A Project Report

On

Driver for GPIO INTERRUPT

BY

Ayilagari Chakradhar Reddy 2020H1400227H

Janjirala Anvesh 2020H1400248H

M.E Embedded Systems

Prepared in fulfilment of

EEE G547 Device Drivers

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)
HYDERABAD CAMPUS

(December 2021)

TABLE OF CONTENTS

SUMMARY

Vibration/SW420 Sensor

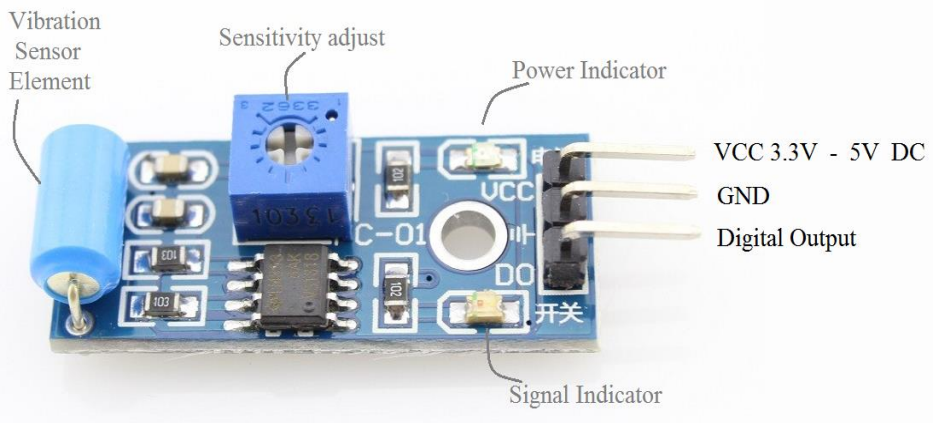


Figure: SW-420 Sensor

- This Vibration Sensor Module consists of an SW-420 Vibration Sensor, resistors, capacitor, potentiometer, comparator LM393, Power, and status LED in an integrated circuit.

- The built-in Comparator detects any vibration beyond the threshold. The threshold can adjust using an onboard potentiometer. During no vibration, the sensor provides Logic Low, and when the vibration is detected, the sensor offers Logic High.

DESIGN SUMMARY

- In this GPIO Interrupt detection system, We considered two GPIOs. One GPIO is configured as input for the sensor connection, and another GPIO is configured as output for LED connection.

- Whenever the SW-420 sensor detects the vibration, it produces the output logic high, which will be detected as an interrupt by the system, and it toggles the LED connected to the GPIO.

Accessing the input GPIO in Linux Kernel

- Identifying the provided GPIO pin value as valid or not.

- On Validation, GPIO request can be done from the Kernel GPIO subsystem. Configure GPIO direction as an input.

- Set the debounce-interval and read the GPIO.

- Enable IRQ and release the GPIO while exiting the driver.

