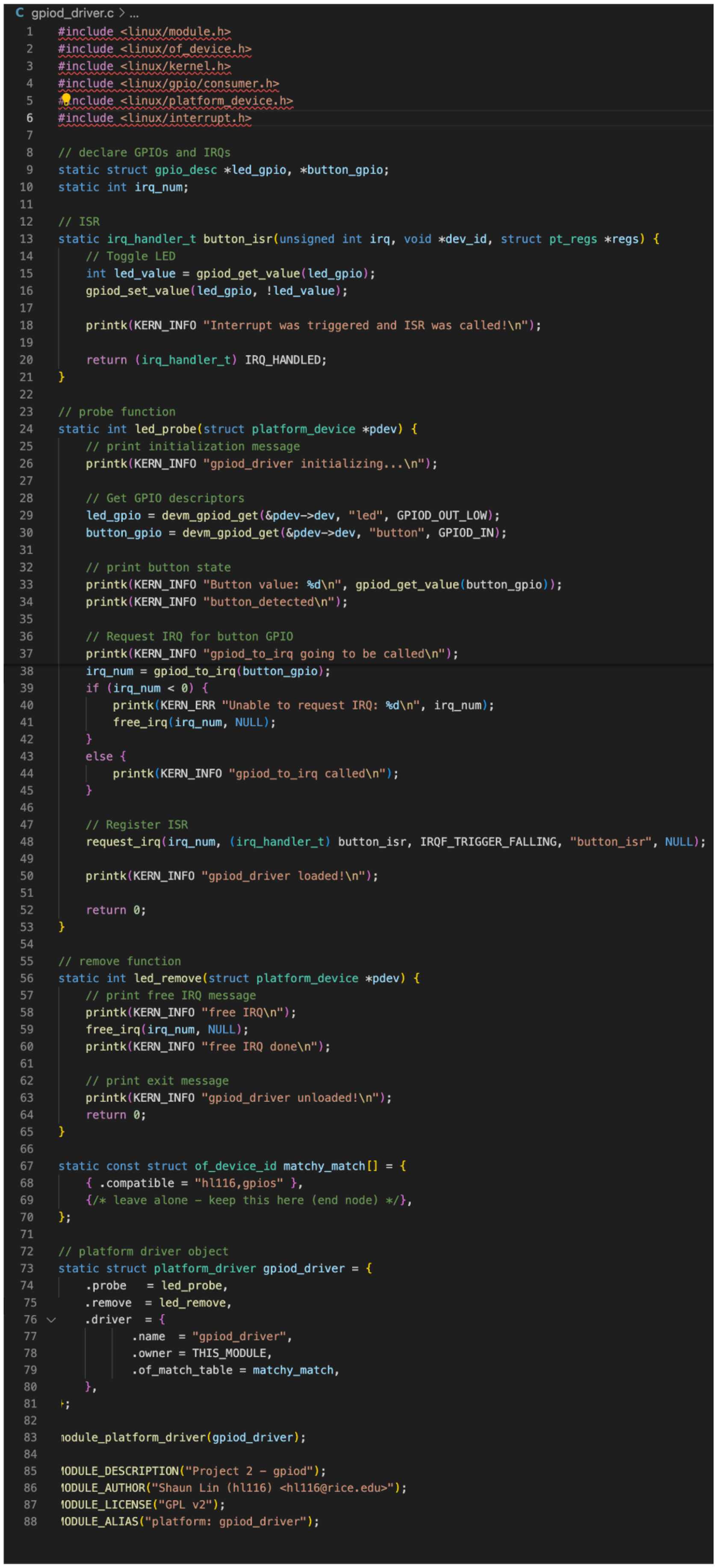
Project 2: gpiod (the dark side)

Shaun Lin (hl116), S01435165

The goal of this project was to develop a custom kernel driver module using the gpiod library to control an LED and button on a Raspberry Pi. I defined the led and button descriptors and ISR in my gpiod_driver.c module. The probe function initializes the GPIOs, prints button state, requests the IRQ, and registers the interrupt handler. The ISR toggles the LED when the button is pressed. Finally, the remove function frees the IRQ. I wrote the device tree overlay defined in the .dts file. I also modified the config.txt file to include the overlay. Overall, I took around 6 hours to complete the project. Through this project I gained experience with Linux driver development, GPIO/IRQ APIs, and device tree overlays.

