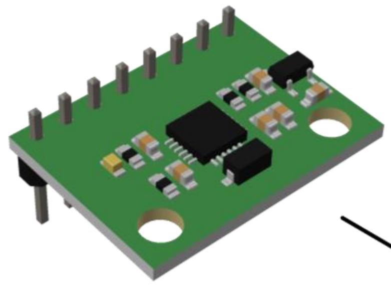
Driver Linux
Sensor MPU6050

Implementación de manejadores de dispositivos

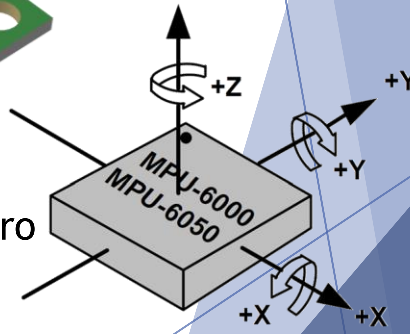
Bach Carlos herrera

Sensor MPU 6050

- Dispositivo de seguimiento de movimiento
- Posee 6 grados de libertad que combina un giroscopio de 3 ejes, un acelerómetro de 3 ejes y un Procesador de Movimiento Digital™ (DMP).



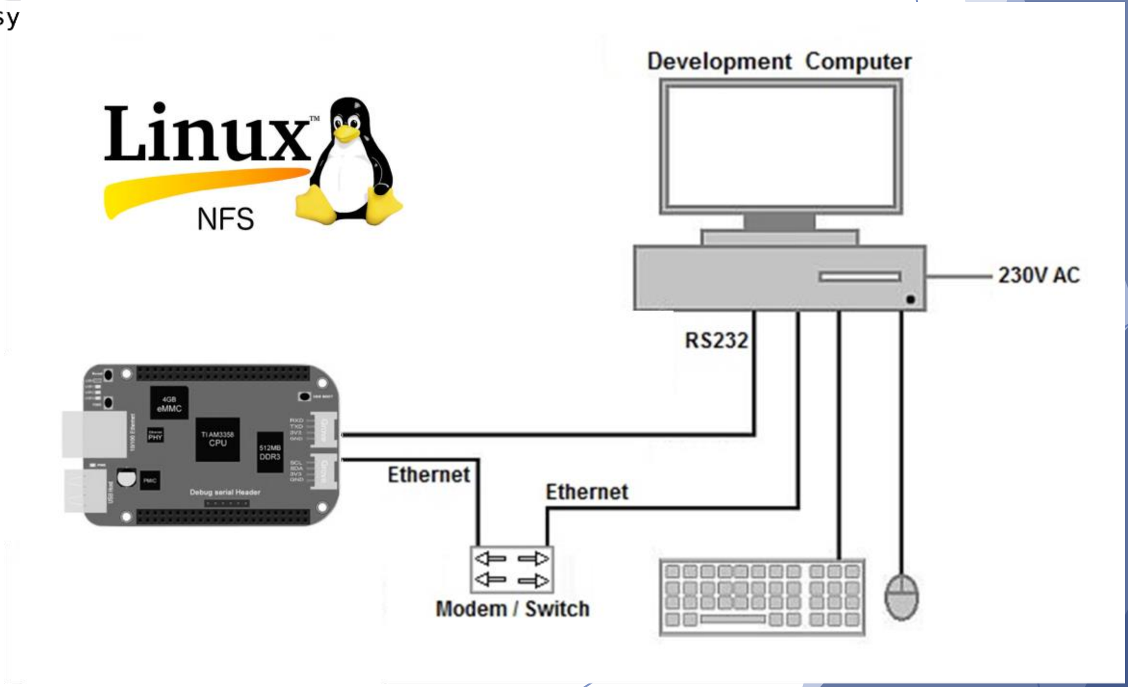
- Comunicación I^{2}C a 400kHz



- Resolución de 16 bits
- Escalas programables para giroscopio (±250, ±500, ±1000, and ±2000°/sec) y acelerómetro (±2*g*, ±4*g*, ±8*g*, and ±16*g*)
- Dirección I^{2}C depende del pin AD0 (AD0=0 address=0x68, **AD0=1 address=0x69**)

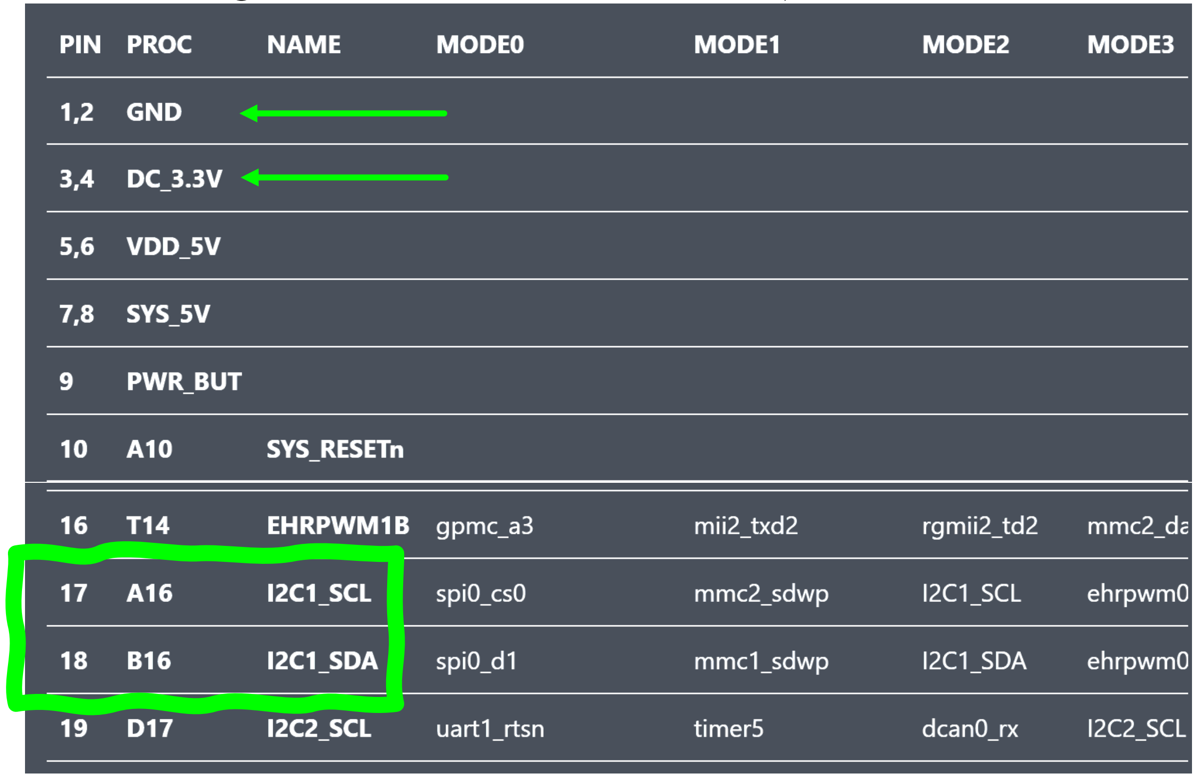
Entorno de desarrollo - Buildroot

- Framework Buildroot
- Kernel Linux 6.6.22
- Toolchain integrado de Buildroot
- Servidor TFTP y NFS