HARDWARE DESIGN

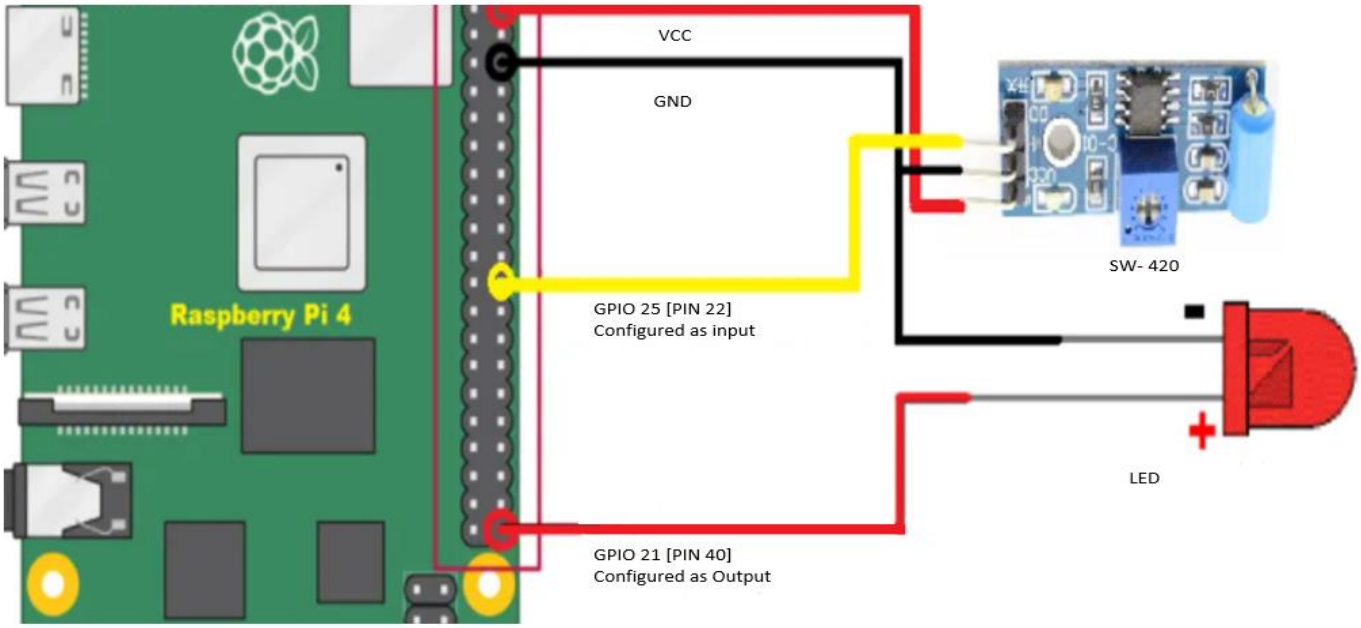


Figure: Schematic of SW-420 Sensor interface with Pi

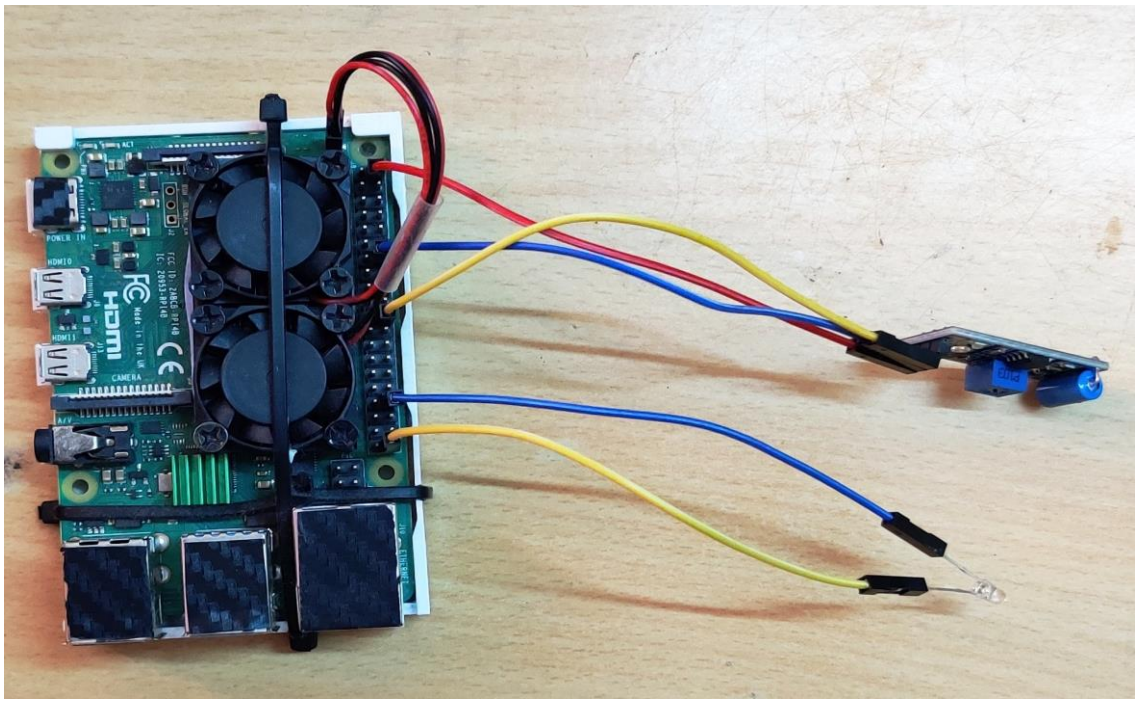


Figure: Actual Design of the system

Key Components:

- Raspberry Pi 4b
- SW-420 Sensor
- LED

HARDWARE

- Connect the GND and VCC pins of the SW-420 sensor to GND and +5V Pins of Raspberry pi respectively.

