Hướng dẫn sử dụng

Đây là hướng dẫn nhỏ cho những ai muốn sử dụng driver ADS1115 trên hệ điều hành Linux 32 bit (OS bulleye).(với 64 bit sẽ gây lỗi về API của hàm trong kernel).

Để thiết lập driver, bạn cần thực hiện một số mục sau theo thứ tự:

1. **Đăng kí i2c device**

- Tải file ads1115-overlay.dts trên github về máy:

```
/dts-v1/;
/plugin/;

/ {
    compatible = "brcm,bcm2835";

    fragment@0 {
        target = <&i2c1>;
        __overlay__ {
            ads1115@48 {
                compatible = "invensense,ads1115";
                reg = <0x48>; // I2C address
                status = "okay";
                // Optional interrupt support
                // interrupt-parent = <&gpio>;
                // interrupts = <17 IRQ_TYPE_LEVEL_HIGH>; // If INT pin is connected to GPIO17
            };
        };
    };
};
```

- Tiếp theo chạy lệnh:

  sudo dtc -@ -I dts -O dtb ads1115-overlay.dts -o ads1115-overlay.dtbo

- Thêm file .dtbo đã biên dịch ra vào thư mục /boot/overlays

  sudo mv ads1115-overlay.dtbo /boot/overlays

- chỉnh sửa file config.txt trong thư mục /boot

  sudo nano /boot/config.txt

```
[pi4]
# Run as fast as firmware / board allows
arm_boost=1

[all]
dtoverlay=dwc2
enable_uart=1
dtoverlay=mpu6050-overlay
dtoverlay=ads1115-overlay

^G Help      ^O Write Out  ^W Where Is  ^K Cut       ^T Execute   ^C Location  M-U Undo    M-A Set Mark
^X Exit      ^R Read File  ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line M-E Redo    M-6 Copy
```

+ thêm vào cuối file dòng lệnh: *dtoverlay = ads1115-overlay*

- Sau cùng khởi động lại Raspberry Pi

  sudo reboot

- Kiểm tra đã thành công:

```
pi@raspberrypi:~ $ ls /sys/bus/i2c/devices/
1-0048  1-0068  i2c-1  i2c-2
pi@raspberrypi:~ $
```

+ Xuất hiện địa chỉ của ADS1115 là 1-0048.

2. Cài đặt driver

- Bước 1: Tạo Makefile

```
obj-m += driver_ads1115.o
KDIR = /lib/modules/$(shell uname -r)/build

all:
    make -C $(KDIR) M=$(shell pwd) modules
clean:
    make -C $(KDIR) M=$(shell pwd) clean
```

- Bước 2 Chạy Makefile
- Bước 3: Nạp module bằng lệnh: sudo insmod ads1115_driver.ko

3. Giới thiệu chung các hàm cơ bản

- khai báo các biến và struct dùng trong driver:

```
static struct i2c_client *ads1115_client;
static int major_number;
static struct class *ads1115_class = NULL;
static struct device *ads1115_device = NULL;

//biến giao tiếp toàn cục giữa write-read
static int config[3];
```

+ Static struct i2c_client *ads1115_client:

```c
//biến khởi tạo khi có thấy địa chỉ tương ứng
static struct i2c_driver ads1115_driver = {
    .driver = {
        .name = DRIVER_NAME,                // tên driver (phải khớp với device tree)
        .owner  = THIS_MODULE,
    },
    .probe = ads1115_probe,
    .remove = ads1115_remove,
    .id_table = ads1115_id,                 // cho non-DT hệ thống
};
```

- Cấu trúc này khai báo các hàm khởi tạo (kiểm tra thiết bị và gán driver) và xóa driver khi không dùng đến thông qua hàm __exit()
- Đăng kí ID riêng trong kernel bằng:

```c
static const struct i2c_device_id ads1115_id[] = {

    { "ads1115", 0 }, //đăng kí id trống

    { }

};

MODULE_DEVICE_TABLE(i2c, ads1115_id);
```

+ static struct file_operations fops():

```c
//biến điều khiển cho character device
static struct file_operations fops = {
    .owner = THIS_MODULE,
    .open = ads1115_open,
    .read = ads1115_read,
    .write = ads1115_write,
    .release = ads1115_release,
};
```

- Cấu trúc này khai báo các hàm mà người dùng có thể thao tác trên lớp user space.

