A REPORT ON

GPIO DRIVER FOR TOUCH SENSOR WITH LED

BY

SUNIL REDDY

RAJASEKAR

M.E. EMBEDDED SYSTEMS

Prepared in fulfilment of the

(EEE G547)

DEVICE DRIVERS

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, HYDERABAD

(NOVEMBER 2021)

SUMMARY

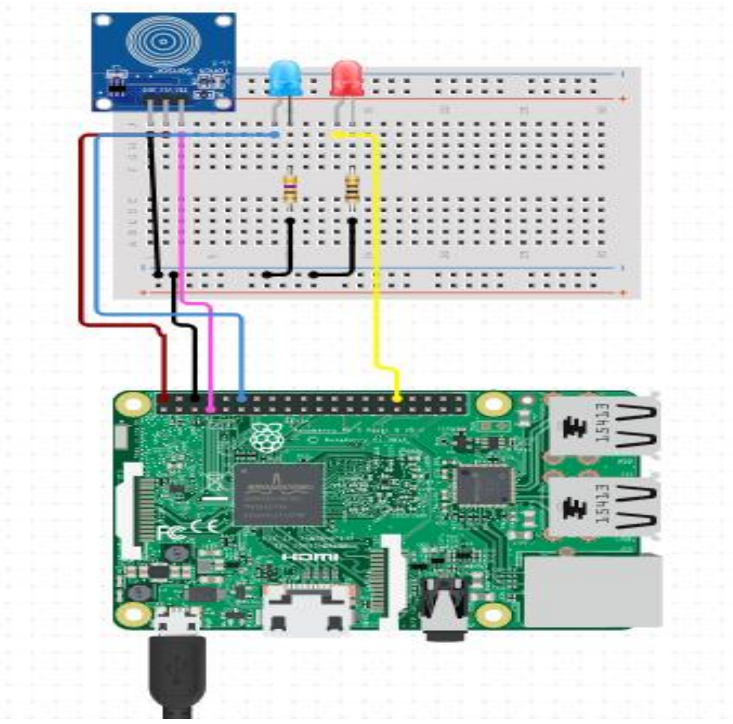
Two LEDs have been connected to the OUTPUT pin (GPIO 26 & GPIO 25) and the touch sensor has been connected to the INPUT pin (GPIO 6).

Whenever touch sensor is detected,

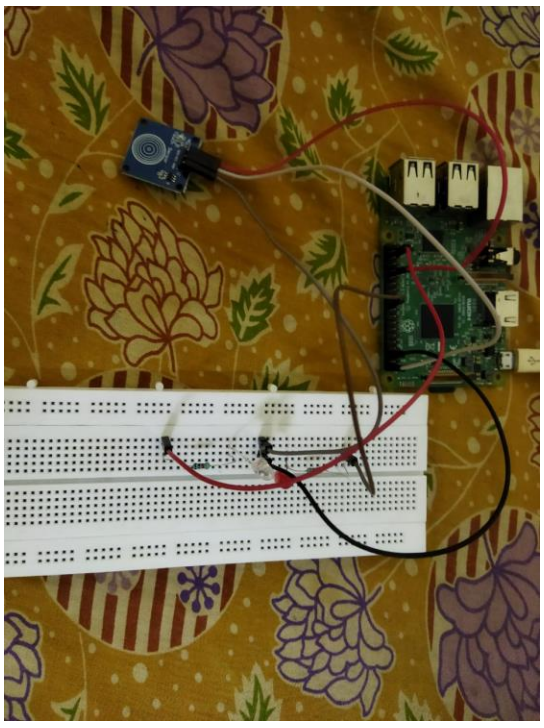
- White LED status will be toggled in kernel space,and status is also displayed in User space.
- Red LED status is configurable from user space.