- Connect the digital output pin of the SW-420 sensor to the configured [input] GPIO pin of RPi.

- Connect Negative pin of LED [Interrupt Indicator] to GND pin of RPi, and

- Connect Positive of LED to the configured [output] GPIO pin of RPi.

For Example,

```
unsigned int LED = 0;             //GPIO_IN value toggle
unsigned int IRQ_NUM;             //For IRQ number

unsigned int GPIO_IN = 25;        // For Interrupt Handling [SW- 420]
unsigned int GPIO_OUT = 21;       // Interrupt Indicator [LED]
```

- Here PIN 40 of the RPi [GPIO 21] is configured as OUTPUT, So the LED positive terminal will be connected to GPIO 21.

- PIN 22 of the RPi [GPIO 25] is configured as INPUT, So the SW-420 sensor digital output pin will be connected to GPIO 25.

KERNEL SPACE DRIVER & BUILDING PROCESS

```c
#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/device.h>
#include <linux/delay.h>
#include <linux/jiffies.h>
#include <linux/uaccess.h>
#include <linux/gpio.h>
#include <linux/interrupt.h>
#define EN_DEBOUNCE



unsigned int LED = 0;             //GPIO_IN value toggle
unsigned int IRQ_NUM;             //For IRQ number
unsigned int GPIO_IN = 25;        // Interrupt Generator [SW- 420]
unsigned int GPIO_OUT = 21;       // Interrupt Indicator [LED]

#ifdef EN_DEBOUNCE
extern unsigned long volatile jiffies;
unsigned long shake = 0;
#endif


static irqreturn_t GPIO_IRQ(int irq,void *dev_id)   //Interrupt handler will be invoked when raising edge detected.
{
  static unsigned long flags = 0;
#ifdef EN_DEBOUNCE
   unsigned long diff = jiffies - shake;
   if (diff < 40)
   {
     return IRQ_HANDLED;
   }
  shake = jiffies;
#endif

  local_irq_save(flags);
  LED = (0x01 ^ LED);
  gpio_set_value(GPIO_OUT, LED);                    // For Interrupt indicator toggling
  pr_info("Interrupt has occurred, Configured OUTPUT GPIO Logic %d \n",gpio_get_value(GPIO_OUT));
  local_irq_restore(flags);
  return IRQ_HANDLED;
}


// holds device number
dev_t dev = 0;

// cdev variable & file operation
static struct class *dev_class;
static struct cdev etx_cdev;
```

```c
//Driver functions
static int __init GPIO_driver_init(void);
static void __exit GPIO_driver_exit(void);
static int intr_open(struct inode *inode, struct file *file);
static ssize_t intr_read(struct file *filp, char __user *buf, size_t len,loff_t * off);
static ssize_t intr_write(struct file *filp, const char *buf, size_t len, loff_t * off);
static int intr_release(struct inode *inode, struct file *file);

//File operations structure
static struct file_operations fops =
{
  .read          = intr_read,
  .write         = intr_write,
  .open          = intr_open,
  .release       = intr_release,
  .owner         = THIS_MODULE,
};


// Open Function
static int intr_open(struct inode *inode, struct file *file)
{
  pr_info("GPIO Interrupt Driver File Opened\n");
  return 0;
}

//Close Function
static int intr_release(struct inode *inode, struct file *file)
{
  pr_info("GPIO Interrupt Driver File Closed\n");
  return 0;
}

// Read Function
static ssize_t intr_read(struct file *filp, char __user *buf, size_t len, loff_t *off)
{
  uint8_t gpio_state = 0;
   // For reading the GPIO value
  gpio_state = gpio_get_value(GPIO_OUT);
  len = 1;
  if( copy_to_user(buf, &gpio_state, len) > 0) {
    pr_err("Insufficient data copied to user\n");
  }

  pr_info(" GPIO_OUT = %d read function \n", gpio_state);

  return 0;
}
```

```c
//Write Function
static ssize_t intr_write(struct file *filp, const char __user *buf, size_t len, loff_t *off)
{
  uint8_t rec_buf[10] = {0};

  if( copy_from_user( rec_buf, buf, len ) > 0) {
    pr_err("Insufficent data copied from user\n");
  }

  pr_info(" GPIO_OUT = %c write function \n", rec_buf[0]);

  if (rec_buf[0]=='1') {              // set GPIO Out pin
    gpio_set_value(GPIO_OUT, 1);
  } else if (rec_buf[0]=='0') {       //reset GPIO Out pin
    gpio_set_value(GPIO_OUT, 0);
  } else {
    pr_err(" Invalid Value, insert 0/1 \n");
  }

  return len;
}



// Module Init function
static int __init GPIO_driver_init(void)
{

  // Dynamic allocation of device number
  // Single device with base minor value 0

  if((alloc_chrdev_region(&dev, 0, 1, "etx_Dev")) <0){
    pr_err("Cannot allocate the major number to the GPIO Driver\n");
    goto r_unreg;
  }

  //Printing the device number information
  pr_info("Allocated Major Number = %d Minor Number = %d \n",MAJOR(dev), MINOR(dev));


  // For device registration,  initializing cdev structure with file operations

  cdev_init(&etx_cdev,&fops);

  //Device Registration
  if((cdev_add(&etx_cdev,dev,1)) < 0){
    pr_err("Cannot insert the device file\n");
    goto r_del;
  }

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

