

# 1. 04.sensor

2024/7/18 Table of Contents

04.sensor

目的

構成データ

センサー制御

デバイスドライバ

例題 asevent

課題1 asled

## 1.1. 目的

組込みアプリケーション開発 04.sensor

## 1.2. 構成データ

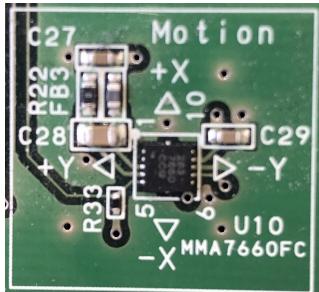
### 1.2.1. /media/sf\_ArmadilloX1/hwpwm/work/R06\_2024/Application\_debug/text/practice ディレクトリ

▼ …/share/ArmadilloX1/hwpwm/work/R06\_2024/Application\_debug/text/practice/ の構成

```
1 user@1204PC-Z490M:/mnt/v/VirtualBoxWork/share/ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice$ tree -af
2 .
3   +-- 04.sensor/
4     |   +-- asevent.c*           <---- 例題 デバイス制御用ソース
5     |   +-- asled.c*            <---- 課題1 デバイス制御用ソース
6     |   +-- drivers/
7       |   +-- acceleration/
8         |   |   +-- acceleration.c* <---- ドライバソース
9         |   |   +-- Makefile*        <---- ドライバ用Makefile
10        |   +-- leds/
11        |   +-- Makefile*          <---- デバイス制御用Makefile
12
```

## 1.3. センサー制御

### 1.3.1. デバイス仕様



## 1.4. デバイスドライバ

### 1.4.1. ソース

#### acceleration.c

▼ 04.sensor/drivers/acceleration.c

```

#include <linux/init.h>
#include <linux/i2c.h>
#include <linux/platform_device.h>
#include <linux/interrupt.h>
#include <linux/input.h>
#include <linux/jiffies.h>
#include <linux/sysfs.h>
#include <linux/workqueue.h>
#include <linux/module.h>

#define I2C_DEVICE_ADDRESS      0x4c

#define MILLI_G_FACTOR          47

#define VALUE_ALERT_BIT         0x40
#define VALUE_SING_BIT          0x20

#define VALUE_COMPLEMENTARY     (~0x1f)

#define INTSET_GINT             0x10

#define MODE_ACTIVE              0x01

#define IS_ALERT(value)          ((value) & VALUE_ALERT_BIT)
#define IS_MINUS_VALUE(value)    ((value) & VALUE_SING_BIT)

enum AccelerationRegister {
    REG_X_VALUE,
    REG_Y_VALUE,
    REG_Z_VALUE,
    REG_TILT_STATUS,
    REG_SAMPLE_RATE_STATUS,
    REG_SLEEP_COUNT,
    REG_INTERRUPT_SETTING,
    REG_MODE_SETTING,
    REG_SAMPLE_RATE,
    REG_TAP_DETECTION_THRESHOLD,
    REG_TAP_DEBOUNCE_COUNT,
};

static struct i2c_client *acceleration_i2c_client;

// I2C通信 データ受信関数(acceleration_i2c_recv_data)
static int acceleration_i2c_recv_data(enum AccelerationRegister reg_number, unsigned char *recv_buffer)
{
    // 加速度センサから1バイトデータを受信するため
    // I2Cメッセージ構造体配列を宣言します。
    struct i2c_msg msg[] = {
        {
            .addr   = I2C_DEVICE_ADDRESS,
            .flags  = 0,
            .len    = 1,
            .buf    = (unsigned char *)&reg_number,
        },
        {
            .addr   = I2C_DEVICE_ADDRESS,
            .flags  = I2C_M_RD,
            .len    = 1,
            .buf    = recv_buffer,
        },
    };
}

// I2Cメッセージを転送します。(i2c_transfer)
return i2c_transfer(acceleration_i2c_client->adapter, msg, 2);
}

// I2C通信 データ送信関数(acceleration_i2c_send_data)
static int acceleration_i2c_send_data(enum AccelerationRegister reg_number, unsigned char send_data)

