EEE G547
Interfacing of ADXL345 sensor using I2C protocol with Raspberry Pi 4b.

Aakanksha Gujarathi (2020H1400181P) and Anuj Sanathanan(2020H1400171P)

Summary:

This project aims to implement the i2c client side driver for ADXL345 accelerometer. It communicates to Raspberry Pi 4b using I2C bus 1. Apart from reading the acceleration values, the driver can implement the power-on and standby mode.

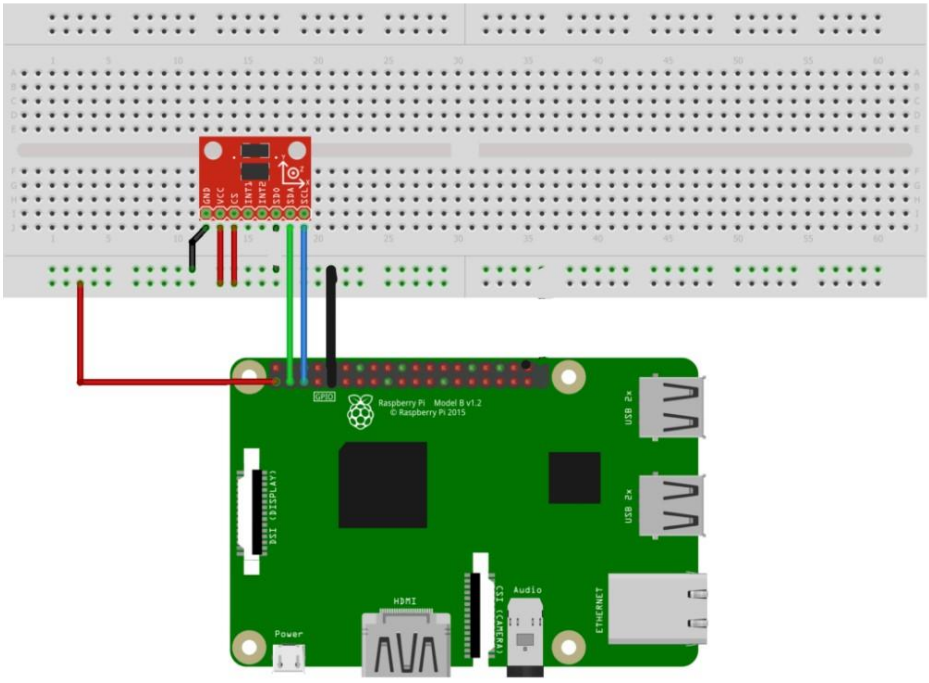
How it Works: MEMS - Micro Electro-Mechanical Systems:

The sensor consists of a micro-machined structure on a silicon wafer. The structure is suspended by polysilicon springs which allow it to deflect smoothly in any direction when subject to acceleration in the X, Y and/or Z axis. Deflection causes a change in capacitance between fixed plates and plates attached to the suspended structure. This change in capacitance on each axis is converted to an output voltage proportional to the acceleration on that axis.

Hardware Design:

Actual Hardware:

Schematic Hardware:



Kernel Space Driver:

The kernel space driver adxldriver.c is a client side driver that initializes the i2c client for communication with adxl345 and it also implements IOCTL calls for reading the 6 data registers and configuring the device into standby and poweron mode.

The function adxl_init() will create a device file named adxl345 in the /dev directory and initializes the I2C -1 bus.

```
my_adapter = i2c_get_adapter(1); // the adxl is connected to i2c-1
if (!(my_client = i2c_new_dummy_device(my_adapter, SLAVE_ADDRESS))){
        printk(KERN_INFO "Couldn't acquire i2c slave");
        device_destroy(cl, first);
        class_destroy(cl);
        unregister_chrdev_region(first, 1);
        return -1;
}
```

The driver first checks if the device is in reset mode.

```
readvalue = adxl_read(my_client,REG_DEVID);
if (readvalue == (0b11100101)) {
        printk("Accelerometer detected, value = %d\r\n",readvalue);
}
```

The device is set to measurement mode, with the data register right justified.

```
//Setting Data format to be right justified
temp = adxl_read(my_client,REG_DATA_FORMAT);
temp = temp & (~(1<<2));
adxl_write(my_client, REG_DATA_FORMAT, temp);


//setting the adxl in measurement mode.
temp = adxl_read(my_client,REG_POWER_CTL);
temp = temp | (1<<3);
adxl_write(my_client, REG_POWER_CTL, temp);
```

\

The read and write to the device is done using I2C_SMBS drivers in the function adxl_read and adxl_write.

```
//function to read the device using i2c bus
static u8 adxl_read(struct i2c_client *client, u8 reg)
{
        int ret;
        ret = i2c_smbus_read_byte_data(client, reg);
        if (ret < 0)
                dev_err(&client->dev,
                        "can not read register, returned %d\n", ret);

        return (u8)ret;
}

//function to write the device using i2c bus
static int adxl_write(struct i2c_client *client, u8 reg, u8 data)
{
        int ret;

        ret = i2c_smbus_write_byte_data(client, reg, data);
        if (ret < 0)
                dev_err(&client->dev,"can not write register, returned %d\n", ret);

        return ret;
}
```

By default the ADXL sensor operates in 100Hz, we can modify the rate of operation using the BW_RATE register.
The IOCTL call function implements the 11 calls for reading the 6 data registers, configuring the device into poweron, sleep and standby mode and changing the bandwidth to 3200 Hz (highest) and 0.2 Hz. The measurement data is copied and transferred to the user space application by ioctl_param using copy_to_user().

