A REPORT ON

Driver for ICM20948

BY

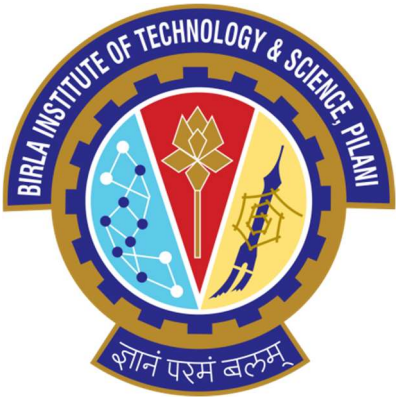
CHEKURI DEEPAK	**2020H1400222H**
POLICE MANOJ KUMAR REDDY	**2020H1400246H**
AMAN KUMAR	**2020H1400228H**

M.E. EMBEDDED SYSTEMS

Prepared in fulfillment of the

(MEL G547)

DEVICE DRIVERS



BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(DECEMBER 2021)

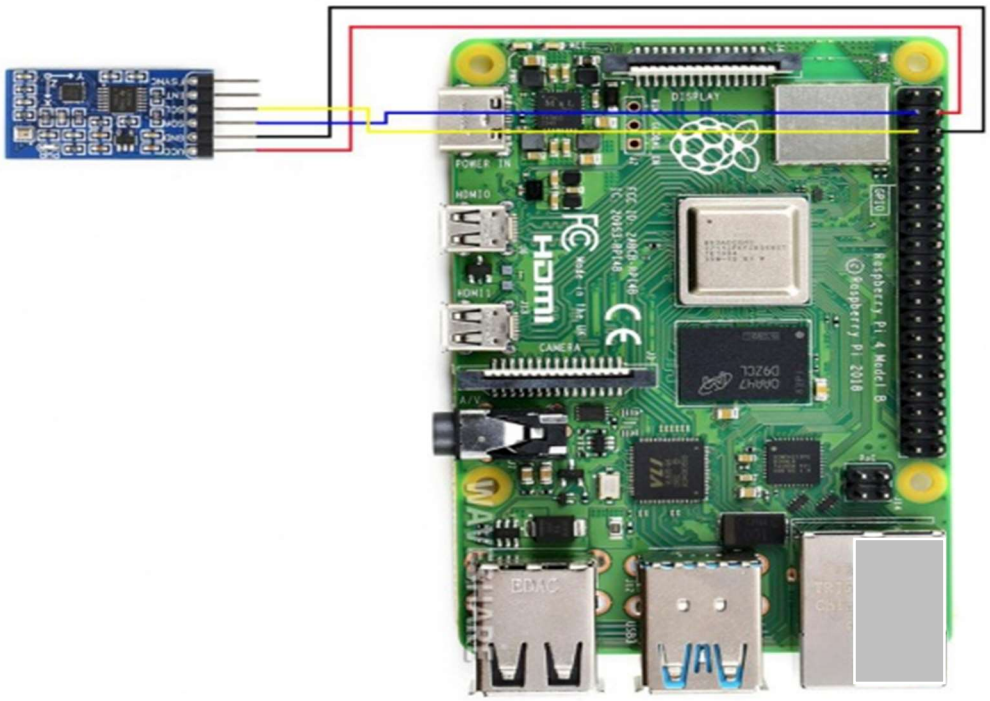
SUMMARY

This project aims to develop a driver for ICM20948. The ICM-20948 is the world's lowest-power 9-axis motion-tracking device, making it excellent for Smartphones, Tablets, Wearable Sensors, and Internet-of-Things (IoT) applications. Auxiliary I2 C interface to external sensors, on-chip 16-bit ADCs, programmable digital filters, an inbuilt temperature sensor, and programmable interrupts are all features of the ICM-20948. I2C protocol is used to communicate between ICM20948 and the master device.