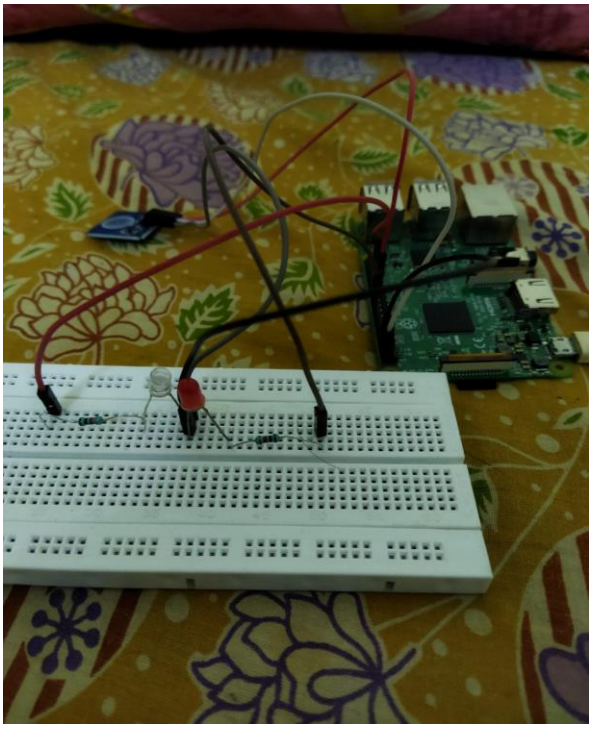
HARDWARE DESIGN

Schematic Diagram



Actual Circuit





KERNEL SPACE DRIVER CODE & BUILD PROCESS

```
/***************************************************************************//**
*  \file       main.c
*
*  \details    Simple GPIO driver interfacing with touch sensor
*
*  \author     Rajasekar  &  Sunilreddy

*******************************************************************************/
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kdev_t.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/uaccess.h>  //copy_to/from_user()
#include <linux/gpio.h>     //GPIO
#include <linux/interrupt.h> // Interrupt handling fuction
#include <linux/device.h>
#include <linux/delay.h>
#include "config.h"

// The below code is used to avoid repetitive interrupt occurence(No debounce support in Raspberrypi)gpio_irq
#define DEBOUNCE

#ifdef DEBOUNCE
#include <linux/jiffies.h>

extern unsigned long volatile jiffies;
unsigned long old_jiffie = 0;
#endif
```

```
 //Sensor's Digital port is connected to this GPIO
 #define GPIO_IN  (6)

 //LED is connected to this GPIO
 #define GPIO_OUT (26)
 #define GPIO_OUT1 (25)

 //GPIO_IN value toggle
 volatile int16_t  led_logic = 0;
 volatile int16_t toggle=0;
 volatile int16_t user_input;
 //This used for storing the IRQ number for the GPIO
 unsigned int GPIO_irqNumber;
 //int8_t toggle=0;

 //Interrupt handler for GPIO 25. This will be called whenever there is a raising edge detected.
 static irqreturn_t gpio_interrupt(int irq,void *dev_id)
{
   static unsigned long flag = 0;

#ifdef DEBOUNCE
    unsigned long diff = jiffies - old_jiffie;
    if (diff < 20)
    {
      return IRQ_HANDLED;
    }

   old_jiffie = jiffies;
#endif
```

```
    local_irq_save(flag);
    toggle=1;
    led_logic = (0x01 ^ led_logic);
                        // toggle the old value
    gpio_set_value(GPIO_OUT, led_logic);                        // toggle the GPIO_OUT

    pr_info("Interrupt Occurred : Output Value : %d ",gpio_get_value(GPIO_OUT));
    local_irq_restore(flag);
    return IRQ_HANDLED;
}
long ioctl_dev(struct file *file, unsigned int ioctl_num, unsigned long ioctl_param)
{

 switch(ioctl_num)
 {
  case LED1_STATUS:
  put_user(led_logic,(int16_t *)ioctl_param);
  break;
  case LED2_INPUT:
  put_user(toggle,(int16_t *)ioctl_param);
  toggle=0;
  break;
  case LED2_USER_INPUT:
  get_user(user_input,(int16_t *)ioctl_param);
  gpio_set_value(GPIO_OUT1, user_input);
  break;
 }
  return 0;
}
```

```
dev_t dev = 0;
static struct class *dev_class;
static struct cdev led_cdev;

static int __init led_driver_init(void);
static void __exit led_driver_exit(void);


/*************** Driver functions **********************/
static int led_open(struct inode *inode, struct file *file);
static int led_release(struct inode *inode, struct file *file);
static ssize_t led_read(struct file *filp,
                        char __user *buf, size_t len,loff_t * off);
static ssize_t led_write(struct file *filp,
                        const char *buf, size_t len, loff_t * off);
/******************************************************/

//File operation structure
static struct file_operations fops =
{
  .owner          = THIS_MODULE,
  .read           = led_read,
  .write          = led_write,
  .open           = led_open,
  .release        = led_release,
  .unlocked_ioctl = ioctl_dev,
};
```

```
/*
** This function will be called when we open the Device file
*/
static int led_open(struct inode *inode, struct file *file)
{
  pr_info("Device File Opened...!!!\n");
  return 0;
}
/*
** This function will be called when we close the Device file
*/
static int led_release(struct inode *inode, struct file *file)
{
  pr_info("Device File Closed...!!!\n");
  return 0;
}
```

```c
/*
** This function will be called when we read the Device file
*/
static ssize_t led_read(struct file *filp,
                        char __user *buf, size_t len, loff_t *off)
{
  uint8_t gpio_logic = 0;
  //reading GPIO value
  gpio_logic = gpio_get_value(GPIO_OUT);
  //write to user
  len = 1;
  if( copy_to_user(buf, &gpio_logic, len) > 0) {
    pr_err("ERROR: Not all the bytes have been copied to user\n");
  }

  pr_info("Read function : GPIO get value = %d \n", gpio_logic);

  return 0;
}
```

```c
/*
** This function will be called when we write the Device file
*/