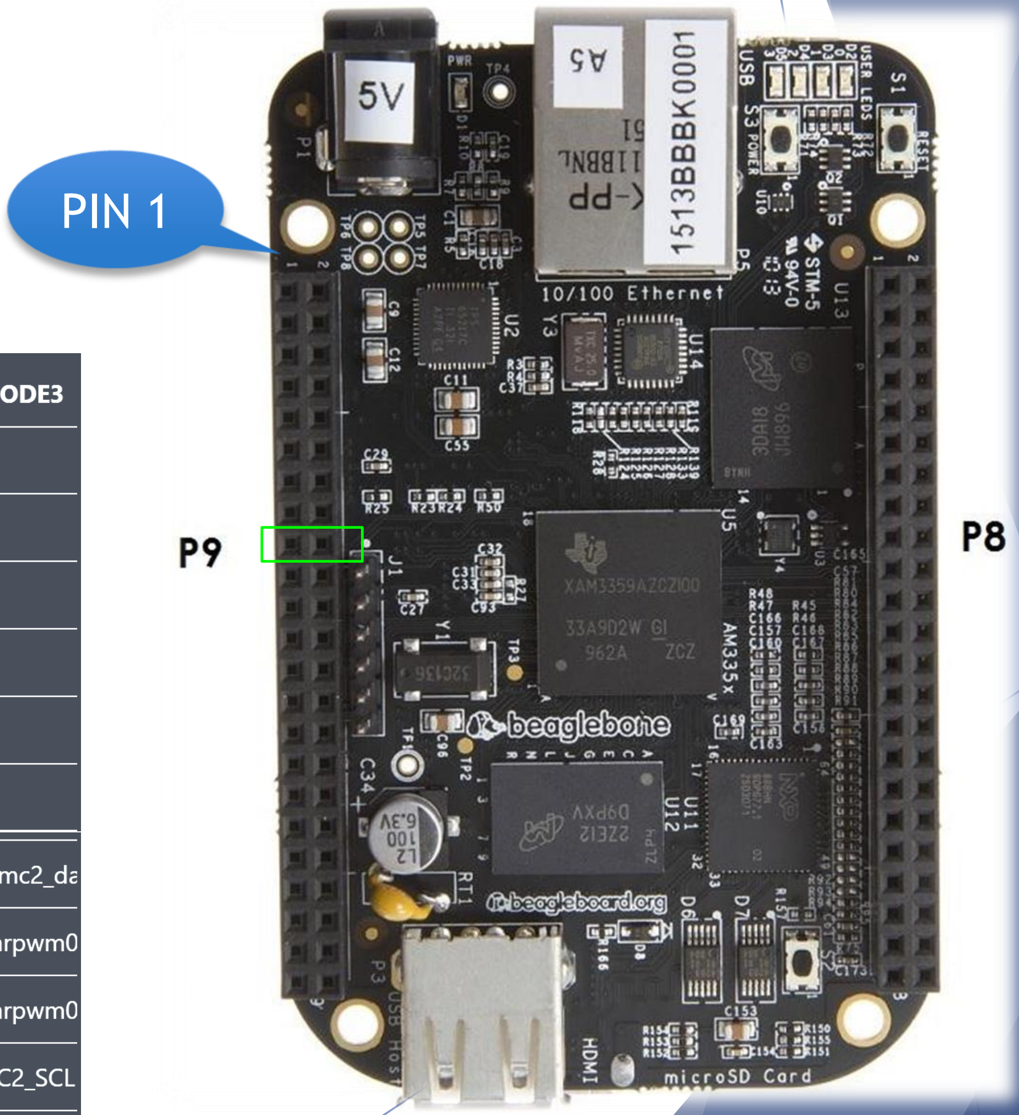


Beaglebone Black I2C Hardware

- Pines I2C1 en el conector P9 (https://docs.beagleboard.org/latest/boards/beaglebone/black/ch07.html)

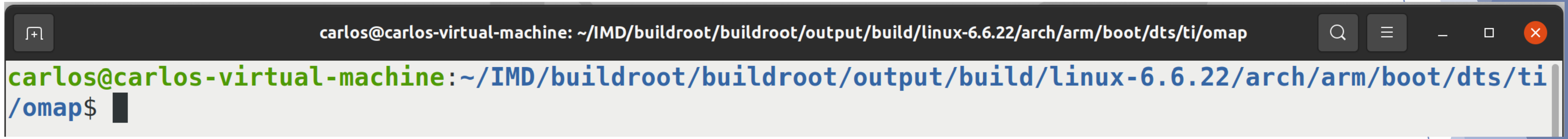


PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3
1,2	GND					
3,4	DC_3.3V					
5,6	VDD_5V					
7,8	SYS_5V					
9	PWR_BUT					
10	A10	SYS_RESETn				
16	T14	EHRPWM1B	gpmc_a3	mii2_txd2	rgmii2_td2	mmc2_da
17	A16	I2C1_SCL	spi0_cs0	mmc2_sdwp	I2C1_SCL	ehrpwm0
18	B16	I2C1_SDA	spi0_d1	mmc1_sdwp	I2C1_SDA	ehrpwm0
19	D17	I2C2_SCL	uart1_rtsn	timer5	dcan0_rx	I2C2_SCL