```

```

{
    // 加速度センサへ送信するデータを用意します。
    unsigned char send_buffer[] = {
        reg_number,
        send_data,
    };
    // 加速度センサへ1バイトデータを送信するための
    // I2Cメッセージ構造体配列を宣言します。
    struct i2c_msg msg[] = {
        {
            .addr = I2C_DEVICE_ADDRESS,
            .flags = 0,
            .len = sizeof(send_buffer),
            .buf = send_buffer,
        }
    };
    // I2Cメッセージを転送します。(i2c_transfer)
    return i2c_transfer(acceleration_i2c_client->adapter, msg, 1);
}

static bool is_acceleration_enable = false;

// 加速度センサ有効化関数(acceleration_enable_setting)
static int acceleration_enable_setting(void)
{
    int ret;

    // 既に有効になっている場合、何もせずに降の処理を終了します。
    if (true == is_acceleration_enable)
        return 0;

    // ワークキューの処理が完了するまで待機します。(flush_scheduled_work)
    flush_scheduled_work();

    is_acceleration_enable = true;

    // 加速度センサから割込みが発生するように設定します。(acceleration_i2c_send_data)
    ret = acceleration_i2c_send_data(REG_INTERRUPT_SETTING, INTSET_GINT);
    if (ret < 0)
        return ret;

    // 加速度センサをアクティブモードに設定します。(acceleration_i2c_send_data)
    ret = acceleration_i2c_send_data(REG_MODE_SETTING, MODE_ACTIVE);
    if (ret < 0)
        return ret;

    // 加速度センサの割込みを有効にします。(enable_irq)
    // ->割込み番号は、acceleration_i2c_client->irqを指定します。
    enable_irq(acceleration_i2c_client->irq);

    return 0;
}

// 加速度センサ無効化関数(acceleration_disable_setting)
static int acceleration_disable_setting(void)
{
    int ret;

    // 既に無効になっている場合、何もせずに降の処理を終了します。
    if (false == is_acceleration_enable)
        return 0;

    is_acceleration_enable = false;

    // ワークキューの処理が完了するまで待機します。(flush_scheduled_work)
    flush_scheduled_work();

    // 加速度センサをスタンバイモードに設定します。(acceleration_i2c_send_data)

```

```

    ret = acceleration_i2c_send_data(REG_MODE_SETTING, 0);
    if (ret < 0)
        return ret;

    // 加速度センサから割込みが発生しないように設定します。(acceleration_i2c_send_data)
    ret = acceleration_i2c_send_data(REG_INTERRUPT_SETTING, 0);
    if (ret < 0)
        return ret;

    return 0;
}

// 加速度センサ有効/無効設定関数(acceleration_store_enable)
static ssize_t acceleration_store_enable(struct device *dev, struct device_attribute *attr, const char *buf, size_t count)
{
    int ret;
    unsigned long enable;

    // 文字列の有効/無効の値を数値に変換します。(kstrtol)
    ret = kstrtol(buf, 0, &enable);
    if (ret < 0)
        return ret;

    // 加速度センサの有効/無効を設定します。
    // ->"0"以外の数字が書き込まれた場合、有効にします。(acceleration_enable_setting)
    // "0"が書き込まれた場合、無効にします。(acceleration_disable_setting)
    if (enable)
        ret = acceleration_enable_setting();
    else
        ret = acceleration_disable_setting();

    if (ret < 0)
        return ret;

    return count;
}

unsigned long event_interval_ms = 1000;

// 通知間隔設定関数(acceleration_store_delay)
static ssize_t acceleration_store_delay(struct device *dev, struct device_attribute *attr, const char *buf, size_t count)
{
    int ret;
    unsigned long interval_ms;

    // 文字列の通知間隔を数値に変換します。(kstrtol)
    ret = kstrtol(buf, 0, &interval_ms);
    if (ret < 0)
        return ret;

    // 通知間隔をグローバル変数(event_interval_ms)に格納します。
    event_interval_ms = interval_ms;

    return count;
}

// 加速度値読み取り関数(read_acceleration_value)
static int read_acceleration_value(enum AccelerationRegister reg_number)
{
    int buf = 0;

    do {
        // 第1引数で指定されたレジスタからデータを受信します。(acceleration_i2c_recv_data)
        acceleration_i2c_recv_data(reg_number, (unsigned char *)&buf);

        // アラートがある場合は再度データを取得します。(IS_ALERT)
    } while(IS_ALERT(buf));

    // 加速度値が負数の場合は、2の補数として不足しているビットを補完します。
}