My first challenge was trying to understand the structure of device tree overlay. Initially I didn’t properly define the device tree overlays, so I couldn’t compile it. I also struggled with debouncing, as I kept getting return value: -524 errors until Dr. Joseph Young announced that the section could be skipped. The biggest bug was using "led" instead of "led-gpios" in the devm_gpiod_get() call, which I eventually realized after my friend gave me some hints. In this project, I learned how to develop custom Linux drivers and work with GPIOs/IRQs from kernel space. Based on this project, I think we can write a motor driver using the gpiod library, and the driver could control RC car motors by setting GPIOs based on speed/direction values. The challenge would be PWM for variable speed and integrating with other sensors or using image process technology to control the RC car.

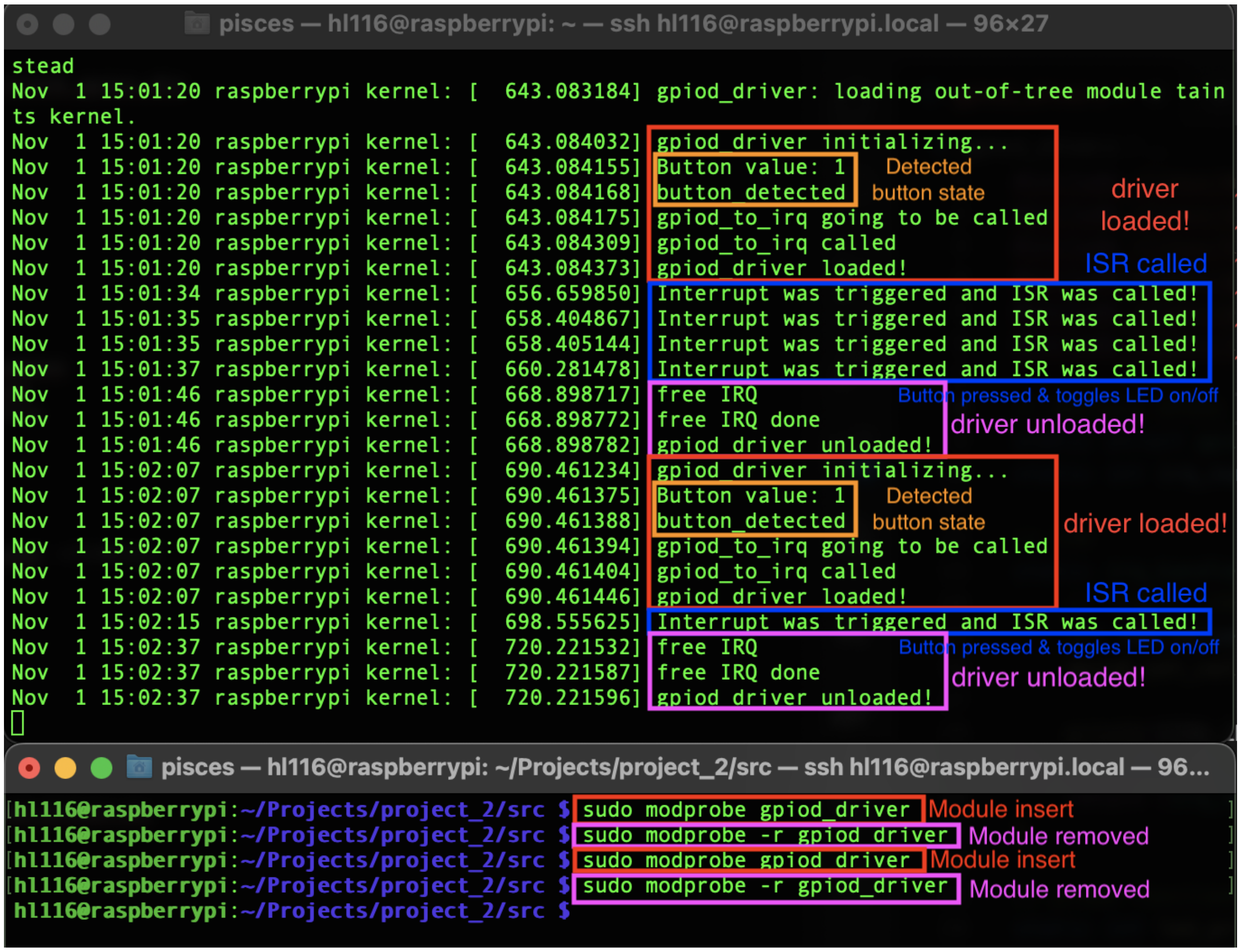
(5 points) Include a screenshot showing a significant portion or all of your driver code.



```c
#include <linux/module.h>
#include <linux/of_device.h>
#include <linux/kernel.h>
#include <linux/gpio/consumer.h>
#include <linux/platform_device.h>
#include <linux/interrupt.h>

// declare GPIOs and IRQs
static struct gpio_desc *led_gpio, *button_gpio;
static int irq_num;

// ISR
static irq_handler_t button_isr(unsigned int irq, void *dev_id, struct pt_regs *regs) {
    // Toggle LED
    int led_value = gpiod_get_value(led_gpio);
    gpiod_set_value(led_gpio, !led_value);

    printk(KERN_INFO "Interrupt was triggered and ISR was called!\n");

    return (irq_handler_t) IRQ_HANDLED;
}

// probe function
static int led_probe(struct platform_device *pdev) {
    // print initialization message
    printk(KERN_INFO "gpiod_driver initializing...\n");