- Gửi giá trị value đến thanh ghi reg của client

```c
static int ads1115_write_register(struct i2c_client *client,u8 reg,u16 value){
    // Gửi 3 byte đến thiết bị: [0x01][MSB][LSB] để cấu hình ADS1115
    int ret = i2c_smbus_write_word_data(client, reg, cpu_to_be16(value));
    if (ret < 0) {
        printk(KERN_ERR "ads1115: Failed to write to reg=0x%02x val=0x%04x\n", reg, value);
        return -EIO;
    }
    return 0;
}
```

Nếu gửi không thành công (ret < 0) thì in ra lỗi và trả về -EIO

- Hàm nhận dữ liệu từ User Space và truyền thông số thiết lập đến IC ADS1115

```c
static ssize_t ads1115_write(struct file *file, const char __user *buf, size_t len, loff_t *offset)
{

    char kbuf[32];  // Buffer trong kernel space
    ssize_t ret;
    u8 reg;
    u16 value;
    // Kiểm tra độ dài
    if (len > sizeof(kbuf)) {
        printk(KERN_ERR "Data size too large\n");
        return -EINVAL;  // Trả về lỗi nếu dữ liệu quá lớn
    }
    // Kiểm tra đúng 3 byte dữ liệu để ghi cấu hình
    if (len != 3) {
        printk(KERN_ERR "ads1115: Expected 3 bytes: register + 2 config bytes\n");
        return -EINVAL;
    }
    // Copy dữ liệu từ user space vào kernel space
    ret = copy_from_user(kbuf, buf, len);
    if (ret) {
        printk(KERN_ERR "Failed to copy data from user\n");
        return -EFAULT;  // Trả về lỗi nếu copy thất bại
    }
    //copy thành biến toàn cục
    config[0] = kbuf[0];
    config[1] = kbuf[1];
    config[2] = kbuf[2];
    reg = kbuf[0];
    value = (kbuf[1] << 8) | kbuf[2];  // Ghép MSB và LSB thành 16-bit value

    // xử lý cấu hình thanh ghi
    printk(KERN_INFO "ads1115: Configuration written: reg=0x%02x val=0x%04x\n", reg, value);
    ads1115_write_register(ads1115_client,reg,value);
    return len;
}
```

Hàm nhận một giá trị buffer từ User space, sau đó gán giá trị của biến global config bằng giá trị của buffer vừa nhận. Nếu kích thước buffer khác 3 thì sẽ báo lỗi -EINVAL. Nếu copy thất bại thì sẽ báo lỗi -EFAULT.
Trong đó byte đầu tiên là byte địa chỉ thanh ghi của ADS1115, hai byte còn lại là giá trị để thiết lập thanh ghi. Trong sample_code.c có bao gồm code mẫu để tạo ra buffer bên lớp User space một cách dễ dàng. Sau đó sẽ ghi những giá trị trên xuống ADS1115.

Hàm đọc giá trị từ ADS1115

```c
static ssize_t ads1115_read(struct file *file, char __user *buf, size_t len, loff_t *offset)
{
    char rbuf[32];  // Dữ liệu cần trả về từ kernel
    ssize_t ret;
    u8 reg_c;
    u16 value;
    u8 data[2];
    s16 adc_value;
    u16 check_reg;
    u8 reg = CONV_REG;;  // Conversion register

    if ((config[1] & (1<<0)) == 0){ //continue mode
        // Gửi địa chỉ thanh ghi Conversion (0x00)
        ret = i2c_master_send(ads1115_client, &reg, 1);
        if (ret < 0) {
            printk(KERN_ERR "Failed to set pointer to conversion register\n");
            return -EIO;
        }
        // Đọc 2 byte từ Conversion register
        ret = i2c_smbus_read_i2c_block_data(ads1115_client, reg, 2,data);
        if (ret < 0) {
            printk(KERN_ERR "Failed to read conversion data\n");
            return -EIO;
        }
        printk(KERN_INFO "Continous mode\n");
    }
    else {
        //bật lại bit OS
        reg_c = config[0];
        value = (config[1] << 8) | config[2];
        ads1115_write_register(ads1115_client,reg_c,value);

        ret = i2c_master_send(ads1115_client, &reg, 1);
        if (ret < 0) {
            printk(KERN_ERR "Failed to set pointer to conversion register\n");
            return -EIO;
        }
```

```c
        // Đọc 2 byte từ Conversion register
        ret = i2c_smbus_read_i2c_block_data(ads1115_client, reg, 2,data);
        if (ret < 0) {
            printk(KERN_ERR "Failed to read conversion data\n");
            return -EIO;
        }
        printk(KERN_INFO "Single mode\n");
    }

    adc_value = (data[0] << 8) | data[1];
    memcpy(rbuf, &adc_value, sizeof(adc_value));
    // Nếu user không yêu cầu đọc gì, trả về 0
    if (*offset == 0) {
        ret = copy_to_user(buf, rbuf, strlen(rbuf) + 1);  // +1 để bao gồm ký tự NULL kết thúc chuỗi
        if (ret) {
            printk(KERN_ERR "Failed to copy data to user\n");
            return -EFAULT;  // Trả về lỗi nếu copy thất bại
        }

        *offset += strlen(rbuf) + 1;  // Cập nhật offset
        return strlen(rbuf) + 1;  // Trả về số byte đã đọc
    }


    ret = i2c_smbus_read_i2c_block_data(ads1115_client, CONFIG_REG, 2,data);
    if (ret < 0) {
        printk(KERN_ERR "Failed to read conversion data\n");
        return -EIO;
    }
    check_reg = (data[0] << 8) | data[1];
    printk(KERN_INFO "ads1115: Configuration written: reg=0x%02x val=0x%04x\n", CONFIG_REG, check_reg);

    return 0;  // Nếu không có dữ liệu để đọc nữa
}
```