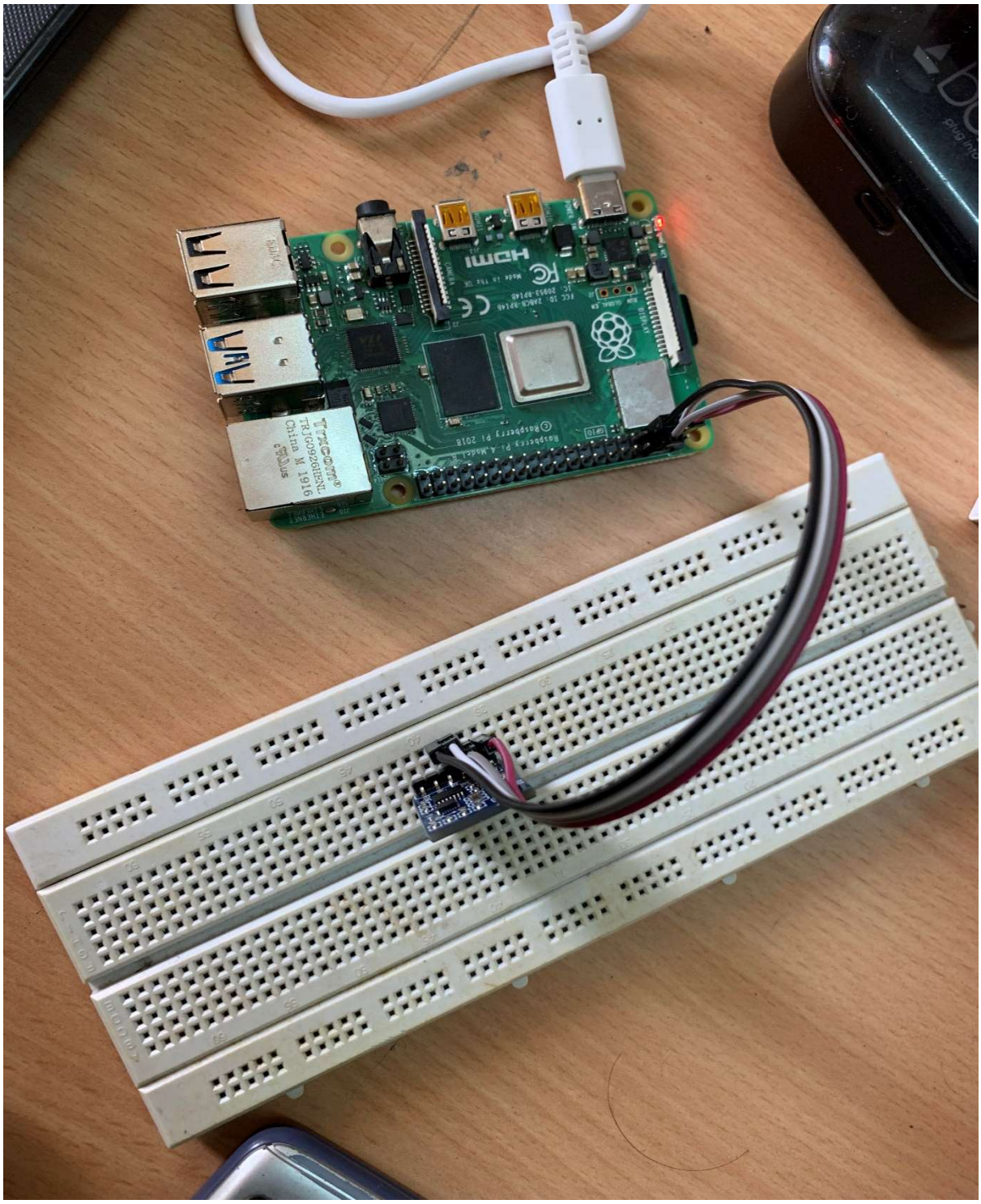
We have developed a driver for reading 3-axis accelerometer and 3-axis gyroscope data in this project. Raspberry pi has been taken as the master device.

