A REPORT ON

Driver for I2C PROTOCOL MPU6050 SENSOR

BY

MALI SAINADH	**2020H1400219H**
SAYAN BAIDYA	**2020H1400235H**

M.E. EMBEDDED SYSTEMS

Prepared in fulfilment of the

(EEE G547)

Device Drivers

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(December 2021)

TABLE OF CONTENTS

1. SUMMARY

Sensor – MPU6050

- MPU6050 is a three-axis accelerometer and three-axis gyroscope and accelerometer Micro Electro-mechanical system (MEMS).

- It aids in the measurement of velocity, direction, acceleration, displacement, and other motion-related characteristics.

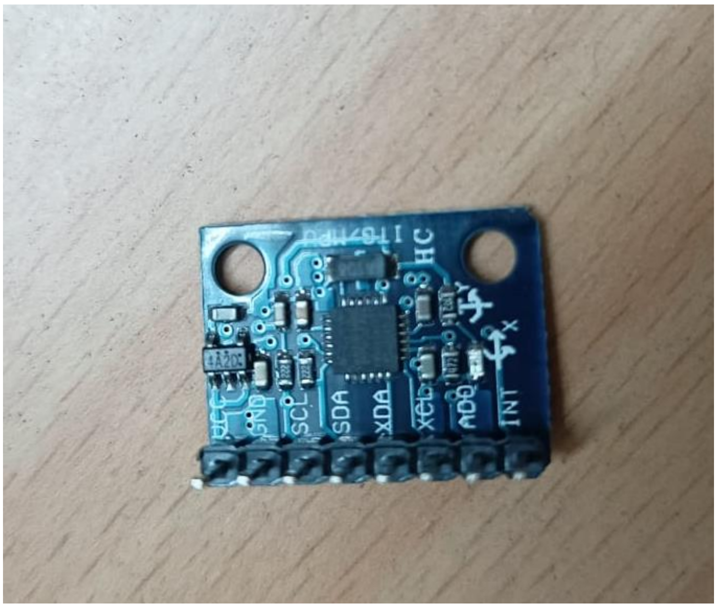


Figure 1. MPU 6050

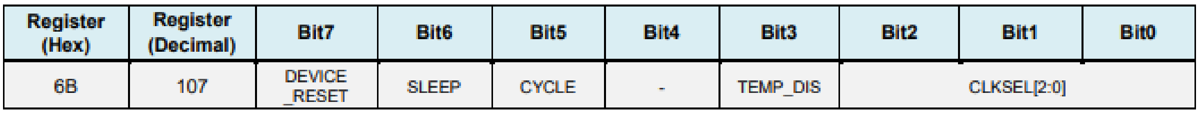
- The MPU6050's gyroscope can detect rotation around the three axes of X, Y, and Z.
- When the gyros are rotated around any of the axes, the Coriolis effect creates vibrations.
- The capacitor picks up on these vibrations.
- After that, the signal is amplified, demodulated, and filtered to generate a voltage proportionate to the angular rate.
- The voltage is then converted to digital using ADCs.

2. CONFIGURING REQUIRED REGISTERS OF MPU6050 SENSOR

- **Power Management Configuration**

- It is the register number 107 (6B in in hexadecimal) and it will store the configuration of the power mode and the clock source.

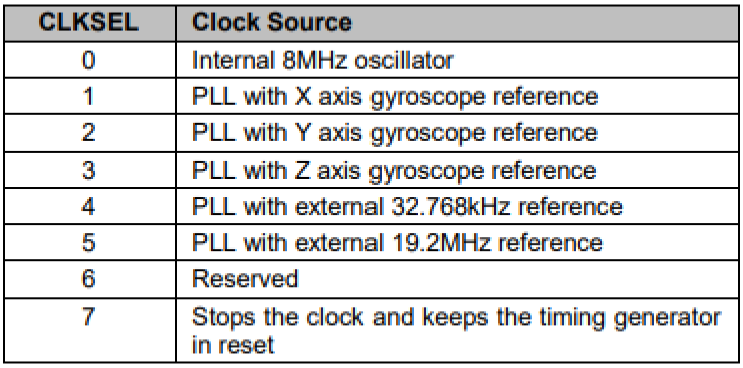
- Provides bits resetting and disabling the sensor.



Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE _RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

Figure 2. Power Management Register

- It is an 8-Bit register with,
  - ➢ Bit 7: To reset the device.
  - ➢ Bit 6: To put device in sleep mode.
  - ➢ Bit 5: To sleep and wakes up to take single sample data in the accelerometer.
  - ➢ Bit 4: It is reserved bit.
  - ➢ Bit 3: To disable temperature sensor.
  - ➢ Bit [2:0]: It selects the clock source for the device. (CLKSEL)



CLKSEL	Clock Source
0	Internal 8MHz oscillator
1	PLL with X axis gyroscope reference
2	PLL with Y axis gyroscope reference
3	PLL with Z axis gyroscope reference
4	PLL with external 32.768kHz reference
5	PLL with external 19.2MHz reference
6	Reserved
7	Stops the clock and keeps the timing generator in reset

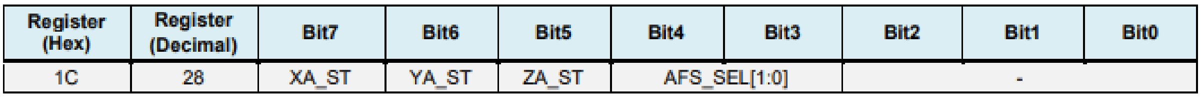
Figure 3. Bits [2:0] functionality of Power Management Register

- We have chosen CLKSEL as 1, i.e., PLL with X axis of gyroscope.