Modificación del Device Tree

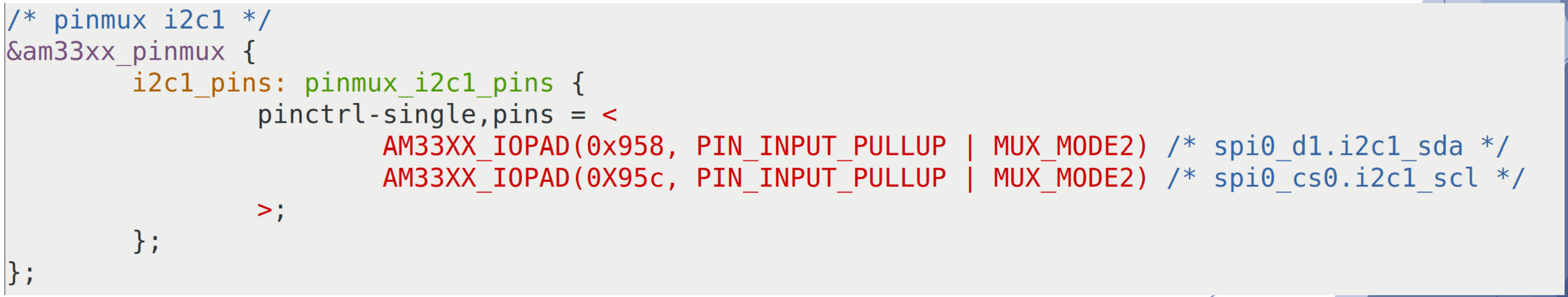
- Se realiza una copia con el nombre **`MSE_IMD_TPF-boneblack.dts`** del DTS original del BBB (`am335x-boneblack.dts`) ubicado en la siguiente ruta.



carlos@carlos-virtual-machine: ~/IMD/buildroot/buildroot/output/build/linux-6.6.22/arch/arm/boot/dts/ti/omap

```
carlos@carlos-virtual-machine:~/IMD/buildroot/buildroot/output/build/linux-6.6.22/arch/arm/boot/dts/ti
/omap$ 
```

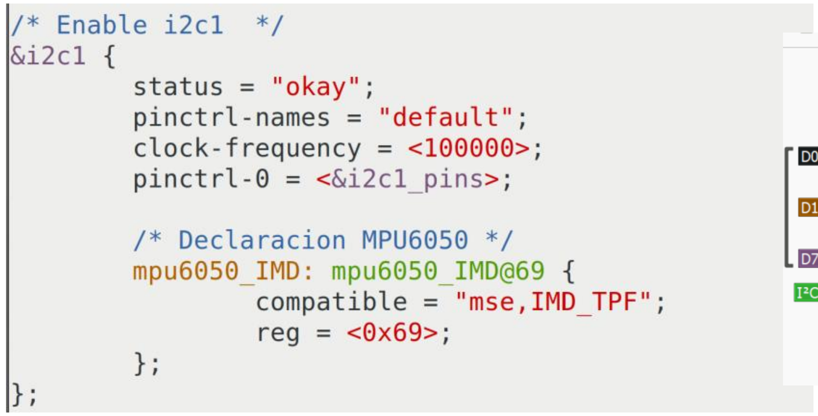
- Se adicionan la especificación de pines del **i2c1**



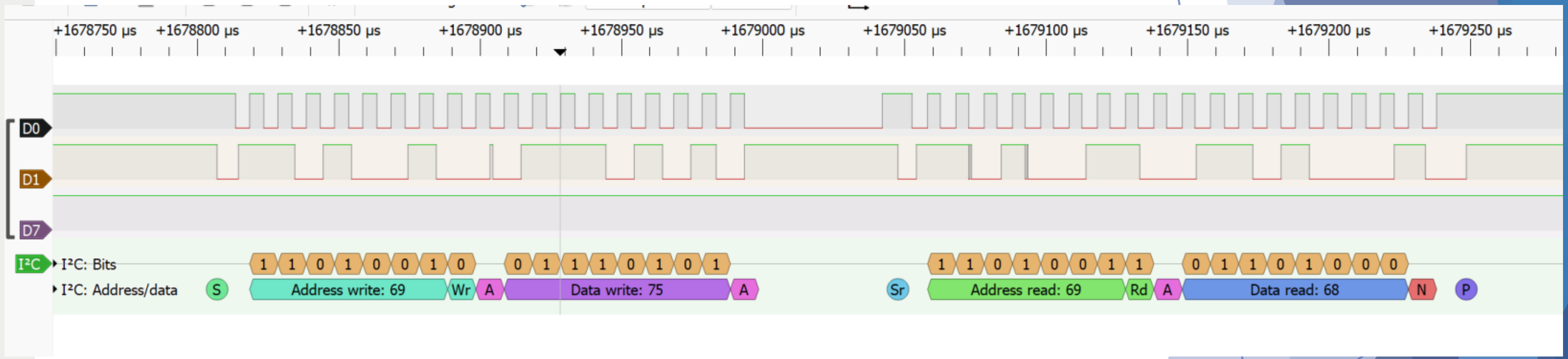
```
/* pinmux i2c1 */
&am33xx_pinmux {
        i2c1_pins: pinmux_i2c1_pins {
                pinctrl-single,pins = <
                        AM33XX_IOPAD(0x958, PIN_INPUT_PULLUP | MUX_MODE2) /* spi0_d1.i2c1_sda */
                        AM33XX_IOPAD(0X95c, PIN_INPUT_PULLUP | MUX_MODE2) /* spi0_cs0.i2c1_scl */
                >;
        };
};
```

Modificación del Device Tree

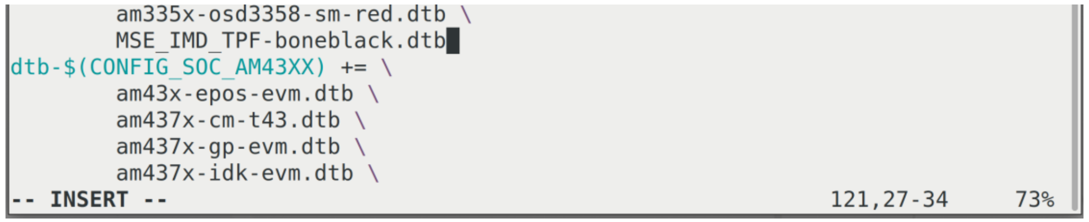
- Se agrega el nodo **i2c1** (disponible) con velocidad de 400kHz y el subnodo **mpu6050_IMD** con dirección 0x69h



```
/* Enable i2c1  */
&i2c1 {
        status = "okay";
        pinctrl-names = "default";
        clock-frequency = <100000>;
        pinctrl-0 = <&i2c1_pins>;

        /* Declaracion MPU6050 */
        mpu6050_IMD: mpu6050_IMD@69 {
                compatible = "mse,IMD_TPF";
                reg = <0x69>;
        };
};
```