HARDWARE

Schematic



Hardware wiring



Kernel Space Driver

```
#include <linux/ioctl.h>

#include <linux/types.h>

#include <linux/module.h>

#include <linux/init.h>

#include <linux/fs.h>

#include <linux/version.h>

#include <linux/cdev.h>

#include <linux/uaccess.h>

#include <linux/slab.h>

#include <linux/i2c.h>

#include <linux/kernel.h>

#include <linux/delay.h>



#include "config.h"



#define DRIVER_NAME "imu20948"

#define DRIVER_CLASS "imu20948Class"
```

```
static struct i2c_adapter *imu_i2c_adapter = NULL;

static struct i2c_client *imu20948_i2c_client = NULL;



static struct gyroData gyro;

static struct accelData accel;




#define I2C_BUS_AVAILABLE       1               /* The I2C Bus available on the raspberry */

#define SLAVE_DEVICE_NAME       "imu20948"      /* Device and Driver Name */

#define IMU20948_SLAVE_ADDRESS    0x68          /* IMU20948 I2C address */



/*****************************************************************/


static const struct i2c_device_id imu_id[]={

{ SLAVE_DEVICE_NAME, 0},

{ }

};


static struct i2c_driver imu_driver = {
```

```
 .driver = {
   .name = SLAVE_DEVICE_NAME,
   .owner = THIS_MODULE
  }
};

static struct i2c_board_info imu20948_i2c_board_info = {
 I2C_BOARD_INFO(SLAVE_DEVICE_NAME, IMU20948_SLAVE_ADDRESS)
};

/*****************************************************************/

static dev_t myDeviceNr;
static struct class *myClass;
static struct cdev myDevice;

/*****************************************************************/

static void read_accel(void)
{
```

```
    int8_t u8Buf[2];
    int16_t s16x,s16y,s16z;

    i2c_smbus_write_byte_data(imu20948_i2c_client, 0x7f,0x00);     //select reg bank 0

    u8Buf[0] = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x2E);  //read higher byte of
X-axis
    u8Buf[1] = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x2D);  //read lower byte of
X-axis
    s16x = ((uint32_t)u8Buf[1] << 8) | (uint32_t)u8Buf[0];          // read 16 bits raw data of x-
axis

    u8Buf[0] = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x30);  //read higher byte of
Y-axis
    u8Buf[1] = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x2F);  //read lower byte of
Y-axis
    s16y = ((uint32_t)u8Buf[1] << 8) | (uint32_t)u8Buf[0];          // read 16 bits raw data of y-
axis
```

```
    u8Buf[0] = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x32);  //read higher byte of
Z-axis
    u8Buf[1] = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x31);  //read lower byte of
Z-axis
    s16z = ((uint32_t)u8Buf[1] << 8) | (uint32_t)u8Buf[0];           // read 16 bits raw data of z-
axis


    accel.x=  s16x;
    accel.y=  s16y;
    accel.z=  s16z;
    return;
}


void read_gyro(void)
{
    int8_t u8Buf[2];
    int16_t s16Buf[3] ;
    int ir; // ir for error detection
```

```
    ir=i2c_smbus_write_byte_data(imu20948_i2c_client, 0x7f,0x00);   //select reg bank 0


    if(ir<0) return;


    u8Buf[0] = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x34); //read higher byte of
X-axis
    u8Buf[1] = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x33); //read lower byte of
X-axis
    s16Buf[0] = ((uint32_t)u8Buf[1] << 8) | (uint32_t)u8Buf[0];     // read 16 bits raw data of
x- axis


    u8Buf[0] = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x36); //read higher byte of
Y-axis
    u8Buf[1] = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x35); //read lower byte of
Y-axis
    s16Buf[1] = ((uint32_t)u8Buf[1] << 8) | (uint32_t)u8Buf[0];     // read 16 bits raw data of
y- axis


    u8Buf[0] = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x38); //read higher byte of
Z-axis
    u8Buf[1] = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x37); //read higher byte of
Z-axis
```

```
    s16Buf[2] = ((uint32_t)u8Buf[1] << 8) | (uint32_t)u8Buf[0];     // read 16 bits raw data of
z- axis









    gyro.x = s16Buf[0];

    gyro.y = s16Buf[1];

    gyro.z = s16Buf[2];



    return;

}



  /*****************************************************/



static ssize_t driver_read(struct file *File, char __user *user_buffer, size_t count, loff_t *offs)

{



   int to_copy, not_copied, delta;

   char out_string[100];
```

```
 int16_t  s16x,s16y,s16z;

 int16_t s16gx,s16gy,s16gz;




to_copy = min(sizeof(out_string), count);

read_accel();

s16x = accel.x;

s16y = accel.y;