    // Get GPIO descriptors
    led_gpio = devm_gpiod_get(&pdev->dev, "led", GPIOD_OUT_LOW);
    button_gpio = devm_gpiod_get(&pdev->dev, "button", GPIOD_IN);

    // print button state
    printk(KERN_INFO "Button value: %d\n", gpiod_get_value(button_gpio));
    printk(KERN_INFO "button_detected\n");

    // Request IRQ for button GPIO
    printk(KERN_INFO "gpiod_to_irq going to be called\n");
    irq_num = gpiod_to_irq(button_gpio);
    if (irq_num < 0) {
        printk(KERN_ERR "Unable to request IRQ: %d\n", irq_num);
        free_irq(irq_num, NULL);
    }
    else {
        printk(KERN_INFO "gpiod_to_irq called\n");
    }

    // Register ISR
    request_irq(irq_num, (irq_handler_t) button_isr, IRQF_TRIGGER_FALLING, "button_isr", NULL);

    printk(KERN_INFO "gpiod_driver loaded!\n");

    return 0;
}

// remove function
static int led_remove(struct platform_device *pdev) {
    // print free IRQ message
    printk(KERN_INFO "free IRQ\n");
    free_irq(irq_num, NULL);
    printk(KERN_INFO "free IRQ done\n");

    // print exit message
    printk(KERN_INFO "gpiod_driver unloaded!\n");
    return 0;
}

static const struct of_device_id matchy_match[] = {
    { .compatible = "hl116,gpios" },
    {/* leave alone - keep this here (end node) */},
};

// platform driver object
static struct platform_driver gpiod_driver = {
    .probe   = led_probe,
    .remove  = led_remove,
    .driver  = {
           .name  = "gpiod_driver",
           .owner = THIS_MODULE,
           .of_match_table = matchy_match,
    },
};

odule_platform_driver(gpiod_driver);

ODULE_DESCRIPTION("Project 2 - gpiod");
ODULE_AUTHOR("Shaun Lin (hl116) <hl116@rice.edu>");
ODULE_LICENSE("GPL v2");
ODULE_ALIAS("platform: gpiod_driver");
```