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

        u8 data,temp;
        switch(ioctl_num){
                case IOCTL_Accx0:data=adxl_read(my_client, REG_DATAX0);
                                 break;
                case IOCTL_Accx1 :data = adxl_read(my_client, REG_DATAX1);
                                 break;
                case IOCTL_Accy0:data =adxl_read(my_client, REG_DATAY0);
                                 break;
                case IOCTL_Accy1 :data = adxl_read(my_client, REG_DATAY1);
                                 break;
                case IOCTL_Accz0:data = adxl_read(my_client, REG_DATAZ0);
                                 break;
                case IOCTL_Accz1:data = adxl_read(my_client, REG_DATAZ1);
                                 break;
                case IOCTL_standby:temp = adxl_read(my_client,REG_POWER_CTL);
                                   temp = temp & ~(1<<3);
                                   adxl_write(my_client, REG_POWER_CTL, temp);
                                 break;
                case IOCTL_poweron:temp = adxl_read(my_client,REG_POWER_CTL);
                                   temp = temp | (1<<3);
                                   adxl_write(my_client, REG_POWER_CTL, temp);
                                 break;
                case IOCTL_sleep:temp = adxl_read(my_client,REG_POWER_CTL);
                                   temp = temp | (1<<3) | (1<<2);
                                   adxl_write(my_client, REG_POWER_CTL, temp);
                                 break;
                case IOCTL_BW_high:temp = adxl_read(my_client,REG_BW_RATE);
                                   temp = temp | (0b00001111);
                                   adxl_write(my_client, REG_POWER_CTL, temp);
                                 break;
                case IOCTL_BW_low:temp = adxl_read(my_client,REG_BW_RATE);
                                   temp = temp | (0b1);
                                   adxl_write(my_client, REG_POWER_CTL, temp);
                                 break;
                default: printk(KERN_ALERT "IOCTL SYS CALL ERROR");
                         data =0;
                         break;

        }
        copy_to_user((u8*)ioctl_param,&data, sizeof(data));
```

User Space Code:

The code opens the device file adxl345 for further tests The first set of measurements in measurement mode are displayed. The device is set to standby mode and measurements are displayed, in standby mode no further measurement is done by the sensor, and the last measured value is read from the respective registers, hence no change in the value of the accelerometer is seen in standby mode. The device is then configured to measurement mode to get the sensor working back to normal mode of operation by configuring the POWER_CONTROL register. The sensor can be operated in sleep mode by enabling the Sleep bit in the POWER_CONTROL register. When the sensor is put in sleep mode it operates at lower data rate and lower current(<40uA). The data rate in sleep is determined by wake bits of POWER_CONTROL register. Here we are putting the sensor at 8 Hz.

```
//Accelerometer readings
printf("Accelerometer readings\n");
for(int j = 0; j <2 ;j++){
        for(int i =0;i<3;i++){
                ioctl_readData(file_desc, i);
        }
}
ioctl_standby(file_desc);
printf("adxl345 on standby\n");
for(int j = 0; j <5 ;j++){
        for(int i =0;i<3;i++){
                ioctl_readData(file_desc, i);
        }
}
ioctl_poweron(file_desc);
printf("adxl345 powered on\n");
        for(int i =0;i<3;i++){
                ioctl_readData(file_desc, i);
        }
printf("adxl345 bandwidth changed to 3200HZ\n");
ioctl_BW_high(file_desc);
for(int j = 0; j <5 ;j++){
        for(int i =0;i<3;i++){
                ioctl_readData(file_desc, i);
        }
}
printf("adxl345 bandwidth changed to 0.2 HZ\n");
ioctl_BW_low(file_desc);
for(int j = 0; j <5 ;j++){
        for(int i =0;i<3;i++){
                ioctl_readData(file_desc, i);
        }
}
printf("adxl345 on sleep\n");
ioctl_sleep(file_desc);
for(int j = 0; j <2 ;j++){
        for(int i =0;i<3;i++){
                ioctl_readData(file_desc, i);
        }
}
```

Build process:

The following commands are to be executed:

1. make - the makefile contains the command to build the driver (adxldriver.c) and user space application (adxluserapp.c)
2. sudo insmod adxldriver.ko - to insert the kernel module object file.
3. sudo chmod 777 /dev/adxl345 - to change the permissions of the device file adxl345 and allow the user space application to open the file and call ioctl functions.
4. ./ioctlusr - to execute the user space application.