- Khởi tạo các biến cục cho việc xử lý data gửi lên lớp user space
- Tiếp theo cấu trúc if-else để kiểm tra bit MODE mà người dùng truyền vào để đọc giá trị đúng cho từng chế độ là Continuous mode hay Single mode.
- Xử lý 2 byte nhận về từ hàm i2c_smbus_read_i2c_block_data().
- Hàm memcpy() sẽ thực hiện trả về giá trị nhị phân (s16, 2 byte).
- Sau cùng là truyền lên lớp user space bằng hàm copy_to_user()
- Trong hàm này luôn có các câu lệnh điều kiện để kiểm tra hàm thực thi, nếu thất bại sẽ báo lỗi lên kernel log.

Hàm ads1115_probe():

```c
static int ads1115_probe(struct i2c_client *client, const struct i2c_device_id *id) {
    printk(KERN_INFO "ADS1115 at address 0x%02x on bus %d\n", client->addr, client->adapter->nr);
    ads1115_client = client;
    //cấu hình đầu tiên cho ads1115 ở dây
    if (!ads1115_client) {
        dev_err(&client->dev, "ADS1115 client is NULL\n");
        return -ENODEV;  // Trả về lỗi nếu client bị null
    }
    printk(KERN_INFO "ADS1115 driver installed\n");
    return 0;
}
```

- In ra thông tin thiết bị I2C tìm thấy, gán nó bằng một biến global. Nếu giá trị của biến là NULL thì sẽ báo lỗi -ENODEV. Nếu tìm thấy và kết nối được sẽ thông báo đã cài đặt.

Hàm ads1115_init():

```c
static int __init ads1115_init(void)
{
    printk(KERN_INFO "Initializing ADS1115 driver\n");
    // Đăng ký character device
    major_number = register_chrdev(0, DRIVER_NAME, &fops);
    if (major_number < 0) {
        printk(KERN_ERR "Failed to register char device\n");
        return major_number;
    }
    // Tạo class
    ads1115_class = class_create(THIS_MODULE, DRIVER_NAME);
    if (IS_ERR(ads1115_class)) {
        unregister_chrdev(major_number, DRIVER_NAME);
        printk(KERN_ALERT "Failed to register device class\n");
        return PTR_ERR(ads1115_class);
    }
    // Tạo device trong /dev
    ads1115_device = device_create(ads1115_class, NULL, MKDEV(major_number, 0), NULL, DRIVER_NAME);
    if (IS_ERR(ads1115_device)) {
        class_destroy(ads1115_class);
        unregister_chrdev(major_number, DRIVER_NAME);
        printk(KERN_ALERT "Failed to create the device\n");
        return PTR_ERR(ads1115_device);
    }
    printk(KERN_INFO "ADS1115 device registered correctly\n");
    return i2c_add_driver(&ads1115_driver);  //register driver in kernel but it run on i2c base
}
```

- Hàm này sẽ được chạy đầu tiên khi load driver vào kernel.
- Thực hiện đăng kí character device cho phép user space thao tác với driver thông qua hàm read write.