```
  //Creating device
  if((device_create(dev_class,NULL,dev,NULL,"etx_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;
  }

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

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

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

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

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

//  EN_DEBOUNCE to handle the driver.
#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
  IRQ_NUM = gpio_to_irq(GPIO_IN);
  pr_info("GPIO Interrupt Req Number = %d \n", IRQ_NUM);

  if (request_irq(IRQ_NUM, (void *)GPIO_IRQ, IRQF_TRIGGER_RISING, "etx_device", NULL)) {
    pr_err("Interrupt Request cannot registered");
    goto r_gpio_in;
  }

  pr_info("GPIO Interrupt Driver Inserted\n");
  return 0;

r_gpio_in:
  gpio_free(GPIO_IN);
r_gpio_out:
  gpio_free(GPIO_OUT);
r_device:
  device_destroy(dev_class,dev);
r_class:
  class_destroy(dev_class);
r_del:
  cdev_del(&etx_cdev);
r_unreg:
  unregister_chrdev_region(dev,1);
  return -1;
}



// Module exit function
static void __exit GPIO_driver_exit(void)
{
  free_irq(IRQ_NUM,NULL);
  gpio_free(GPIO_IN);
  gpio_free(GPIO_OUT);
  device_destroy(dev_class,dev);
  class_destroy(dev_class);
  cdev_del(&etx_cdev);
  unregister_chrdev_region(dev, 1);
  pr_info("GPIO Interrupt Driver Removed\n");
}


module_init(GPIO_driver_init);
module_exit(GPIO_driver_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Chakradhar");
MODULE_DESCRIPTION("GPIO Interrupt Driver");
```

MAKEFILE