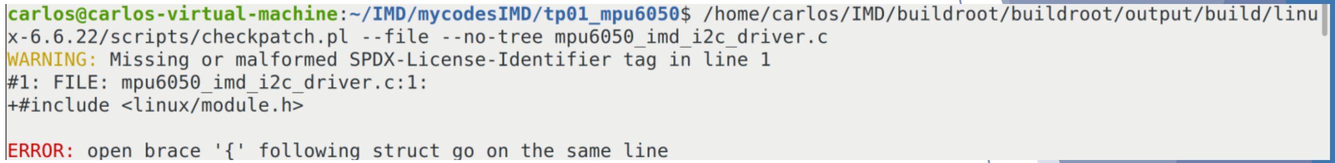
- En el Makefile de la carpeta del device tree se adiciona el archivo el correspondiente archivo .dtb



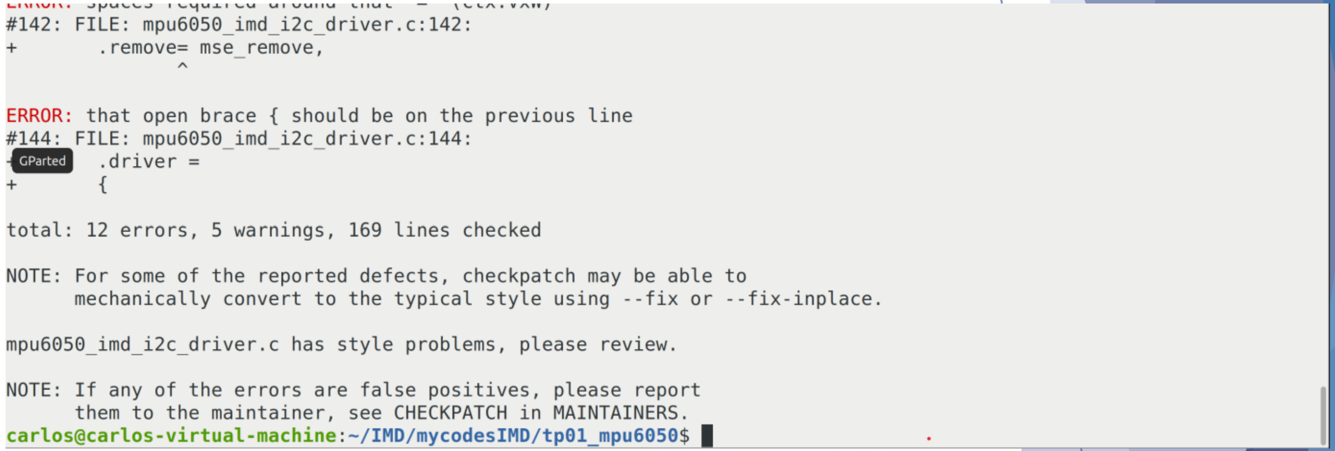
```
        am335x-osd3358-sm-red.dtb \
        MSE_IMD_TPF-boneblack.dtb
dtb-$(CONFIG_SOC_AM43XX) += \
        am43x-epos-evm.dtb \
        am437x-cm-t43.dtb \
        am437x-gp-evm.dtb \
        am437x-idk-evm.dtb \
-- INSERT --                                121,27-34      73%
```

Módulo Driver - Identación

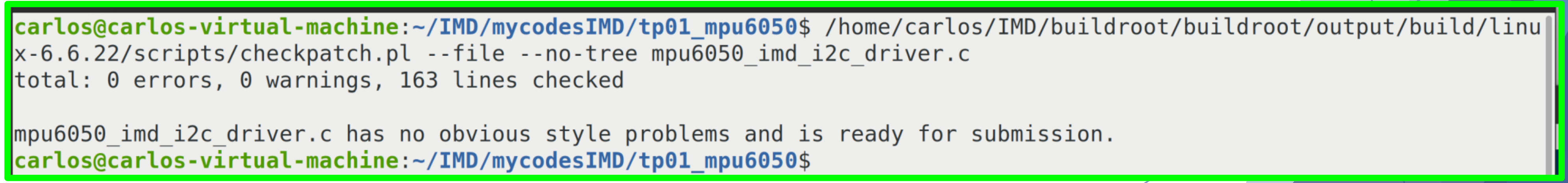
- En primer lugar, se realiza una revisión de la identación del código driver base.



```
carlos@carlos-virtual-machine:~/IMD/mycodesIMD/tp01_mpu6050$ /home/carlos/IMD/buildroot/buildroot/output/build/linu
x-6.6.22/scripts/checkpatch.pl --file --no-tree mpu6050_imd_i2c_driver.c
WARNING: Missing or malformed SPDX-License-Identifier tag in line 1
#1: FILE: mpu6050_imd_i2c_driver.c:1:
+#include <linux/module.h>

ERROR: open brace '{' following struct go on the same line
```



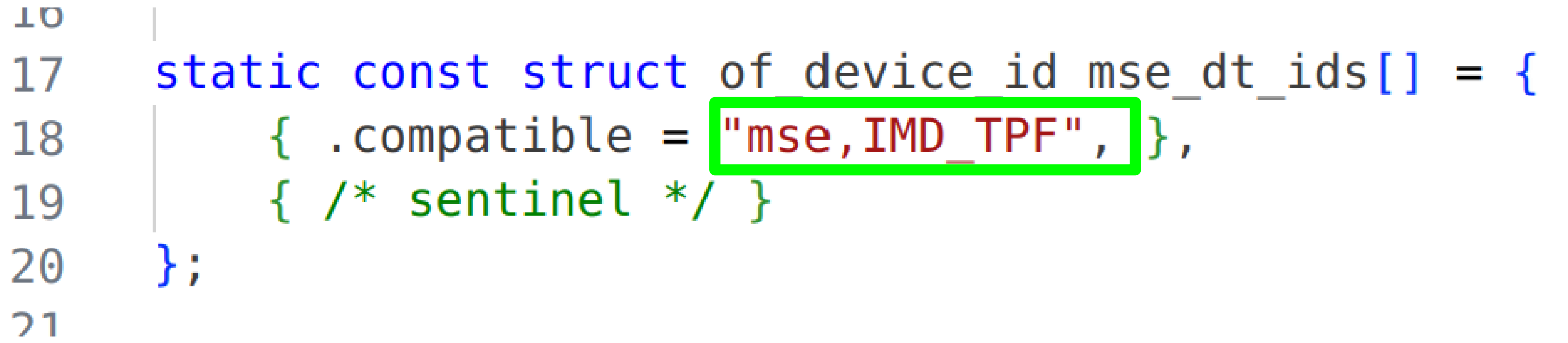
```
#142: FILE: mpu6050_imd_i2c_driver.c:142:
+       .remove= mse_remove,
                ^

ERROR: that open brace { should be on the previous line
#144: FILE: mpu6050_imd_i2c_driver.c:144:
+       .driver =
+       {

total: 12 errors, 5 warnings, 169 lines checked

NOTE: For some of the reported defects, checkpatch may be able to
      mechanically convert to the typical style using --fix or --fix-inplace.

mpu6050_imd_i2c_driver.c has style problems, please review.

NOTE: If any of the errors are false positives, please report
      them to the maintainer, see CHECKPATCH in MAINTAINERS.
carlos@carlos-virtual-machine:~/IMD/mycodesIMD/tp01_mpu6050$
```



```
carlos@carlos-virtual-machine:~/IMD/mycodesIMD/tp01_mpu6050$ /home/carlos/IMD/buildroot/buildroot/output/build/linu
x-6.6.22/scripts/checkpatch.pl --file --no-tree mpu6050_imd_i2c_driver.c
total: 0 errors, 0 warnings, 163 lines checked

mpu6050_imd_i2c_driver.c has no obvious style problems and is ready for submission.
carlos@carlos-virtual-machine:~/IMD/mycodesIMD/tp01_mpu6050$
```