- So, the Power Management Config Register is configured with 0x01 value.

- **Accelerometer Configuration**

- It is the register number 28 (1C in in hexadecimal).



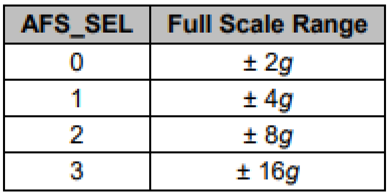
Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

Figure 4. Accelerometer configure Register

- This register is used to trigger accelerometer self-test and configure the accelerometer full scale range.

- It is an 8-bit Register, with

  - Bit 7: Self-test for accelerometer in X axis (kept 0).
  - Bit 6: Self-test for accelerometer in Y axis (kept 0).
  - Bit 5: Self-test for accelerometer in Z axis (kept 0).
  - Bit [4:3]: It selects the full-scale range of the accelerometer.
  - Bit [2:0]: These are kept 0.



AFS_SEL	Full Scale Range
0	± 2*g*
1	± 4*g*
2	± 8*g*
3	± 16*g*

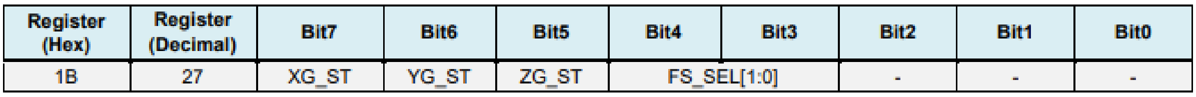
Figure 5. Bit [2:0] functionality of Accelerometer configure Register

- We are choosing Full scale range of +-2g.

- So, the Accelerometer configure register is configured with 0x00 value.

- **Gyroscope Configuration**

- It is the register number 27 (1B in in hexadecimal).



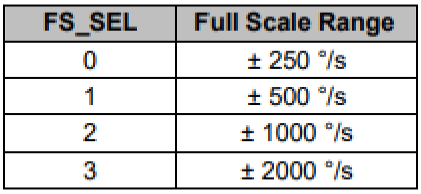
Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

Figure 6. Gyroscope configure Register

- This register is used to trigger gyroscope self-test and configure the gyroscope full scale range.

- It is an 8-bit Register, with

  - ➢ Bit 7: Self-test for gyroscope in X axis (kept 0).
  - ➢ Bit 6: Self-test for gyroscope in Y axis (kept 0).
  - ➢ Bit 5: Self-test for gyroscope in Z axis (kept 0).
  - ➢ Bit [4:3]: It selects the full-scale range of the gyroscope.
  - ➢ Bit [2:0]: These are kept 0.



FS_SEL	Full Scale Range
0	± 250 °/s
1	± 500 °/s
2	± 1000 °/s
3	± 2000 °/s

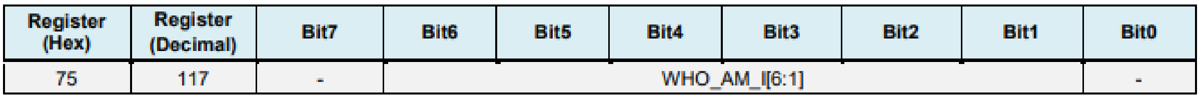
Figure 7. Bit [2:0] functionality of Gyroscope configure Register

- We are choosing Full scale range of +-2000 degrees/seconds.

- So, the Gyroscope configure register is configured with 0x18 value.

- **Who am I Register**

- It is the register number 117 (75 in hexadecimal)
- It is used for verifying identity of the device.



Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
75	117	-	WHO_AM_I[6:1]						-

Figure 8. Who Am I Register

- The address of the register, which is 0x75, is passed as input and if the values of Bit [6:1] is 110 100, then the device is properly identified and connection with the sensor is made successfully.

- Bit 0 and Bit 7 are hard coded as 0.