```
obj-m += driver.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
```

SOFTWARE

- Compile and generate the kernel object “.ko” file from .c file using Makefile with the command "make" in the linux terminal.

- Insert the ".ko" driver file into the system kernel using the command " sudo insmod <filename.ko>".

- "lsmod" command can be used to check the inserted device driver file in the kernel modules list.

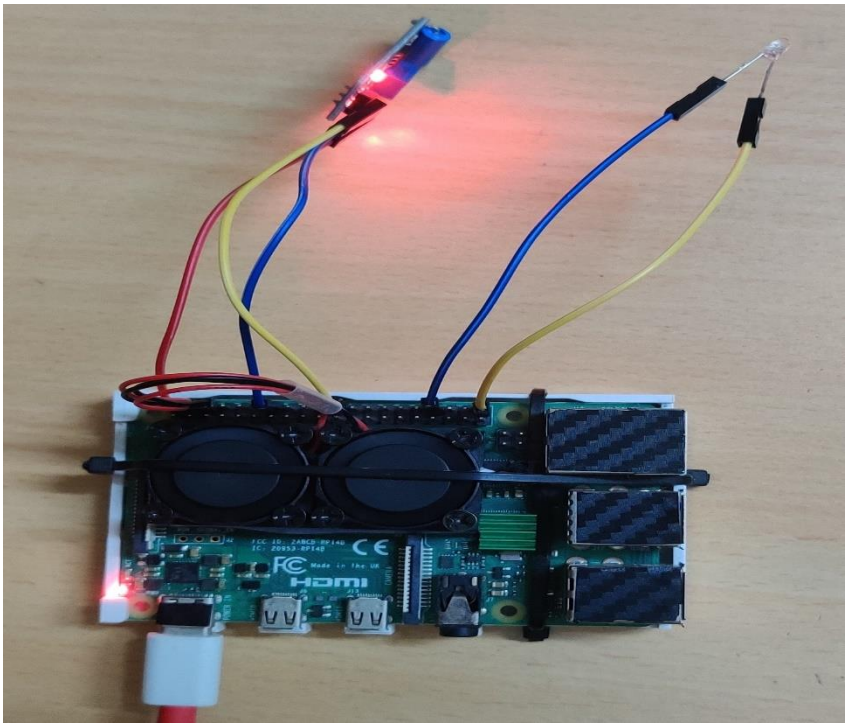
- After inserting the GPIO Interrupt driver file, LED can be toggled by vibrating the SW-420 sensor.

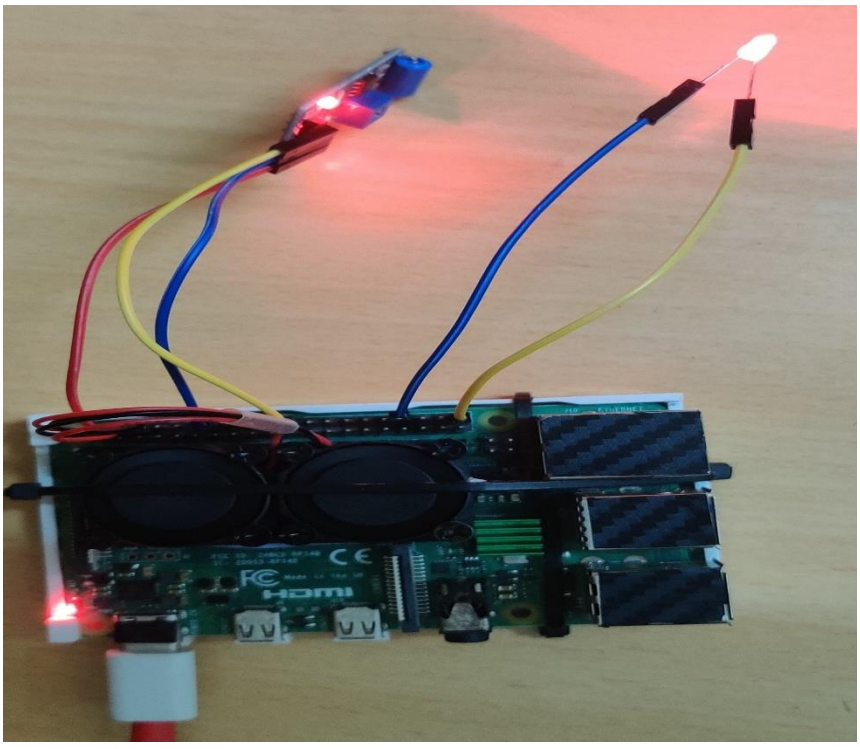
- "sudo rmmod <filename>" command will remove the inserted driver file from kernel.

- "dmesg" command can be used to check the kernel log.

Output:

When the [Interrupt] vibration is detected, the sensor outputs Logic High which triggers the [Interrupt Indicator] LED to turn ON/OFF depending on it's previous state.





Output Messages