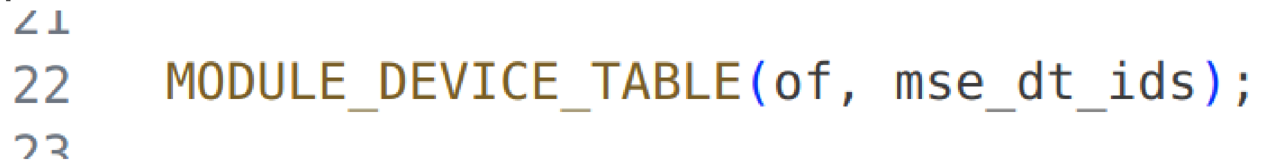
Módulo Driver – Device Tree (Open Firmware OF)

- Se define las propiedades necesarias para el devicetree, en este caso, el string compatible "**mse,IMD_TPF“**



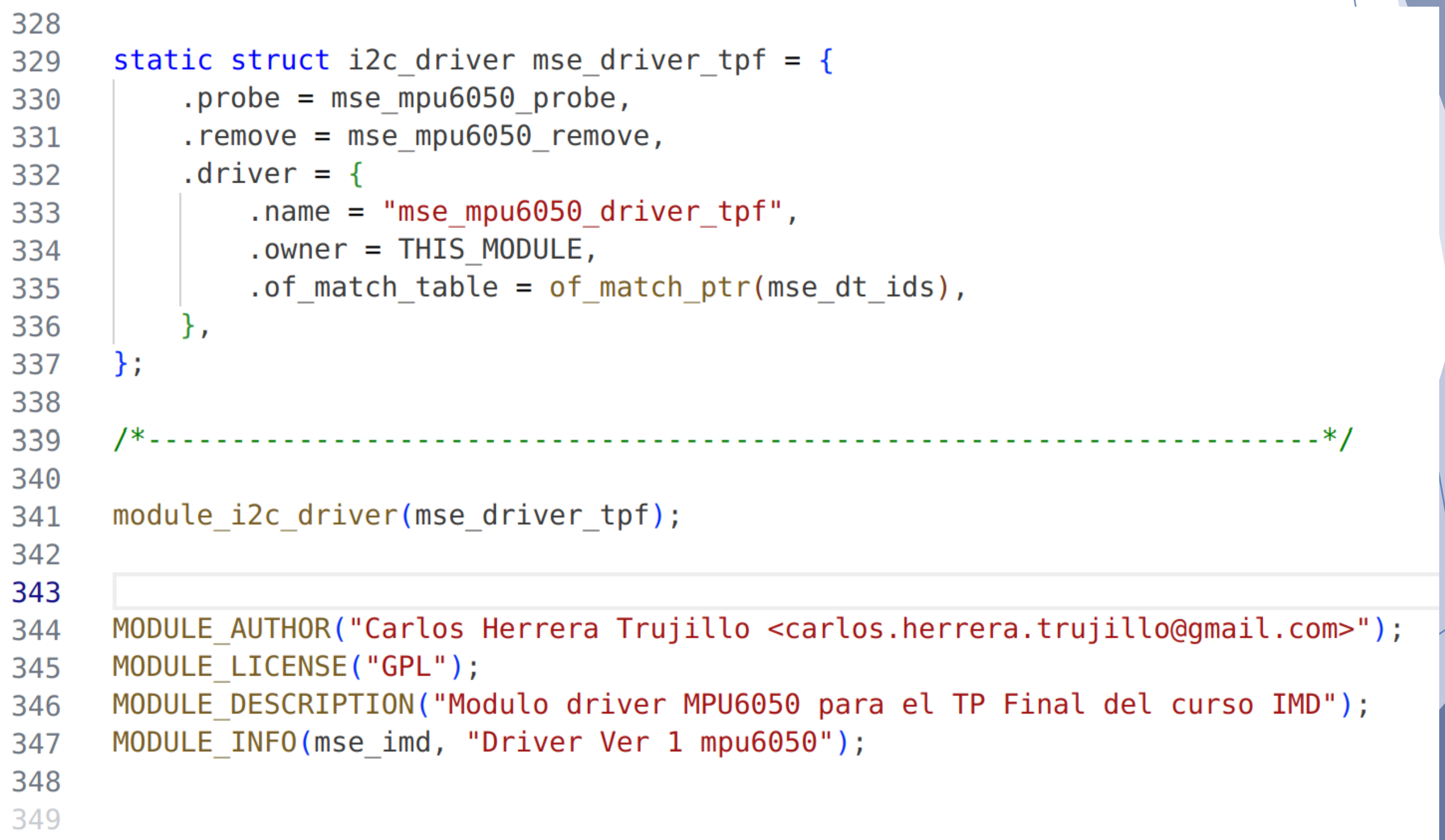
```
static const struct of_device_id mse_dt_ids[] = {
    { .compatible = "mse,IMD_TPF", },
    { /* sentinel */ }
};
```

- Luego se informa al Kernel mediante la macro MODULE_DEVICE_TABLE la tabla del tipo “**of**”