3. HARDWARE DESIGN

Schematic Diagram of Design

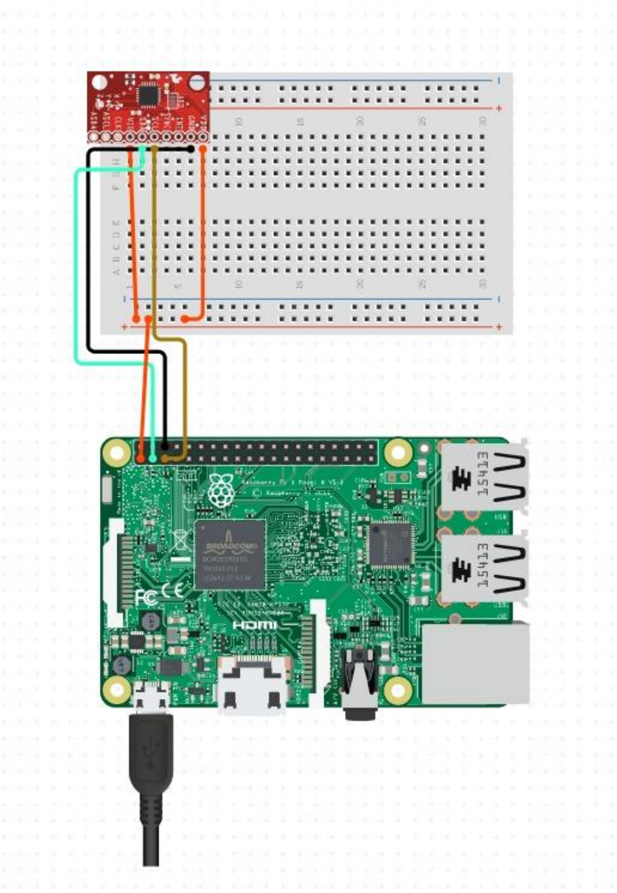


Figure 9. Schematic Diagram of Design

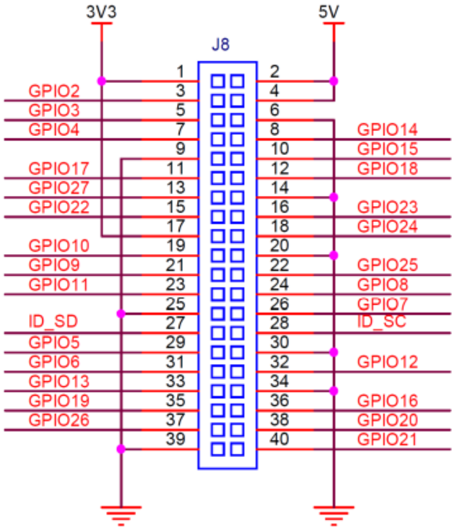


Figure 10: Pin Diagram of Raspberry Pi

The connections are as follows:

- 3.3V is given to pin 1 of Raspberry pi and the Vdd pin of MPU0605.
- Pin number 3 (GPIO2) of Raspberry Pi is connected to the SDA pin of MPU0605.
- Pin number 5 (GPIO3) of Raspberry Pi is connected to the SCL pin of MPU0605.
- Pin number 6 of Raspberry Pi and GND pin of MPU0605 are connected to ground.

Actual Design using Raspberry Pi and MPU6050

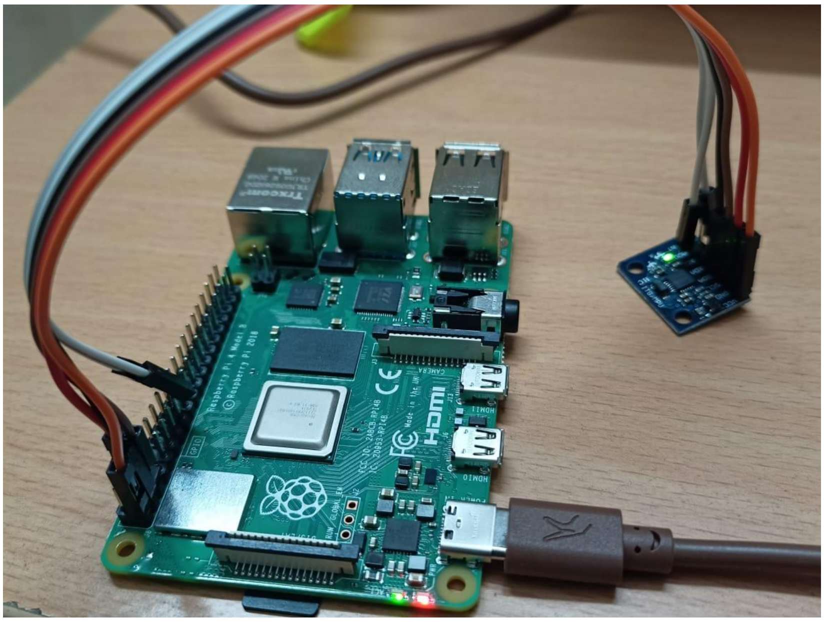


Figure 11. Actual Design using Raspberry Pi and MPU6050

4. CODES FOR KERNEL SPACE DRIVER AND USERSPACE

1. Kernel space driver code (main.c)

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/i2c.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/version.h>
#include <linux/cdev.h>
#include <linux/uaccess.h>
#include "mpu6050.h"

#define DRIVER_NAME "MPU6050"
#define DRIVER_CLASS "MPU6050Class"

static struct i2c_adapter * MPU6050_i2c_adapter = NULL;
static struct i2c_client * MPU6050_i2c_client = NULL;

static struct gyroData gyro;
static struct accelData accel;