static ssize_t led_write(struct file *filp,
                         const char __user *buf, size_t len, loff_t *off)
{
  uint8_t led_buf[10] = {0};

  if( copy_from_user( led_buf, buf, len ) > 0) {
    pr_err("ERROR: Not all the bytes have been copied from user\n");
  }

  pr_info("Write Function : GPIO Set value = %c\n", led_buf[0]);

  if (led_buf[0]=='1') {
    //set the GPIO value to HIGH
    gpio_set_value(GPIO_OUT, 1);
  } else if (led_buf[0]=='0') {
    //set the GPIO value to LOW
    gpio_set_value(GPIO_OUT, 0);
  } else {
    pr_err("Unknown command : Please provide either 1 or 0 \n");
  }

  return len;
}
```

```c
/*
** Module Init function
*/
static int __init led_driver_init(void)
{
  /*Allocating Major number*/
  if((alloc_chrdev_region(&dev, 0, 1, "led_Dev")) <0){
    pr_err("Cannot allocate major number\n");
    goto r_unreg;
  }
  pr_info("Major = %d Minor = %d \n",MAJOR(dev), MINOR(dev));

  /*Creating cdev structure*/
  cdev_init(&led_cdev,&fops);

  /*Adding character device to the system*/
  if((cdev_add(&led_cdev,dev,1)) < 0){
    pr_err("Cannot add the device to the system\n");
    goto r_del;
  }

  /*Creating struct class*/
  if((dev_class = class_create(THIS_MODULE,"led_class")) == NULL){
    pr_err("Cannot create the struct class\n");
    goto r_class;
  }
```

```c
  /*Creating device*/
  if((device_create(dev_class,NULL,dev,NULL,"led_Device")) == NULL){
    pr_err( "Cannot create the Device \n");
    goto r_device;
  }

  //Output GPIO configuration
  //Checking the GPIO is valid or not
  if(gpio_is_valid(GPIO_OUT) == false){
    pr_err("GPIO %d is not valid\n", GPIO_OUT);
    goto r_device;
  }
  if(gpio_is_valid(GPIO_OUT1) == false){
    pr_err("GPIO1 %d is not valid\n", GPIO_OUT1);
    goto r_device;
  }

  //Requesting the GPIO
  if(gpio_request(GPIO_OUT,"GPIO_OUT") < 0){
    pr_err("ERROR: GPIO %d request\n", GPIO_OUT);
    goto r_gpio_out;
  }
   if(gpio_request(GPIO_OUT1,"GPIO_OUT1") < 0){
    pr_err("ERROR: GPIO1 %d request\n", GPIO_OUT1);
    goto r_gpio_out;
  }
```

```
    //configure the GPIO as output
    gpio_direction_output(GPIO_OUT, 0);
    gpio_direction_output(GPIO_OUT1, 0);


    //Input GPIO configuratioin
    //Checking the GPIO is valid or not
    if(gpio_is_valid(GPIO_IN) == false){
      pr_err("GPIO %d is not valid\n", GPIO_IN);
      goto r_gpio_in;
    }

    //Requesting the GPIO
    if(gpio_request(GPIO_IN,"GPIO_IN") < 0){
      pr_err("ERROR: GPIO %d request\n", GPIO_IN);
      goto r_gpio_in;
    }

    //configure the GPIO as input
    gpio_direction_input(GPIO_IN);

   // Handles debouce
#ifndef EN_DEBOUNCE
    //Debounce the button with a delay of 200ms
    if(gpio_set_debounce(GPIO_IN, 200) < 0){
      pr_err("ERROR: gpio_set_debounce - %d\n", GPIO_IN);
      //goto r_gpio_in;
    }
#endif
```

```
    //Get the IRQ number for our GPIO
    GPIO_irqNumber = gpio_to_irq(GPIO_IN);
    pr_info("GPIO_irqNumber = %d\n", GPIO_irqNumber);

    if (request_irq(GPIO_irqNumber,             //IRQ number
                    (void *)gpio_interrupt,   //IRQ handler
                    IRQF_TRIGGER_RISING,        //Handler will be called in raising edge
                    "led_Device",               //used to identify the device name using this IRQ
                    NULL)) {                    //device id for shared IRQ
      pr_err("my_device: cannot register IRQ ");
      goto r_gpio_in;
    }



    pr_info("Device Driver Insert...Done!!!\n");
    return 0;

r_gpio_in:
    gpio_free(GPIO_IN);
r_gpio_out:
    gpio_free(GPIO_OUT);
    gpio_free(GPIO_OUT1);
r_device:
    device_destroy(dev_class,dev);
r_class:
    class_destroy(dev_class);
r_del:
    cdev_del(&led_cdev);
r_unreg:
    unregister_chrdev_region(dev,1);
```

```
    return -1;
}

/*
** Module exit function
*/
static void __exit led_driver_exit(void)
{
    free_irq(GPIO_irqNumber,NULL);
    gpio_free(GPIO_IN);
    gpio_free(GPIO_OUT);
    device_destroy(dev_class,dev);
    class_destroy(dev_class);
    cdev_del(&led_cdev);
    unregister_chrdev_region(dev, 1);
    pr_info("Device Driver Remove...Done!!\n");
}

module_init(led_driver_init);
module_exit(led_driver_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Rajasekar&Sunilreddy");
MODULE_DESCRIPTION("GPIO Driver with LED and touch sensor ");
```