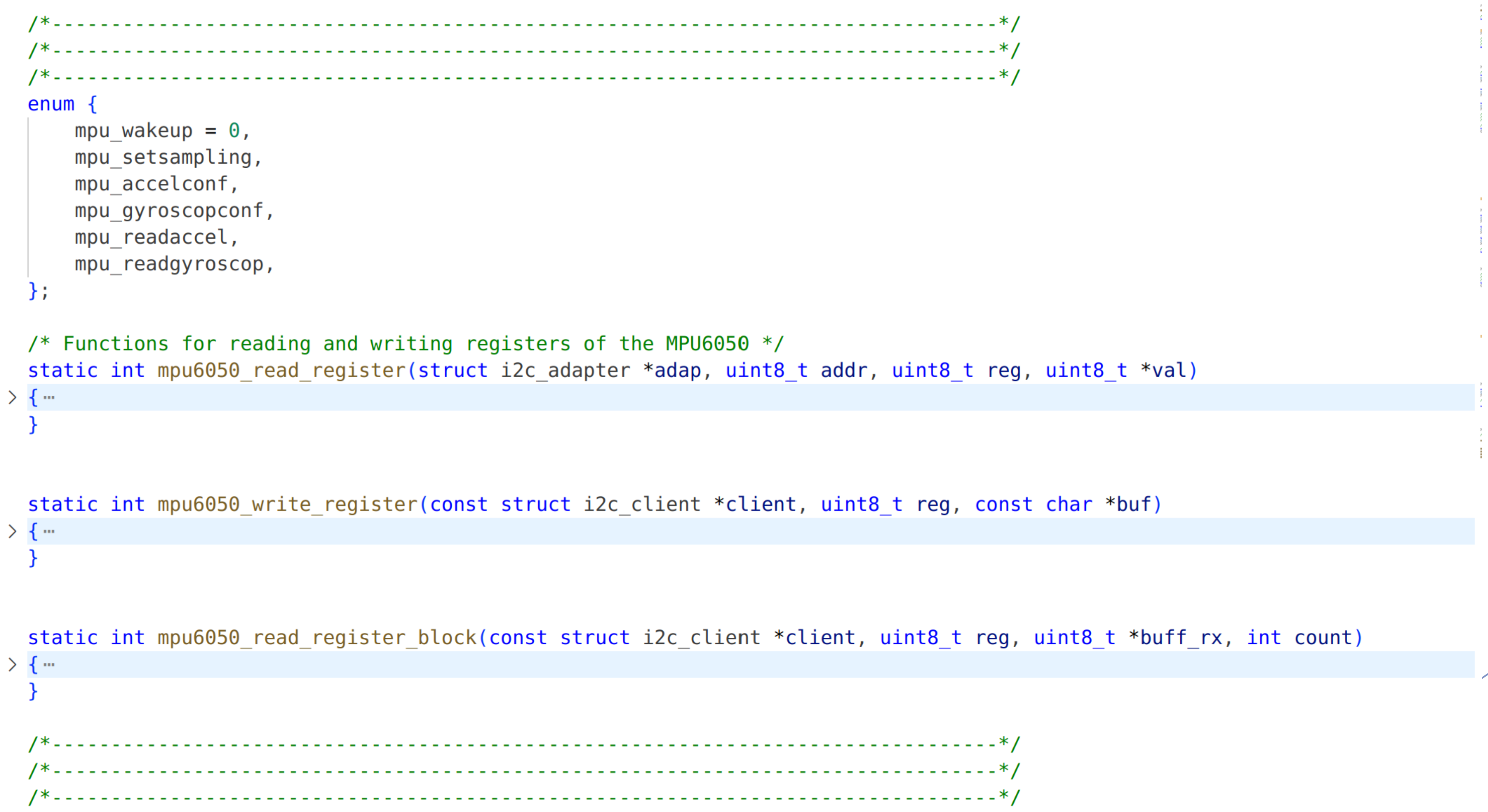
```
MODULE_DEVICE_TABLE(of, mse_dt_ids);
```

Driver I2C



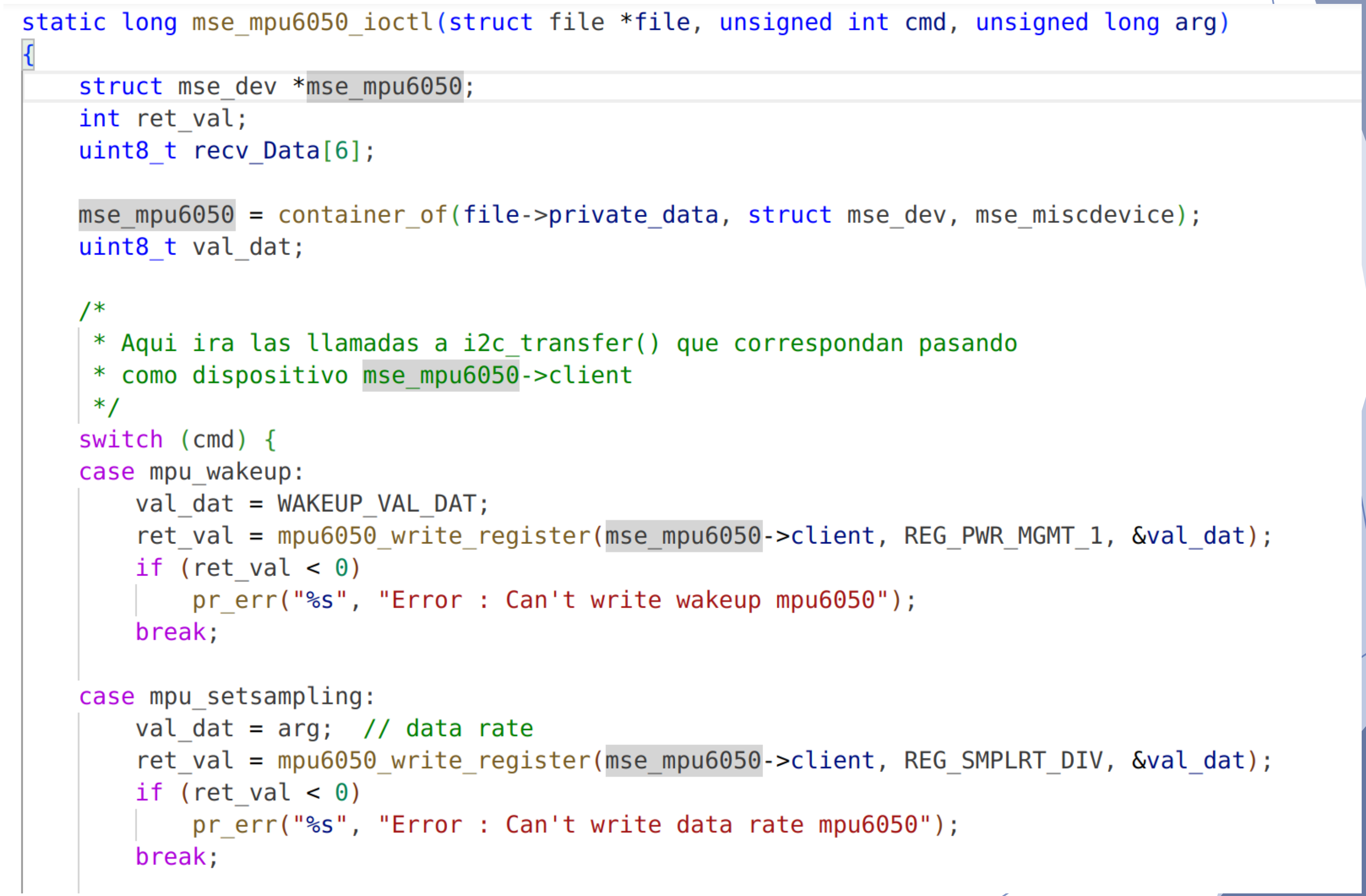
```c

static struct i2c_driver mse_driver_tpf = {
    .probe = mse_mpu6050_probe,
    .remove = mse_mpu6050_remove,
    .driver = {
        .name = "mse_mpu6050_driver_tpf",
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(mse_dt_ids),
    },
};

/*----------------------------------------------------------------------*/

module_i2c_driver(mse_driver_tpf);


MODULE_AUTHOR("Carlos Herrera Trujillo <carlos.herrera.trujillo@gmail.com>");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Modulo driver MPU6050 para el TP Final del curso IMD");
MODULE_INFO(mse_imd, "Driver Ver 1 mpu6050");