#define I2C_BUS_AVAILABLE       1               /* I2C Bus SDA at GPIO2, SCL at GPIO3 available on the raspberry */
#define SLAVE_DEVICE_NAME       "MPU6050"       /* Device and Driver Name */
#define MPU6050_SLAVE_ADDRESS    0x68           /* I2C address of MPU6050 */

static const struct i2c_device_id MPU6050_id[]={
{ SLAVE_DEVICE_NAME, 0},
{ }
};

static struct i2c_driver MPU6050_driver = {
 .driver = {
   .name = SLAVE_DEVICE_NAME,
   .owner = THIS_MODULE
 }
};

static struct i2c_board_info MPU6050_i2c_board_info = {
 I2C_BOARD_INFO(SLAVE_DEVICE_NAME, MPU6050_SLAVE_ADDRESS)    // Platform device
};

static dev_t dev_no;        //variable for device number
static struct cdev c_dev;     //variable for the character device structure
static struct class *dev_class;  //varible for the device class

static int open_driver(struct inode *i, struct file *f)   //open function of file
{
 printk("Device : Open() \n");
 return 0;
}

static int close_driver(struct inode *i, struct file *f)  //close function of file
{
       printk("Device : Close()\n");
       return 0;
}

static uint32_t read_accelerometer(void)                              //reading accelerometer values from MPU_6050 sensor
{

   uint8_t lsb,msb,lsb1,msb1,lsb2,msb2;
   uint16_t x_accel,y_accel,z_accel;

   lsb = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x3C); //reading ACCEL_XOUT_LSB value of MPU6050 from reg60
located at 0x3C
   msb = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x3B); //reading ACCEL_XOUT_LSB value of MPU6050 from reg59
located at 0x3B
   x_accel = ((uint16_t)msb<<8) | ((uint16_t)lsb);          // concatinating the 8-bit values to form 16-bit value
   accel.x= x_accel;

   lsb1 = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x3E); //reading ACCEL_YOUT_LSB value of MPU6050 from reg62
located at 0x3E
```

```
  msb1 = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x3D); //reading ACCEL_YOUT_MSB value of MPU6050 from
reg61 located at 0x3D

  y_accel = ((uint16_t)msb1<<8) | ((uint16_t)lsb1);         // concatinating the 8-bit values to form 16-bit value

  accel.y=  y_accel;


  lsb2 = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x40); //reading ACCEL_ZOUT_LSB value of MPU6050 from reg64
located at 0x40

  msb2 = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x3F);    //reading ACCEL_ZOUT_MSB value of MPU6050 from
reg63 located at 0x3F

  z_accel = ((uint16_t)msb2<<8) | ((uint16_t)lsb2);         // concatinating the 8-bit values to form 16-bit value

  accel.z=  z_accel;

  return 0;


}

static uint32_t read_gyroscope(void)                        //reading gyroscope values from MPU_6050 sensor

{

  uint8_t lsb,msb,lsb1,msb1,lsb2,msb2;

  uint16_t gyro_x,gyro_y,gyro_z;


  lsb = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x44);  //reading GYRO_XOUT_LSB value of MPU6050 from reg68
located at 0x44

  msb = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x43);   //reading GYRO_XOUT_MSB value of MPU6050 from
reg67 located at 0x43

  gyro_x = ((uint16_t)msb<<8) | ((uint16_t)lsb);            // concatinating the 8-bit values to form 16-bit value

  gyro.x=  gyro_x;


  lsb1 = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x46); //reading GYRO_YOUT_LSB value of MPU6050 from reg70
located at 0x46

  msb1 = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x45); //reading GYRO_YOUT_MSB value of MPU6050 from
reg69 located at 0x45

  gyro_y = ((uint16_t)msb1<<8) | ((uint16_t)lsb1);          // concatinating the 8-bit values to form 16-bit value

  gyro.y=  gyro_y;


  lsb2 = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x48); //reading GYRO_YOUT_LSB value of MPU6050 from reg72
located at 0x48

  msb2 = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x47); //reading GYRO_YOUT_MSB value of MPU6050 from
reg71 located at 0x47

  gyro_z= ((uint16_t)msb2<<8) | ((uint16_t)lsb2);           // concatinating the 8-bit values to form 16-bit value

  gyro.z=  gyro_z;

  return 0;
```

```
}

static ssize_t read_driver(struct file *File, char __user *user_buffer, size_t count, loff_t *offs)  //read function of file
{
 int to_copy, not_copied, delta;
 char out_string[100];              // declaring no of characters to get printed
 int16_t accel_x,accel_y,accel_z;
 int16_t x_gyro,y_gyro,z_gyro;
 int32_t k,k1;
 to_copy = min(sizeof(out_string), count);

 k=read_accelerometer();            // Reading accelerometer values
 accel_x = accel.x;
 accel_y = accel.y;
 accel_z = accel.z;

 k1=read_gyroscope();             // Reading gyroscope values
 x_gyro = gyro.x;
 y_gyro = gyro.y;
 z_gyro = gyro.z;

snprintf(out_string,sizeof(out_string),"accel_readings:x:%d,y:%d,z:%d,Gyro_readings:x:%d,y:%d,z:%d,\n",accel_x,accel_y,ac
cel_z,x_gyro,y_gyro,z_gyro);
 not_copied = copy_to_user(user_buffer, out_string, to_copy);
 delta = to_copy - not_copied;
 return delta;
}
///////////////// IOCTL FUNCTION /////////////////////////////////////
long ioctl(struct file *file, unsigned int ioctl_num, unsigned long ioctl_param)      // number and param for ioctl
{

 switch(ioctl_num)
 {
  case IOCTL_GYRO:
  read_gyroscope();
  copy_to_user((struct gyroData *)ioctl_param,&gyro,sizeof(struct gyroData));   //passing gyro data to user space
  break;
 case IOCTL_ACCEL:
 read_accelerometer();
 copy_to_user((struct accelData *)ioctl_param,&accel,sizeof(struct accelData)); //passing accel data to user space
 break;

