Course: Embedded Linux

Course code: EC3321-1

A Mini Project Report *on*

Linux Kernel Module for 4*4 Keypad

Submitted By

NAME	USN
Nikhitha R Aithal	NNM22EC100
Pavitra Poojary	NNM22EC107
Pranamya Acharya	NNM22EC111
Rachana Shenoy	NNM22EC127

Under the guidance of

Dr. Sukesh Rao M

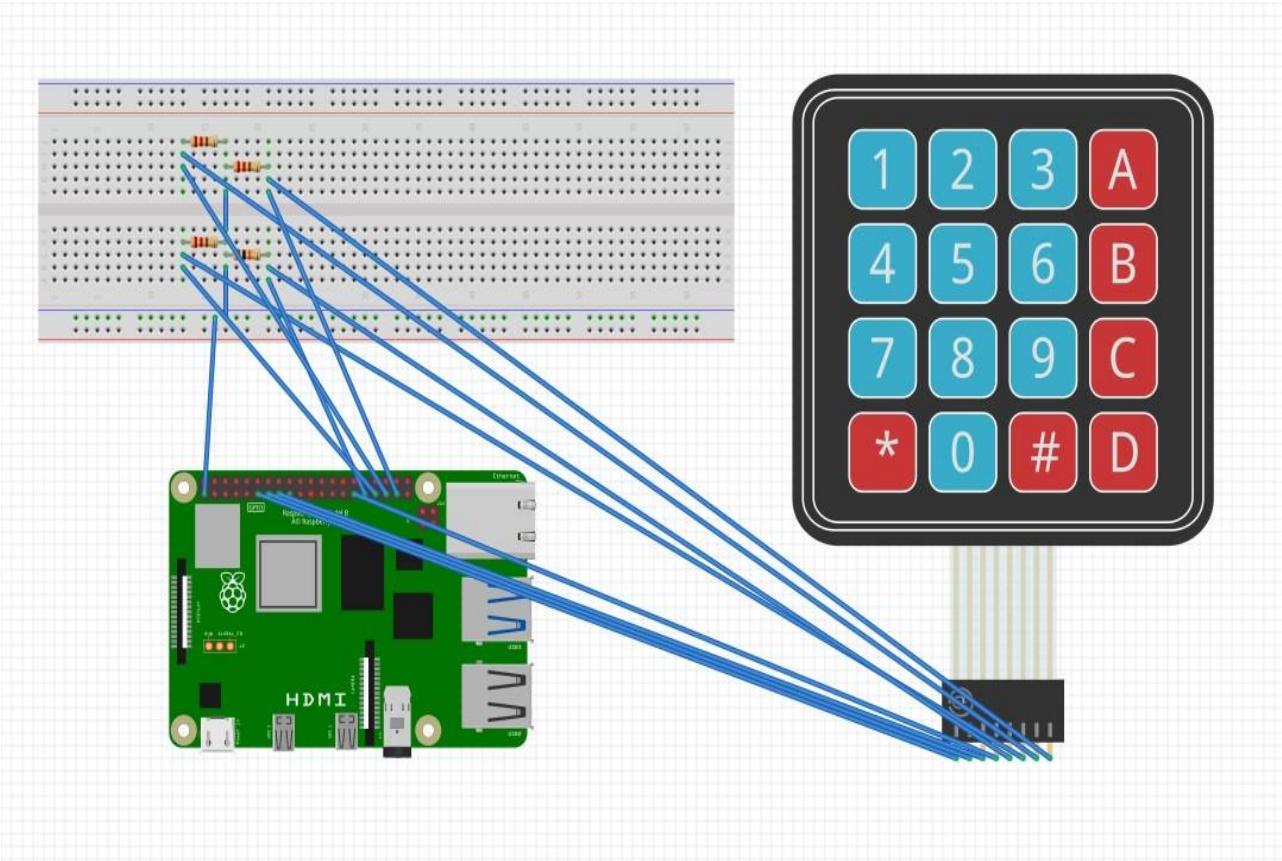
Associate Professor

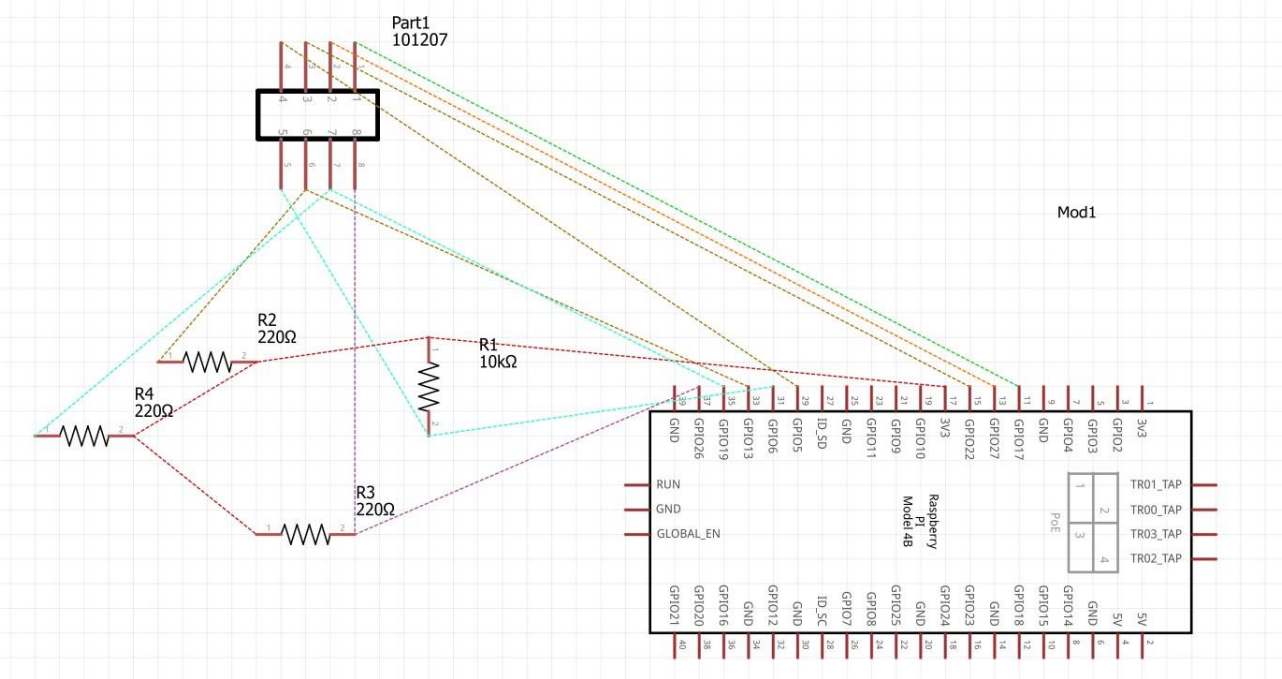
DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

N.M.A.M. INSTITUTE OF TECHNOLOGY, NITTE – 574110

2022 – 2023

CIRCUIT DIAGRAM





APPLICATION FILE

```
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>

#define KEY_PRESSED _IOR('a','a',char*)
#define MAX_TRIALS 5
#define PASSWORD "123"

int main()
{
    int fd;
    char key;
    char input[4] = {0}; // To store 3 characters + null terminator
    int trial = 0;
    int input_index = 0;

    fd = open("/dev/keypad_ioctl", O_RDWR);
    if (fd < 0) {
        perror("Failed to open device");
        return fd;
    }

    while (trial < MAX_TRIALS) {
        printf("Trial %d: Enter 3-character password:\n", trial + 1);
```

```
        input_index = 0;
        memset(input, 0, sizeof(input)); // Clear input buffer

        // Collect 3 characters
        while (input_index < 3) {
            if (ioctl(fd, KEY_PRESSED, &key) < 0) {
                printf("Failed to read key\n");
                close(fd);
                return -1;
            }
            input[input_index++] = key;
            printf("%c", key);
}
printf("\n");
        // Compare with password
        if (strcmp(input, PASSWORD) == 0) {
            printf("Password matched!\n");
            close(fd);
            return 0;
        } else {
            printf("Password not matched\n");
        }
        trial++;
    }
    printf("Trials over. No match found.\n");
    close(fd);
    return 0;
}
```