```

MPU6050 write read



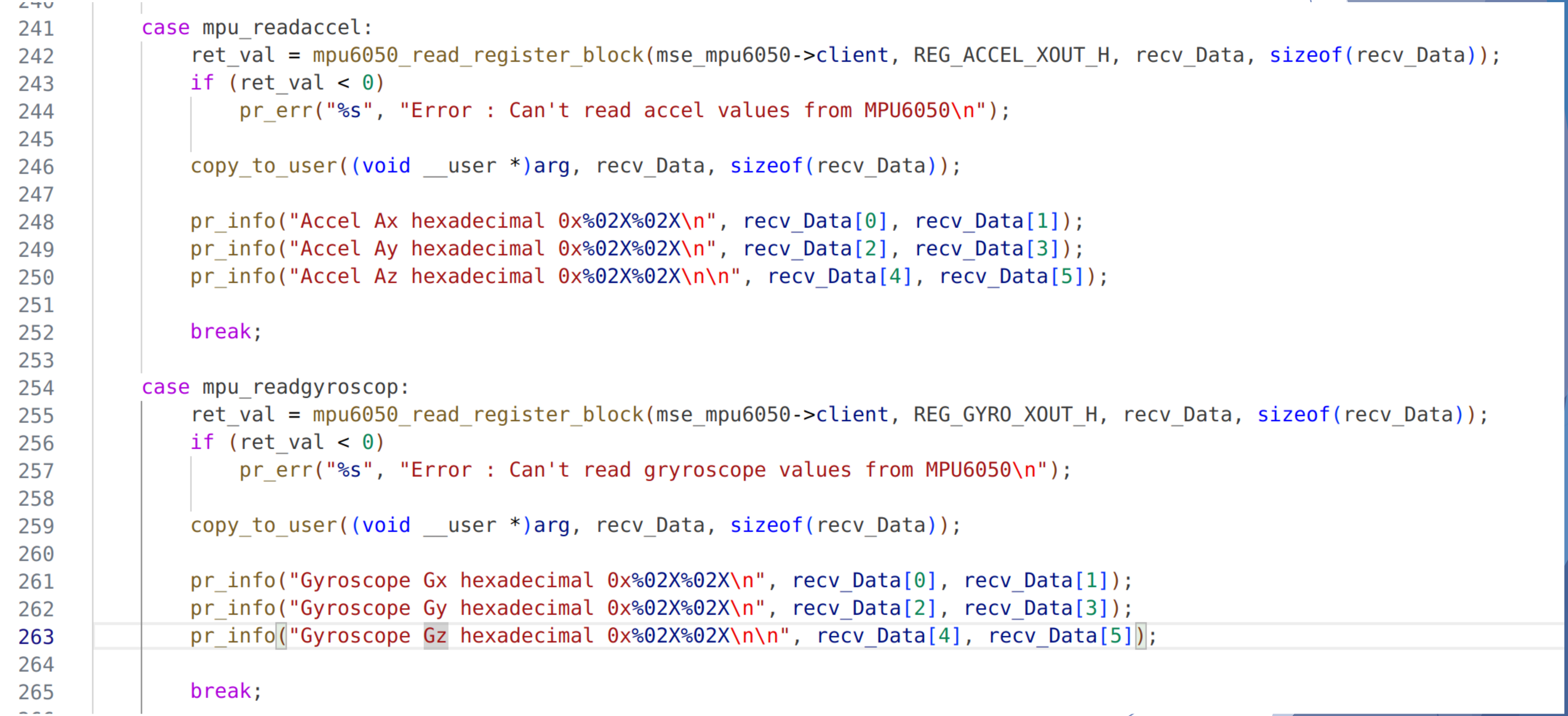
```
/*--------------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------------*/
enum {
    mpu_wakeup = 0,
    mpu_setsampling,
    mpu_accelconf,
    mpu_gyroscopconf,
    mpu_readaccel,
    mpu_readgyroscop,
};

/* Functions for reading and writing registers of the MPU6050 */
static int mpu6050_read_register(struct i2c_adapter *adap, uint8_t addr, uint8_t reg, uint8_t *val)
{…
}


static int mpu6050_write_register(const struct i2c_client *client, uint8_t reg, const char *buf)
{…
}


static int mpu6050_read_register_block(const struct i2c_client *client, uint8_t reg, uint8_t *buff_rx, int count)
{…
}

/*--------------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------------*/
```