 }
return 0;
}
static struct file_operations fops  = {
 .owner = THIS_MODULE,
 .open = open_driver,
 .release = close_driver,
 .unlocked_ioctl = ioctl_dev,
 .read = read_driver,
};


static int __init mydriver_init(void)
{
 int ret =-1;
 u8 check;
 printk("Driver for MPU6050 sensor registered\n");
 /////////////////////reserve <major,minor>////////////////////////
 if ( alloc_chrdev_region(&dev_no, 0, 1, DRIVER_NAME) < 0) {
            printk("Failed to assign Device Number!\n");
      }
      printk("Driver with device number %d for MPU6050 sensor registered\n", dev_no);

  ///////////////dynamically create device node in /dev directory    ///////////////////////
      if ((dev_class = class_create(THIS_MODULE, DRIVER_CLASS)) == NULL)
                          {
            printk("Failed to create Device Class!\n");
             unregister_chrdev(dev_no, DRIVER_NAME);  //unregistering the character device with major and minor number
            return (-1);
      }
```

```
//////////// creating device node ////////////////////////
    if (device_create(dev_class, NULL, dev_no, NULL, DRIVER_NAME) == NULL)
   {
        printk("Failed to create device file!\n");
         class_destroy(dev_class);                  //destroying the device class
         unregister_chrdev(dev_no, DRIVER_NAME);     //unregistering the character device with major and minor number
        return (-1);
    }



/////////// Link file_operations and Cdev to device node///////////////////
    cdev_init(&c_dev, &fops);

    /* register device to kernel */
    if (cdev_add(&c_dev, dev_no, 1) == -1)
                    {
        printk("Failed to register device to kernel!\n");
         device_destroy(dev_class, dev_no);           //destroy device node
         class_destroy(dev_class);                    //destroying the device class
         unregister_chrdev(dev_no, DRIVER_NAME);       //unregistering the character device with major and minor number
        return (-1);
    }

    MPU6050_i2c_adapter = i2c_get_adapter(I2C_BUS_AVAILABLE);

    if(MPU6050_i2c_adapter != NULL) {
        MPU6050_i2c_client = i2c_new_client_device(MPU6050_i2c_adapter, &MPU6050_i2c_board_info);
        if(MPU6050_i2c_client != NULL) {
            if(i2c_add_driver(&MPU6050_driver) != -1) {
                ret = 0;
            }
            else
                printk("Can't add driver...\n");
        }
        i2c_put_adapter(MPU6050_i2c_adapter);
    }
```

```
       printk("MPU6050 Driver Init\n");


              check = i2c_smbus_read_byte_data(MPU6050_i2c_client, 0x75);      // Reading value from WHO AM I stored at
reg117 located at 0x75 which returns bit6:bit1 110 100 whether connection is established or not

    printk("Checking whether communication is established or not: 0x%x\n",check);

     i2c_smbus_write_byte_data(MPU6050_i2c_client, 0x6B, 0x01);  //Configuring power management-1 located at 0x6B

     i2c_smbus_write_byte_data(MPU6050_i2c_client, 0x1B, 0x18);  //Configuring Gyroscope at reg27 located at 0x1B with
full range of +/- 2000 degrees /s

     i2c_smbus_write_byte_data(MPU6050_i2c_client, 0x1C, 0x00);  //Configuring accelerometer at reg28 located at 0x1C
with full range of +/- 2g

       return ret;

}


static void __exit mydriver_exit(void) {

       i2c_unregister_device(MPU6050_i2c_client);   // unregistering the i2c client device

       i2c_del_driver(&MPU6050_driver);         // unregister I2C driver

       cdev_del(&c_dev);                    //deleting the link between cdev and file operations

       device_destroy(dev_class, dev_no);     //destroy device node

       class_destroy(dev_class);            //destroying the device class

       unregister_chrdev_region(dev_no, 1);  //unregistering the character device with major and minor number

       printk("Bye:Driver for MPU6050 sensor unregistered!\n");

}


module_init(mydriver_init);

module_exit(mydriver_exit);

MODULE_AUTHOR("SAINADH");

MODULE_LICENSE("GPL");

MODULE_DESCRIPTION("MPU6050 Sensor Driver");
```

2. Header File (mpu6050.h)

```
#ifndef CHAR_CONFIG_H
#define CHAR_CONFIG_H
#include <linux/ioctl.h>
#define MAGIC_NUM 225

struct gyroData
{
 int16_t x;
 int16_t y;
 int16_t z;
};

struct accelData
{
 int16_t x;
 int16_t y;
 int16_t z;
};