```

```

// ->加速度値が負数かどうかの判定は、IS_MINUS_VALUEマクロを使用します。
// 不足しているビットは、VALUE_COMPLEMENTARYで補完します。
if (IS_MINUS_VALUE(buf))
    buf |= VALUE_COMPLEMENTARY;

// レジスタから取得した値を、ミリGに変換して返却します。
// ->ミリGへの変換は、MILLI_G_FACTORを乗算します。
return buf * MILLI_G_FACTOR;
}

struct input_dev *input;

// 割込み遅延処理関数(irq_worker)
static void irq_worker(struct work_struct *work)
{
    int data_x, data_y, data_z;

    // 加速度センサが無効の場合、以降の処理を終了します。
    if (false == is_acceleration_enable)
        return;

    // 加速度センサから、加速度値X,Y,Zをそれぞれ取得します。(read_acceleration_value)
    // ->レジスタはそれぞれ、
    // REG_X_VALUE, REG_Y_VALUE, REG_Z_VALUE を指定します。
    data_x = read_acceleration_value(REG_X_VALUE);
    data_y = read_acceleration_value(REG_Y_VALUE);
    data_z = read_acceleration_value(REG_Z_VALUE);

    // X,Y,Zの値をそれぞれ通知します。(input_report_abs)
    input_report_abs(input, ABS_X, data_x);
    input_report_abs(input, ABS_Y, data_y);
    input_report_abs(input, ABS_Z, data_z);

    // 同期イベントを通知します。(input_sync)
    input_sync(input);

    // 割込みハンドラで無効にした割込みを有効化します。(enable_irq)
    // ->割込み番号は、acceleration_i2c_client->irqを指定します。
    enable_irq(acceleration_i2c_client->irq);
}

// ワークキュー定義
static DECLARE_DELAYED_WORK(irq_process, irq_worker);

// 割込みハンドラ(irq_handler)
static irqreturn_t irq_handler(int irq, void *dev)
{
    // 割込みを一時的に無効化します。(disable_irq_nosync)
    // ->割込み遅延処理が完了するまで無効にします。
    disable_irq_nosync(irq);

    // 加速度センサから値を取得する処理を、
    // ワークキューで遅延処理させます。(schedule_delayed_work)
    // ->acceleration_store_delay関数で指定されたミリ秒後に実行されるようにします。
    // schedule_delayed_workに指定する単位はjiffiesのため、
    // ミリ秒からjiffiesに変換します。(msecs_to_jiffies)
    schedule_delayed_work(&irq_process,
                         msecs_to_jiffies(event_interval_ms));

    return IRQ_HANDLED;
}

static struct platform_device *pdev;

static DEVICE_ATTR(enable, S_IWUSR, NULL, acceleration_store_enable);
static DEVICE_ATTR(delay, S_IWUSR, NULL, acceleration_store_delay);

// 属性配列
static struct attribute *acceleration_attrs[] = {

```

```

    &dev_attr_enable.attr,
    &dev_attr_delay.attr,
    NULL,
};

// 属性グループ構造体
static struct attribute_group acceleration_attr_group = {
    .attrs = acceleration_attrs,
};

// i2c_probe関数(acceleration_i2c_probe)
static int acceleration_i2c_probe(struct i2c_client *client, const struct i2c_device_id *id)
{
    int ret;

    // I2Cクライアントをグローバル変数(acceleration_i2c_client)に保持します。
    acceleration_i2c_client = client;

    // 入力デバイス構造体を割り当てます。(input_allocate_device)
    input = input_allocate_device();
    if (!input) {
        ret = -ENOMEM;
        goto err_ret;
    }

    // X軸、Y軸、Z軸の最大値、最小値を設定します。(input_set_abs_params)
    input_set_abs_params(input, ABS_X, -32, 31, 0, 0);
    input_set_abs_params(input, ABS_Y, -32, 31, 0, 0);
    input_set_abs_params(input, ABS_Z, -32, 31, 0, 0);

    // 入力デバイス構造体を設定します。
    input->name = "armadillo-x1-extension-acceleration";
    input->phys = "armadillo-x1/input2";
    input->id.bustype = BUS_I2C;
    input->dev.parent = &pdev->dev;

    // ABSイベントを登録します。(set_bit)
    // ->この設定によって、デバイスがどのような機能を持つのかを設定します。
    set_bit(EV_ABS, input->evbit);
    set_bit(ABS_X, input->absbit);
    set_bit(ABS_Y, input->absbit);
    set_bit(ABS_Z, input->absbit);

    // 加速度センサから割込みが発生するように要求します。(request_irq)
    // ->割込み番号は、client->irqを指定します。
    // 割込みハンドラは、irq_handlerを指定します。
    ret = request_irq(client->irq, irq_handler, 0, "armadillo-x1-extension-acceleration", input);
    if (ret < 0)
        goto err_free_mem;

    // 割込みを一時的に無効化します。(disable_irq)
    // ->割込み番号は、client->irqを指定します。
    disable_irq(client->irq);

    // 入力デバイスを登録します。(input_register_device)
    ret = input_register_device(input);
    if (ret < 0)
        goto err_free_irq;

    // 属性グループを作成します。(sysfs_create_group)
    ret = sysfs_create_group(&client->dev.kobj, &acceleration_attr_group);
    if (ret < 0)
        goto err_input_unregister_device;

    return 0;

err_input_unregister_device:
    input_unregister_device(input);
err_free_irq:

```

```

        free_irq(client->irq, input);
err_free_mem:
    input_free_device(input);
err_ret:
    return ret;
}

// i2c_remove関数(acceleration_i2c_remove)
static int acceleration_i2c_remove(struct i2c_client *client)
{
    // 割込み要求を解除します。(free_irq)
    free_irq(client->irq, input);

    // 加速度センサを無効にします。(acceleration_disable_setting)
    acceleration_disable_setting();

    // ワークキューの処理が完了するまで待機します。(flush_scheduled_work)
    flush_scheduled_work();

    // ワークキューの遅延処理をキャンセルします。(cancel_delayed_work_sync)
    cancel_delayed_work_sync(&irq_process);

    // 属性グループを削除します。(sysfs_remove_group)
    sysfs_remove_group(&client->dev.kobj, &acceleration_attr_group);

    // 入力デバイスの登録を解除します。(input_unregister_device)
    input_unregister_device(input);

    // 入力デバイス構造体のメモリ領域を開放します。(input_free_device)
    input_free_device(input);

    // グローバル変数(acceleration_i2c_client)を初期化します。
    acceleration_i2c_client = NULL;

    return 0;
}

// デバイスマッチング構造体配列
static const struct of_device_id acceleration_i2c_of_match[] = {
    { .compatible = "fsl,mma7660" },
    {},
};

// デバイスID構造体配列
static const struct i2c_device_id acceleration_i2c_id_table[] = {
    { "mma7660", 0 },
    {},
};

// I2Cドライバ構造体
static struct i2c_driver acceleration_i2c_driver = {
    .id_table     = acceleration_i2c_id_table,
    .probe        = acceleration_i2c_probe,
    .remove       = acceleration_i2c_remove,
    .driver       = {
        .name   = "acceleration",
        .owner  = THIS_MODULE,
        .of_match_table = of_match_ptr(acceleration_i2c_of_match),
    },
};

// probe関数(acceleration_probe)
static int acceleration_probe(struct platform_device *platform_device)
{
    pdev = platform_device;

    // I2Cドライバを登録します。(i2c_add_driver)
    return i2c_add_driver(&acceleration_i2c_driver);
}

```

```

415 // remove関数(acceleration_remove)
416 static int acceleration_remove(struct platform_device *pdev)
417 {
418     // I2Cドライバの登録を解除します。(i2c_del_driver)
419     i2c_del_driver(&acceleration_i2c_driver);
420     return 0;
421 }
423
424 // プラットフォームドライバ
425 static struct platform_driver acceleration_platform_driver = {
426     .probe      = acceleration_probe,
427     .remove     = acceleration_remove,
428     .driver     = {
429         .name   = "armadillo-x1-extension-acceleration",
430         .owner  = THIS_MODULE,
431     },
432 };
433
434 // 初期化関数(acceleration_init)
435 static int __init acceleration_init(void)
436 {
437     int ret;
438
439     // プラットフォームデバイスを登録します。(platform_device_register_simple)
440     // ->プラットフォーム依存のデバイス情報を登録します。
441     pdev = platform_device_register_simple("armadillo-x1-extension-acceleration", -1, NULL, 0);
442     if (IS_ERR(pdev)) {
443         ret = (int)pdev;
444         goto err_ret;
445     }
446
447     // プラットフォームドライバを登録します。(platform_driver_register)
448     // ->登録したプラットフォームデバイスのリソース情報を取得し、
449     // プラットフォームドライバとして、probe関数とremove関数を登録します。
450     ret = platform_driver_register(&acceleration_platform_driver);
451     if (ret < 0)
452         goto err_platform_device_unregister;
453
454     return 0;
455
456 err_platform_device_unregister:
457     platform_device_unregister(pdev);
458 err_ret:
459     return ret;
460 }
461
462 // 終了関数(acceleration_exit)
463 static void __exit acceleration_exit(void)
464 {
465     // プラットフォームドライバの登録を解除します。(platform_driver_unregister)
466     platform_driver_unregister(&acceleration_platform_driver);
467     // プラットフォームデバイスの登録を解除します。(platform_device_unregister)
468     platform_device_unregister(pdev);
469 }
470
471 // 初期化の際に、初期化関数が呼ばれるように登録します。
472 module_init(acceleration_init);
473 // 終了する際に、終了関数が呼ばれるように登録します。
474 module_exit(acceleration_exit);
475
476 // MODULE_LICENSEは"GPL"とします。
477 MODULE_LICENSE("GPL");