s16z = accel.z;


read_gyro();

s16gx=gyro.x;

s16gy=gyro.y;

s16gz=gyro.z;


 snprintf(out_string, sizeof(out_string),
"accel_x:%d\ty:%d\tz:%d\tgyro_x:%d\t,y:%d\t,z:%d\n",s16x,s16y,s16z,s16gx,s16gy,s16gz);

 mdelay(10);
```

```
  not_copied = copy_to_user(user_buffer, out_string, to_copy);

  delta = to_copy - not_copied;

  return delta;
}

static int driver_close(struct inode *deviceFile, struct file *instance)
{
   printk("Driver Close\n");
   return 0;
}

static int driver_open(struct inode *deviceFile, struct file *instance)
{
  printk("Driver Open\n");
  return 0;
}

/**************************************************************/
```

```
long ioctl_dev(struct file *file, unsigned int ioctl_num, unsigned long ioctl_param)
{

 switch(ioctl_num)
 {
  case IOCTL_GYRO:
  read_gyro();
  copy_to_user((struct gyroData *)ioctl_param,&gyro,sizeof(struct gyroData));
  break;

  case IOCTL_ACCEL:
  read_accel();
  copy_to_user((struct accelData *)ioctl_param,&accel,sizeof(struct accelData));
  break;

 }
return 0;
}
```

```
/*****************************************************/


static struct file_operations fops  = {
 .owner = THIS_MODULE,
 .open = driver_open,
 .release = driver_close,
 .unlocked_ioctl = ioctl_dev,
 .read = driver_read,
};


/*****************************************************/
static int __init ModuleInit(void)
{


  int ret = -1;
  u8 id;
  int i[10]; // for error detection


  printk("MyDeviceDriver - Hello Kernel\n");
```

```
/* Allocate Device Nr */
 if ( alloc_chrdev_region(&myDeviceNr, 0, 1, DRIVER_NAME) < 0)
 {
   printk("Device Nr. could not be allocated!\n");
 }

 printk("MyDeviceDriver - Device Nr %d was registered\n", myDeviceNr);

 /* Create Device Class */
 if ((myClass = class_create(THIS_MODULE, DRIVER_CLASS)) == NULL)
 {
   printk("Device Class can not be created!\n");
   goto ClassError;
 }

 if (device_create(myClass, NULL, myDeviceNr, NULL, DRIVER_NAME) == NULL)
 {
   printk("Can not create device file!\n");
   goto FileError;
 }
```

```
  /* Initialize Device file */
  cdev_init(&myDevice, &fops);

  /* register device to kernel */
  if (cdev_add(&myDevice, myDeviceNr, 1) == -1) {
    printk("Registering of device to kernel failed!\n");
    goto KernelError;
  }

  /*****************************************************/

  imu_i2c_adapter = i2c_get_adapter(I2C_BUS_AVAILABLE);

  if(imu_i2c_adapter != NULL)
  {
    imu20948_i2c_client = i2c_new_client_device(imu_i2c_adapter,
&imu20948_i2c_board_info);
    if(imu20948_i2c_client != NULL)
    {
```

```
    if(i2c_add_driver(&imu_driver) != -1)
    {
      ret = 0;
    }
   else
      printk("Can't add driver...\n");
   }
   i2c_put_adapter(imu_i2c_adapter);
  }

  printk("IMU20948 Driver added!\n");

  /* Read Chip ID */
  id = i2c_smbus_read_byte_data(imu20948_i2c_client, 0x00);
  printk("ID: 0x%x\n", id);

  i[0]=i2c_smbus_write_byte_data(imu20948_i2c_client, 0x7F, 0x00);    // selecting user
bank 0
  i[1]=i2c_smbus_write_byte_data(imu20948_i2c_client, 0x06,  0x80);   // resetting power
management_1 register
```

```
  mdelay(100);
  i[2]=i2c_smbus_write_byte_data(imu20948_i2c_client, 0x06, 0x01);    //configuring power management_1 register to run mode
  i[3]=i2c_smbus_write_byte_data(imu20948_i2c_client, 0x07, 00);      //configuring power management_2 register to enable mode

  i[4]=i2c_smbus_write_byte_data(imu20948_i2c_client, 0x7F, 0x20);    // selecting user bank 2

  mdelay(100);
  i[5]=i2c_smbus_write_byte_data(imu20948_i2c_client, 0x01,0x34);     //initialize Gyro-config-1 reg
  mdelay(100);
  i[6]=i2c_smbus_write_byte_data(imu20948_i2c_client, 0x14,0x32);     //initialize Aceel-config-1 reg
  return ret;
  KernelError:
    device_destroy(myClass, myDeviceNr);
  FileError:
    class_destroy(myClass);
  ClassError:
```

```
    unregister_chrdev(myDeviceNr, DRIVER_NAME);
  return (-1);
}

static void __exit ModuleExit(void) {
  printk("MyDeviceDriver - Goodbye, Kernel!\n");
  i2c_unregister_device(imu20948_i2c_client);
  i2c_del_driver(&imu_driver);
  cdev_del(&myDevice);
  device_destroy(myClass, myDeviceNr);
  class_destroy(myClass);
  unregister_chrdev_region(myDeviceNr, 1);
}
/*********************************************************/
module_init(ModuleInit);
module_exit(ModuleExit);
MODULE_AUTHOR("DPK");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("IMU20948 Sensor Kernel Driver");
```