#define IOCTL_GYRO  _IOWR(MAGIC_NUM, 0, struct gyroData*)   //request code for gyroscope ioctl
#define IOCTL_ACCEL  _IOWR(MAGIC_NUM, 1, struct accelData*)  // request code for accelerometer ioctl
//Device file interface
#define DEVICE_FILE_NAME "/dev/MPU6050"

#endif
```

3. Makefile

```
OUTPUT = userspace
obj-m := main.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules
	gcc -o $(OUTPUT) $(OUTPUT).c

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
	rm userspace
```

4. Userspace application (userspace.c)

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/ioctl.h>
#include<time.h>
#include<fcntl.h>
#include<signal.h>
#include<unistd.h>
#include "mpu6050.h"
int file_desc;

/* Functions for the ioctl calls */
int ioctl_accelerometer(int file_desc, struct accelData *msg)    //Function that reads the Values from Accelerometer
{
 int ret_val;

 ret_val = ioctl(file_desc, IOCTL_ACCEL,msg);    //ioctl function call with request code  IOCTL_ACCEL
 return ret_val;
}

int ioctl_gyroscope(int file_desc, struct gyroData *msg)    //Function that reads the Values from Gyroscope
{
 int ret_val;
 ret_val = ioctl(file_desc, IOCTL_GYRO,msg);     //ioctl function call with request code  IOCTL_GYRO
 return 0;
}
/* Main - Call the ioctl functions */
int main(void)
{
 int ret_val,i;
 struct gyroData gyro_data;
 struct accelData accel_data;
 float xaccel,yaccel,zaccel,xgyro,ygyro,zgyro;

 file_desc = open(DEVICE_FILE_NAME,0); // opening the device node and returning the value to file_desc

 if(file_desc<0)
 {
  printf(" Failed to open device %s\n",DEVICE_FILE_NAME);  //Display if permission is denied to open /dev/MPU6050
  exit(-1);
 }
 while(1)
 {
  ioctl_accelerometer(file_desc,&accel_data);   //Calling the accelerometer ioctl function call
  xaccel= ((float)(accel_data.x))/16384;       // Calibrating the obtained value as per register map datasheet
  yaccel= ((float)(accel_data.y))/16384;       // Calibrating the obtained value as per register map datasheet
  zaccel= ((float)(accel_data.z))/16384;       // Calibrating the obtained value as per register map datasheet
  printf("Raw accelerometer readings: x:%d , y:%d, z:%d\n",accel_data.x,accel_data.y,accel_data.z);
  printf("After calibrating accelerometer readings: x:%f , y:%f, z:%f\n",xaccel,yaccel,zaccel);
  ioctl_gyroscope(file_desc,&gyro_data);      //Calling the gyroscope ioctl function call
  xgyro= ((float)(gyro_data.x))/16.4;       // Calibrating the obtained value as per register map datasheet
  ygyro= ((float)(gyro_data.y))/16.4;       // Calibrating the obtained value as per register map datasheet
  zgyro= ((float)(gyro_data.z))/16.4;       // Calibrating the obtained value as per register map datasheet
  printf("Raw gyroscope readings:x:%d , y:%d, z:%d\n",gyro_data.x,gyro_data.y,gyro_data.z);
  printf("After calibrating gyroscope readings:x:%f , y:%f, z:%f\n",xgyro,ygyro,zgyro);
  sleep(5);
 }
}
```

5. PROCEDURE TO BUILD AND INSERT DRIVER IN KERNEL AND TO RUN USERSPACE

Step-1 : Change path of the system to the directory where all the required driver files are stored using the following command

```
cd  path_address
```

Step-2 : Now here Makefile consists of creating object files, kernel object file and compiling userspace application. Following Command is used

```
sudo make all
```

Step-3 : In this step , we insert the driver in kernel using the following command

```
sudo insmod main.ko
```

Step-4 : To check whether the MPU6050 sensor is identified or not , we are reading WHO AM I register of MPU6050 whose value bit6:bit1 is 110 100 which will be printed in kernel log. Use the following command to check

```
dmesg
```

Note: Check Bit6:Bit1 in the value displayed in the kernel log equal to 110 100.

Step-5: To check the raw data readings of accelerometer and gyroscope in the kernel, we use the following command

```
sudo cat /dev/MPU6050
```

Step-6: As userspace application program is compiled in Makefile so we directly see the output of userspace program using following command

```
sudo  ./userspace
```

Note: If we want to recompile the userspace application program, use the following command

```
sudo gcc -o userspace userspace.c
```