Driver IOCTL



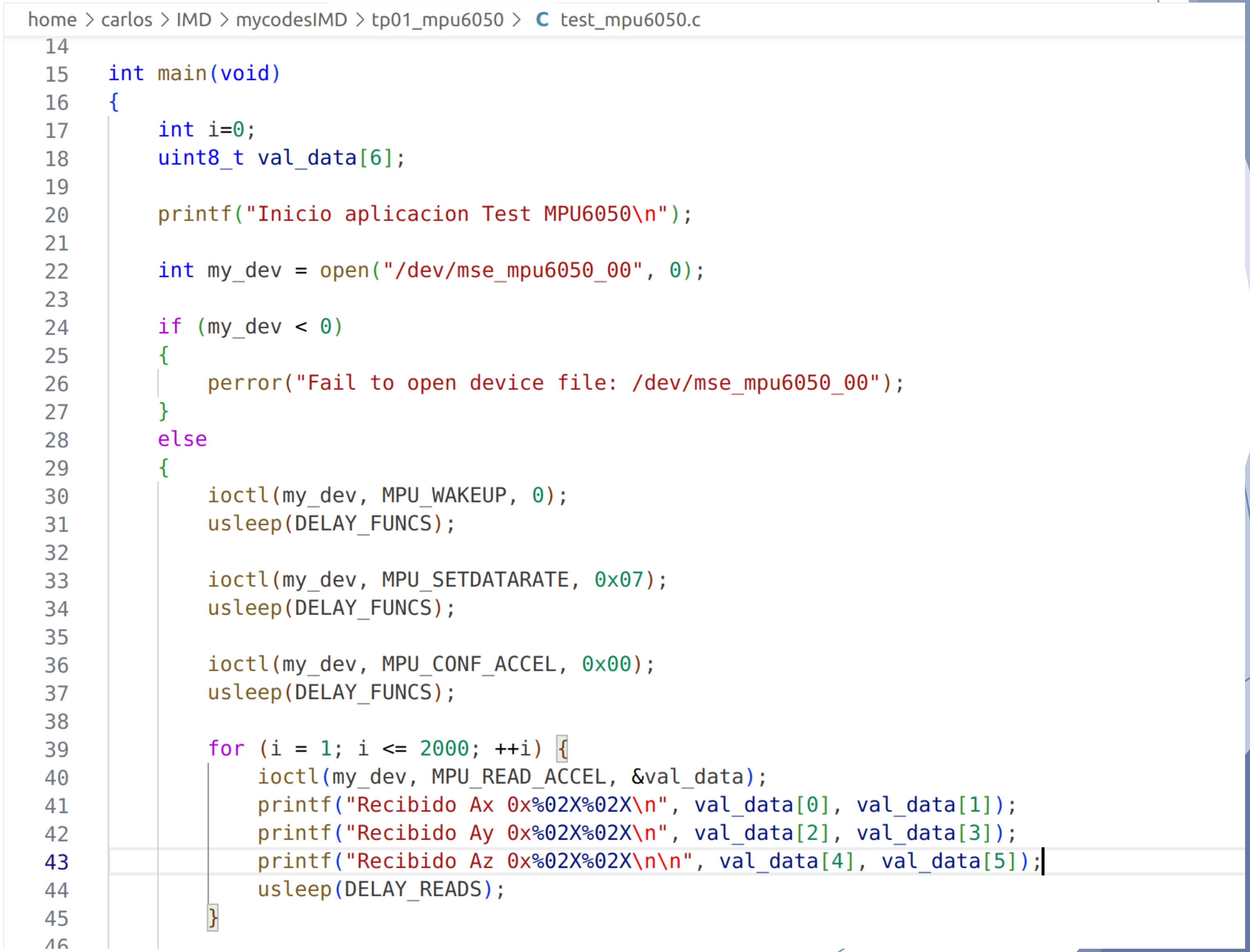
```c
static long mse_mpu6050_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
    struct mse_dev *mse_mpu6050;
    int ret_val;
    uint8_t recv_Data[6];

    mse_mpu6050 = container_of(file->private_data, struct mse_dev, mse_miscdevice);
    uint8_t val_dat;

    /*
     * Aqui ira las llamadas a i2c_transfer() que correspondan pasando
     * como dispositivo mse_mpu6050->client
     */
    switch (cmd) {
    case mpu_wakeup:
        val_dat = WAKEUP_VAL_DAT;
        ret_val = mpu6050_write_register(mse_mpu6050->client, REG_PWR_MGMT_1, &val_dat);
        if (ret_val < 0)
            pr_err("%s", "Error : Can't write wakeup mpu6050");
        break;

    case mpu_setsampling:
        val_dat = arg;  // data rate
        ret_val = mpu6050_write_register(mse_mpu6050->client, REG_SMPLRT_DIV, &val_dat);
        if (ret_val < 0)
            pr_err("%s", "Error : Can't write data rate mpu6050");
        break;
```

Driver IOCTL



```
    case mpu_readaccel:
        ret_val = mpu6050_read_register_block(mse_mpu6050->client, REG_ACCEL_XOUT_H, recv_Data, sizeof(recv_Data));
        if (ret_val < 0)
            pr_err("%s", "Error : Can't read accel values from MPU6050\n");

        copy_to_user((void __user *)arg, recv_Data, sizeof(recv_Data));

        pr_info("Accel Ax hexadecimal 0x%02X%02X\n", recv_Data[0], recv_Data[1]);
        pr_info("Accel Ay hexadecimal 0x%02X%02X\n", recv_Data[2], recv_Data[3]);
        pr_info("Accel Az hexadecimal 0x%02X%02X\n\n", recv_Data[4], recv_Data[5]);

        break;

    case mpu_readgyroscop:
        ret_val = mpu6050_read_register_block(mse_mpu6050->client, REG_GYRO_XOUT_H, recv_Data, sizeof(recv_Data));
        if (ret_val < 0)
            pr_err("%s", "Error : Can't read gryroscope values from MPU6050\n");

        copy_to_user((void __user *)arg, recv_Data, sizeof(recv_Data));

        pr_info("Gyroscope Gx hexadecimal 0x%02X%02X\n", recv_Data[0], recv_Data[1]);
        pr_info("Gyroscope Gy hexadecimal 0x%02X%02X\n", recv_Data[2], recv_Data[3]);
        pr_info("Gyroscope Gz hexadecimal 0x%02X%02X\n\n", recv_Data[4], recv_Data[5]);

        break;
```

Test MPU6050



home > carlos > IMD > mycodesIMD > tp01_mpu6050 > C test_mpu6050.c

```c

int main(void)
{
    int i=0;
    uint8_t val_data[6];

    printf("Inicio aplicacion Test MPU6050\n");

    int my_dev = open("/dev/mse_mpu6050_00", 0);

    if (my_dev < 0)
    {
        perror("Fail to open device file: /dev/mse_mpu6050_00");
    }
    else
    {
        ioctl(my_dev, MPU_WAKEUP, 0);
        usleep(DELAY_FUNCS);

        ioctl(my_dev, MPU_SETDATARATE, 0x07);
        usleep(DELAY_FUNCS);

        ioctl(my_dev, MPU_CONF_ACCEL, 0x00);
        usleep(DELAY_FUNCS);

        for (i = 1; i <= 2000; ++i) {
            ioctl(my_dev, MPU_READ_ACCEL, &val_data);
            printf("Recibido Ax 0x%02X%02X\n", val_data[0], val_data[1]);
            printf("Recibido Ay 0x%02X%02X\n", val_data[2], val_data[3]);
            printf("Recibido Az 0x%02X%02X\n\n", val_data[4], val_data[5]);
            usleep(DELAY_READS);
        }
```