LKM SOURCE CODE

```
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/ioctl.h>
#include <linux/gpio.h>
#include <linux/delay.h>
#include <linux/init.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/kernel.h>
#include <linux/err.h>

#define DEVICE_NAME "keypad_ioctl"
#define CLASS_NAME  "keypad"

#define KEY_PRESSED _IOR('a','a',char*)//changed to read

static int row_pins[4] = {529, 539, 534, 517}; // GPIO17, GPIO27, GPIO22, GPIO5
static int col_pins[4] = {518, 525, 531, 538};



static char keymap[4][4] = {
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};

char key=0;
```

```
static int major;
static struct class* keypad_class = NULL;
static struct device* keypad_device = NULL;

static void set_key(void)
{
int i;
    for (i = 0; i < 4; i++) {
        gpio_request(row_pins[i], "row_pins");
        gpio_direction_output(row_pins[i],1); // Rows high by default
    }
    for (i = 0; i < 4; i++) {
        gpio_request(col_pins[i], "col_pins");
        gpio_direction_input(col_pins[i]);
    }
}

static char read_key(void)
{
int row, col;

    while (1) {
        for (row = 0; row < 4; row++) {
            // Set all rows high
            for (int i = 0; i < 4; i++) {
                gpio_set_value(row_pins[i], 1);
            }
            // Set current row low
            gpio_set_value(row_pins[row], 0);

            // Check each column
```

```
            for (col = 0; col < 4; col++) {
                if (gpio_get_value(col_pins[col]) == 0) {
                    // Debounce
                    mdelay(50);
                    // Confirm key is still pressed
                    if (gpio_get_value(col_pins[col]) == 0) {
                        // Wait for key release
                        while (gpio_get_value(col_pins[col]) == 0) {
                            mdelay(10);
                        }
                        // Restore row state
                        gpio_set_value(row_pins[row], 1);
                        printk(KERN_INFO "Key detected: row=%d, col=%d, key=%c\n", row,
col, keymap[row][col]);
                        return keymap[row][col];
                    }
                }
            }
            // Restore row state
            gpio_set_value(row_pins[row], 1);
        }
        // Optional: Small delay to avoid CPU hogging
        mdelay(10);
    }
    return 0; //
}
static long  keypad_ioctl(struct file *file, unsigned int cmd, unsigned long  arg )
{
    switch (cmd) {
    case KEY_PRESSED:
        set_key();
        key=read_key();
```

```
int not_copied = copy_to_user((char __user *)arg, &key, sizeof(char));
        if (not_copied) {
           // printk(KERN_ERR "IOCTL: Failed to copy  bytes\n", 0);
            return -EFAULT;
        }
        printk(KERN_INFO "IOCTL: Sent key: %c\n", key);

        break;
    default:
        return -EINVAL;
    }
    return 0;
}

static int keypad_open(struct inode *inode, struct file *file)
{
    return 0;
}

static int keypad_release(struct inode *inode, struct file *file)
{
    return 0;
}

static const struct file_operations fops = {
    .owner          = THIS_MODULE,
    .unlocked_ioctl = keypad_ioctl,
    .open           = keypad_open,
    .release        = keypad_release,
};

static int __init keypad_init(void)
```

```
{
//    int ret;

    // Register char device
    major = register_chrdev(0, DEVICE_NAME, &fops);
    if (major < 0) {
        pr_err("Failed to register device\n");
        return major;
    }

    // Create device class and device node
    keypad_class = class_create(CLASS_NAME);
    if (IS_ERR(keypad_class)) {
        unregister_chrdev(major, DEVICE_NAME);
        return PTR_ERR(keypad_class);
    }

    keypad_device = device_create(keypad_class, NULL, MKDEV(major, 0), NULL,
DEVICE_NAME);
    if (IS_ERR(keypad_device)) {
        class_destroy(keypad_class);
        unregister_chrdev(major, DEVICE_NAME);
        return PTR_ERR(keypad_device);
    }

    return 0;
}

static void __exit keypad_exit(void)
{
    device_destroy(keypad_class, MKDEV(major, 0));
    class_destroy(keypad_class);
```

```
    unregister_chrdev(major, DEVICE_NAME);
        for (int i = 0; i < 4; i++) {
    gpio_free(row_pins[i]);
    gpio_free(col_pins[i]);
}
    pr_info("keypad ioctl module unloaded\n");
}

module_init(keypad_init);
module_exit(keypad_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
MODULE_DESCRIPTION("keypad Control using ioctl");
```

MAKEFILE

```
obj-m := k_code.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

OUTPUT SCREENSHOT