Figure 1. Screenshot all of my gpiod_driver.c code

(4 points) Include a screenshot of terminal output showing the messages printed by your code.


pisces — hl116@raspberrypi: ~ — ssh hl116@raspberrypi.local — 96×27
Detected button state
driver loaded!
ISR called
Button pressed & toggles LED on/off
driver unloaded!
Detected button state
driver loaded!
ISR called
Button pressed & toggles LED on/off
driver unloaded!
pisces — hl116@raspberrypi: ~/Projects/project_2/src — ssh hl116@raspberrypi.local — 96...
Module insert
Module removed
Module insert
Module removed

```
stead
Nov  1 15:01:20 raspberrypi kernel: [  643.083184] gpiod_driver: loading out-of-tree module tain
ts kernel.
Nov  1 15:01:20 raspberrypi kernel: [  643.084032] gpiod driver initializing...
Nov  1 15:01:20 raspberrypi kernel: [  643.084155] Button value: 1
Nov  1 15:01:20 raspberrypi kernel: [  643.084168] button detected
Nov  1 15:01:20 raspberrypi kernel: [  643.084175] gpiod_to_irq going to be called
Nov  1 15:01:20 raspberrypi kernel: [  643.084309] gpiod_to_irq called
Nov  1 15:01:20 raspberrypi kernel: [  643.084373] gpiod driver loaded!
Nov  1 15:01:34 raspberrypi kernel: [  656.659850] Interrupt was triggered and ISR was called!
Nov  1 15:01:35 raspberrypi kernel: [  658.404867] Interrupt was triggered and ISR was called!
Nov  1 15:01:35 raspberrypi kernel: [  658.405144] Interrupt was triggered and ISR was called!
Nov  1 15:01:37 raspberrypi kernel: [  660.281478] Interrupt was triggered and ISR was called!
Nov  1 15:01:46 raspberrypi kernel: [  668.898717] free IRQ
Nov  1 15:01:46 raspberrypi kernel: [  668.898772] free IRQ done
Nov  1 15:01:46 raspberrypi kernel: [  668.898782] gpiod driver unloaded!
Nov  1 15:02:07 raspberrypi kernel: [  690.461234] gpiod driver initializing...
Nov  1 15:02:07 raspberrypi kernel: [  690.461375] Button value: 1
Nov  1 15:02:07 raspberrypi kernel: [  690.461388] button_detected
Nov  1 15:02:07 raspberrypi kernel: [  690.461394] gpiod_to_irq going to be called
Nov  1 15:02:07 raspberrypi kernel: [  690.461404] gpiod_to_irq called
Nov  1 15:02:07 raspberrypi kernel: [  690.461446] gpiod driver loaded!
Nov  1 15:02:15 raspberrypi kernel: [  698.555625] Interrupt was triggered and ISR was called!
Nov  1 15:02:37 raspberrypi kernel: [  720.221532] free IRQ
Nov  1 15:02:37 raspberrypi kernel: [  720.221587] free IRQ done
Nov  1 15:02:37 raspberrypi kernel: [  720.221596] gpiod driver unloaded!
```

```
hl116@raspberrypi:~/Projects/project_2/src $ sudo modprobe gpiod_driver
hl116@raspberrypi:~/Projects/project_2/src $ sudo modprobe -r gpiod driver
hl116@raspberrypi:~/Projects/project_2/src $ sudo modprobe gpiod driver
hl116@raspberrypi:~/Projects/project_2/src $ sudo modprobe -r gpiod_driver
hl116@raspberrypi:~/Projects/project_2/src $
```

Figure 2. Screenshot of terminal output