Userspace :

```
#include<stdio.h>

#include<stdlib.h>

#include<sys/ioctl.h>

#include<time.h>

#include<fcntl.h>

#include<signal.h>

#include<unistd.h>

#include "config.h"

int file_descript;

/************************************************/

int ioctl_accel(int file_descript, struct accelData *message)

{

 int ret_accel;

 ret_accel = ioctl(file_descript, IOCTL_ACCEL,message);

 return ret_accel;

}

int ioctl_gyro(int file_descript, struct gyroData *message)

{

 int ret_gyro;
```

```
 ret_gyro = ioctl(file_descript, IOCTL_GYRO,message);
 return 0;
}
/*************************************************************/
int main(void)
{
 int ret_val,i;
 struct gyroData gyro;
 struct accelData accel;
 float accel_x,accel_y,accel_z,gyro_x,gyro_y,gyro_z
 file_descript = open(DEVICE_FILE_NAME,0);
 if(file_descript<0)
 {
  printf(" Failed to open device %s\n",DEVICE_FILE_NAME);
  exit(-1);
 }
 printf("Logging started...\n");
 while(1)
 {
  ioctl_accel(file_descript,&accel);
```

```
  accel_x= ((float)(accel.x))/8192;

  accel_y= ((float)(accel.y))/8192;

  accel_z= ((float)(accel.z))/8192;


  printf("accelerometer readings: x:%f , y:%f, z:%f\n",accel_x,accel_y,accel_z);

sleep(1);

ioctl_gyro(file_descript,&gyro);

  gyro_x= ((float)(gyro.x))/32.8;

  gyro_y= ((float)(gyro.y))/32.8;

  gyro_z= ((float)(gyro.z))/32.8;


  printf("gyroscope readings: x:%f , y:%f, z:%f\n",gyro_x,gyro_y,gyro_z);

  sleep(1);

 }

}
```

Configuration file:

```
#define MAGIC_NUM 22

struct gyroData  // struct for gyro sensor data

{

 int16_t x;

 int16_t y;

 int16_t z;

};

struct accelData // struct for accel sensor data

{

 int16_t x;

 int16_t y;

 int16_t z;

};

//IOCTL interface prototypes

#define IOCTL_GYRO  _IOWR(MAGIC_NUM, 0, struct gyroData*)

#define IOCTL_ACCEL  _IOWR(MAGIC_NUM, 1, struct accelData*)

//Device file interface

#define DEVICE_FILE_NAME "/dev/imu20948"
```

Makefile:

```
USERFILE = user_file

obj-m := main.o

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

kern:
	make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules

user:
	gcc -o $(USERFILE) $(USERFILE).c

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

Build process:

step-1

Setup the Raspberry-pi for I2C communication and install the Kernel headers

sudo apt-get update

Step-2

Change the directory to the present directory with required files (main.c, config.h, user_file.c, Makefile)

sudo cd /dd_proj

step-3

Generate kernel object file for the kernel code

sudo make all

step-4

Insert the kernel module

sudo insmod main.ko

step-5

check the accelerometer and gyroscope raw data

sudo cat /dev/imu20948

step-6

check the accelerometer and gyroscope data using userspace file

sudo ./user_file