Step-7 : To remove the driver from the kernel use the following command

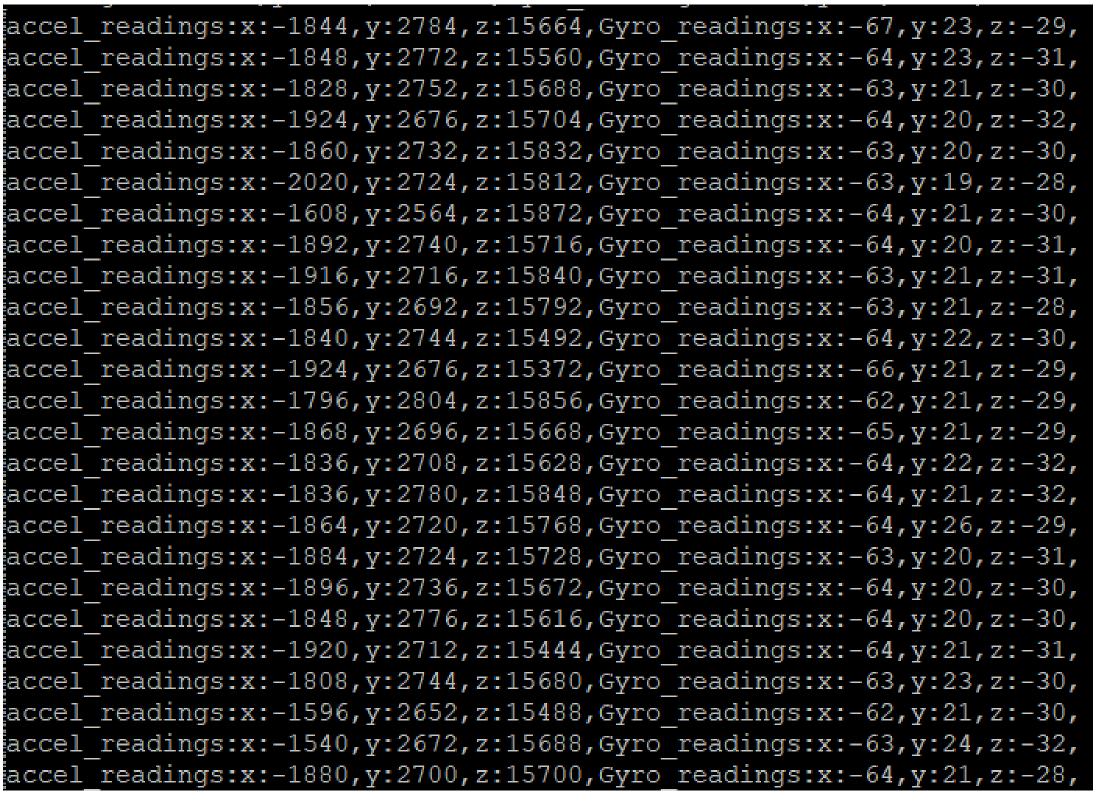
```
sudo rmmod main.ko
```

Step-8 : To remove the object files use the following command

```
sudo make clean
```