Build Process :

- Build the driver by using Makefile (sudo make)
- Load the driver using sudo insmod driver.ko
- Check whether module is inserter in kernel space with lsmod.
- Unload the driver using sudo rmmod driver, after checking LEDs.

USER SPACE APPLICATION CODE & BUILD PROCESS

```c
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<sys/ioctl.h>
#include "config.h"
int file_desc;
int led1_status(int file_desc, int16_t *msg)
{
 int ret_val;
 ret_val = ioctl(file_desc, LED1_STATUS,msg);
 return ret_val;
}
int led2_input(int file_desc, int16_t *msg)
{
 int ret_val;
 ret_val = ioctl(file_desc, LED2_INPUT,msg);
 return ret_val;
}
int main(void)
{
 int ret_val;
 int16_t led1_value,toggle_input,user_input;
 file_desc = open(DEVICE_FILE_NAME,0);
 if(file_desc<0)
 {
  printf("Device Open Failed for %s\n",DEVICE_FILE_NAME);
  exit(-1);
 }
 while(1)
 {
  led2_input(file_desc,&toggle_input);
  if(toggle_input)
  {
  printf("********************************************** \n");
   printf("touch input detected \n");
  led1_status(file_desc,&led1_value);
  printf("WHITE LED status(toggling):%d \n",led1_value);
  printf("ENTER THE RED LED status:");
  scanf("%d",&user_input);
  ioctl(file_desc, LED2_USER_INPUT,(int16_t*)&user_input);
  }
}
}
```

Build Process :

- Compile user application code with gcc -o output user.c
- Run the application (sudo ./output) after inserting kernel driver module.

RESULTS

Initially we have given the touch input which is detected from kernel to userspace then the white led is on and red led status is given as 0 in userspace.
Later we have given the touch input then the white led is off and red led status is given as 1 in userspace which can be seen below.

```
pi@raspberrypi:~/Desktop/test $ gcc -o output user.c
pi@raspberrypi:~/Desktop/test $ sudo ./output
********************************************

touch input detected
WHITE LED status(toggling):1
ENTER THE RED LED status:0
********************************************

touch input detected
WHITE LED status(toggling):0
ENTER THE RED LED status:1
```