- Đăng kí i2c bằng i2c_add_driver();
- Các khối if luôn kiểm tra từng bước trong quá trình đăng kí character device.

Hàm ads1115_exit():

```
static void __exit ads1115_exit(void)
{
    printk(KERN_INFO "Exiting ADS1115 driver\n");
    i2c_del_driver(&ads1115_driver);

    device_destroy(ads1115_class, MKDEV(major_number, 0));
    class_unregister(ads1115_class);
    class_destroy(ads1115_class);
    unregister_chrdev(major_number, DRIVER_NAME);
    printk(KERN_INFO "ADS1115 device unregistered\n");

}
```

- In ra thông báo "Exiting ADS1115 driver"
- Xoá driver ADS1115 khỏi hệ thống
- Xoá device đã tạo trước đó
- Huỷ đăng ký class khỏi hệ thống
- Giải phóng bộ nhớ và huỷ class đã được cấp
- Gỡ bỏ character device
- In ra thông báo “ADS1115 device unregistered”

Hàm thiết lập trong code mẫu:

```
uint16_t ADS1115_init(char PIN_P,char PIN_N, float PGA, char MODE, uint16_t DATA_RATE, char MODE_COMPARATOR, uint8_t NUMBER_CONVERSION)
{
    uint16_t Config =0;

    uint8_t pin=0;
    if(PIN_P == '0' && PIN_N == '1') pin=0;
    else if(PIN_P == '0' && PIN_N == '3') pin=1;
    else if(PIN_P == '1' && PIN_N == '3') pin=2;
    else if(PIN_P == '2' && PIN_N == '3') pin=3;
    else if(PIN_P == '0' && PIN_N == 'G') pin=4;
    else if(PIN_P == '1' && PIN_N == 'G') pin=5;
    else if(PIN_P == '2' && PIN_N == 'G') pin=6;
    else if(PIN_P == '3' && PIN_N == 'G') pin=7;

    uint8_t pga=0;
    if(PGA > 6) pga =0;
    else if(PGA > 4) pga =1;
    else if(PGA > 2) pga =2;
    else if(PGA > 1) pga =3;
    else if(PGA > 0.5) pga =4;
    else pga = 5;

    uint8_t mode=1;
    if(MODE == 'C') mode =0;
    else mode =1;

    uint8_t data_rate=0b0100;
    if(DATA_RATE==860) data_rate=0b111;
    else if(DATA_RATE==475) data_rate=0b110;
    else if(DATA_RATE==250) data_rate=0b101;
    else data_rate = log2(DATA_RATE/8);

    uint8_t mode_comparator =0;
    if(MODE_COMPARATOR == 'W') mode_comparator =1;
```

```
    uint8_t mode_comparator =0;
    if(MODE_COMPARATOR == 'W') mode_comparator =1;

    uint8_t number = NUMBER_CONVERSION;

    Config |= (1<<15) | (pin<<12) | (pga<<9) | (mode << 8) | (data_rate << 5) | (mode_comparator <<4)|(number <<0);
    printf("%d\n",Config);
    return Config;
}
```

Với cái thông số:

- PIN_P: là ký tự ‘0’, ‘1’,’2’ hoặc ‘3’ thiết lập chân AIN_P tương ứng với A0,A1,A2,A3
- PIN_N: là ký tự ‘1’,’3’ hoặc ‘G’ thiết lập chân AIN_N tương ứng với A1,A3 hoặc GND
- PGA: là giá trị Full Scale Range
- MODE: là ký tự ‘C’ hoặc ‘S’ thiết lập chế độ Continous hay Single-Shot
- DATA_RATE: là tốc độ truyền dữ liệu gồm: 8, 16, 32, 64, 128, 250, 475, 860
- MODE_COMPARATOR: là ký tự ‘W’ hoặc ‘T’ thiết lập chế độ ngắt của chân ALERT là Window hay Tranditional
- NUMBER_CONVERSION: gồm các số từ 0 - 3, thiết lập số lượng quá trình đọc giá trị để so sánh.

Hàm sẽ trả về một số nguyên không dấu 16 bit. VD: thiết lập đo giữa chân AIN0 và GND, Continous Mode, ±4.096V, 128SPS, Assert after one conversion kèm theo bit 1 ở bit số 15 để bắt đầu convertion thì giá trị của hàm trả ra sẽ là: 0b1100 0010 1000 0000.