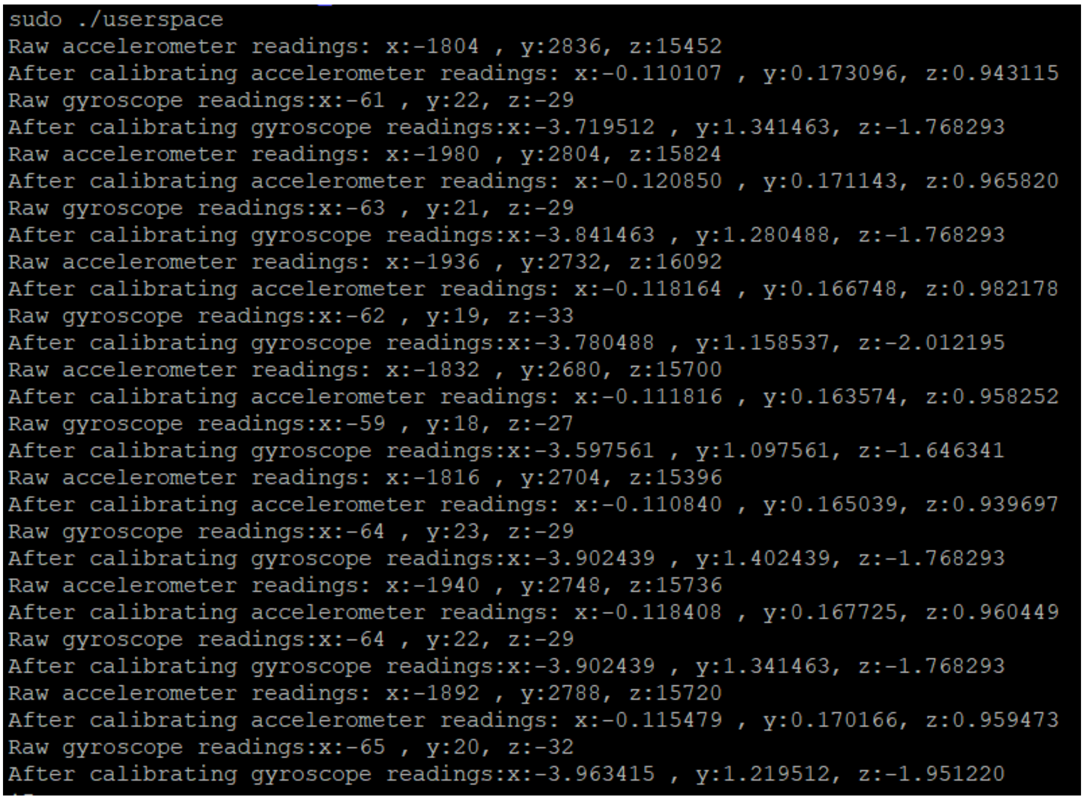
6. OUTPUTS

1. Output displayed using kernel driver



```
accel_readings:x:-1844,y:2784,z:15664,Gyro_readings:x:-67,y:23,z:-29,
accel_readings:x:-1848,y:2772,z:15560,Gyro_readings:x:-64,y:23,z:-31,
accel_readings:x:-1828,y:2752,z:15688,Gyro_readings:x:-63,y:21,z:-30,
accel_readings:x:-1924,y:2676,z:15704,Gyro_readings:x:-64,y:20,z:-32,
accel_readings:x:-1860,y:2732,z:15832,Gyro_readings:x:-63,y:20,z:-30,
accel_readings:x:-2020,y:2724,z:15812,Gyro_readings:x:-63,y:19,z:-28,
accel_readings:x:-1608,y:2564,z:15872,Gyro_readings:x:-64,y:21,z:-30,
accel_readings:x:-1892,y:2740,z:15716,Gyro_readings:x:-64,y:20,z:-31,
accel_readings:x:-1916,y:2716,z:15840,Gyro_readings:x:-63,y:21,z:-31,
accel_readings:x:-1856,y:2692,z:15792,Gyro_readings:x:-63,y:21,z:-28,
accel_readings:x:-1840,y:2744,z:15492,Gyro_readings:x:-64,y:22,z:-30,
accel_readings:x:-1924,y:2676,z:15372,Gyro_readings:x:-66,y:21,z:-29,
accel_readings:x:-1796,y:2804,z:15856,Gyro_readings:x:-62,y:21,z:-29,
accel_readings:x:-1868,y:2696,z:15668,Gyro_readings:x:-65,y:21,z:-29,
accel_readings:x:-1836,y:2708,z:15628,Gyro_readings:x:-64,y:22,z:-32,
accel_readings:x:-1836,y:2780,z:15848,Gyro_readings:x:-64,y:21,z:-32,
accel_readings:x:-1864,y:2720,z:15768,Gyro_readings:x:-64,y:26,z:-29,
accel_readings:x:-1884,y:2724,z:15728,Gyro_readings:x:-63,y:20,z:-31,
accel_readings:x:-1896,y:2736,z:15672,Gyro_readings:x:-64,y:20,z:-30,
accel_readings:x:-1848,y:2776,z:15616,Gyro_readings:x:-64,y:20,z:-30,
accel_readings:x:-1920,y:2712,z:15444,Gyro_readings:x:-64,y:21,z:-31,
accel_readings:x:-1808,y:2744,z:15680,Gyro_readings:x:-63,y:23,z:-30,
accel_readings:x:-1596,y:2652,z:15488,Gyro_readings:x:-62,y:21,z:-30,
accel_readings:x:-1540,y:2672,z:15688,Gyro_readings:x:-63,y:24,z:-32,
accel_readings:x:-1880,y:2700,z:15700,Gyro_readings:x:-64,y:21,z:-28,
```

2. Output displayed by userspace program



```
sudo ./userspace
Raw accelerometer readings: x:-1804 , y:2836, z:15452
After calibrating accelerometer readings: x:-0.110107 , y:0.173096, z:0.943115
Raw gyroscope readings:x:-61 , y:22, z:-29
After calibrating gyroscope readings:x:-3.719512 , y:1.341463, z:-1.768293
Raw accelerometer readings: x:-1980 , y:2804, z:15824
After calibrating accelerometer readings: x:-0.120850 , y:0.171143, z:0.965820
Raw gyroscope readings:x:-63 , y:21, z:-29
After calibrating gyroscope readings:x:-3.841463 , y:1.280488, z:-1.768293
Raw accelerometer readings: x:-1936 , y:2732, z:16092
After calibrating accelerometer readings: x:-0.118164 , y:0.166748, z:0.982178
Raw gyroscope readings:x:-62 , y:19, z:-33
After calibrating gyroscope readings:x:-3.780488 , y:1.158537, z:-2.012195
Raw accelerometer readings: x:-1832 , y:2680, z:15700
After calibrating accelerometer readings: x:-0.111816 , y:0.163574, z:0.958252
Raw gyroscope readings:x:-59 , y:18, z:-27
After calibrating gyroscope readings:x:-3.597561 , y:1.097561, z:-1.646341
Raw accelerometer readings: x:-1816 , y:2704, z:15396
After calibrating accelerometer readings: x:-0.110840 , y:0.165039, z:0.939697
Raw gyroscope readings:x:-64 , y:23, z:-29
After calibrating gyroscope readings:x:-3.902439 , y:1.402439, z:-1.768293
Raw accelerometer readings: x:-1940 , y:2748, z:15736
After calibrating accelerometer readings: x:-0.118408 , y:0.167725, z:0.960449
Raw gyroscope readings:x:-64 , y:22, z:-29
After calibrating gyroscope readings:x:-3.902439 , y:1.341463, z:-1.768293
Raw accelerometer readings: x:-1892 , y:2788, z:15720
After calibrating accelerometer readings: x:-0.115479 , y:0.170166, z:0.959473
Raw gyroscope readings:x:-65 , y:20, z:-32
After calibrating gyroscope readings:x:-3.963415 , y:1.219512, z:-1.951220
```