```

## Makefile

▼ 04.sensor/drivers/acceleration/Makefile

```

1 KERNELDIR = /home/atmark/linux-4.9-x1-at27_dbg
2 ARCH = arm
3 PREFIX = arm-linux-gnueabihf-
4 MOD_PATH = /work/linux/nfsroot
5
6 EXTRA_CFLAGS += -gdwarf-2 -O0
7
8 obj-m := acceleration.o
9
10 modules:
11     $(MAKE) -C $(KERNELDIR) M=`pwd` ARCH=$(ARCH) CROSS_COMPILE=$(PREFIX) modules
12
13 modules_install:
14     $(MAKE) -C $(KERNELDIR) M=`pwd` ARCH=$(ARCH) INSTALL_MOD_PATH=$(MOD_PATH) modules_install
15
16 myinstall:
17     cp -p *.ko /media/sf_ArmadilloX1/hwpwm/dbg/debug_share_hwpwm/R06_2024/04_practice
18     cp -p *.c /media/sf_ArmadilloX1/hwpwm/dbg/debug_share_hwpwm/R06_2024/04_practice
19
20 clean:
21     $(MAKE) -C $(KERNELDIR) M=`pwd` clean

```

## 1.4.2. 動作確認

### make clean

▼ \$ make clean

```

1 atmark@atde8:/media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor/drivers/acceler
2 make -C /home/atmark/linux-4.9-x1-at27_dbg M=`pwd` clean
3 make[1]: ディレクトリ '/home/atmark/linux-4.9-x1-at27_dbg' に入ります
4 make[1]: ディレクトリ '/home/atmark/linux-4.9-x1-at27_dbg' から出ます

```

### make modules

**⚠ 「make[2]: 警告: ファイル '/media/sf\_ArmadilloX1/hwpwm/work/R06\_2024/Application\_debug/text/practice-example/02.led/drivers/leds/leds.o' の修正時刻 20 は未来の時刻です」と表示された場合は chrony を ATDE8 と ArmadilloX1 にインストールすると解決する**

▼ \$ make modules

```

1 atmark@atde8:/media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor/drivers/acceler
2 make -C /home/atmark/linux-4.9-x1-at27_dbg M=`pwd` ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- modules
3 make[1]: ディレクトリ '/home/atmark/linux-4.9-x1-at27_dbg' に入ります
4 CC [M] /media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor/drivers/accelerati
5 Building modules, stage 2.
6 MODPOST 1 modules
7 make[2]: 警告: ファイル '/media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor/drive
8 CC /media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor/drivers/accelerati
9 LD [M] /media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor/drivers/accelerati
10 make[2]: 警告: 時刻のズれを検出. 不完全なビルド結果になるかもしれません.
11 make[1]: ディレクトリ '/home/atmark/linux-4.9-x1-at27_dbg' から出ます

```

### sudo make modules\_install

▼ \$ sudo make modules\_install

```
1 atmark@atde8:/media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor/drivers/acceler
2 [sudo] atmark のパスワード:
3 make -C /home/atmark/linux-4.9-x1-at27_dbg M=`pwd` ARCH=arm INSTALL_MOD_PATH=/work/linux/nfsroot modules_install
4 make[1]: ディレクトリ '/home/atmark/linux-4.9-x1-at27_dbg' に入ります
5     INSTALL /media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor/drivers/accelerati
6     DEPMOD 4.9.133-at27
7 depmod: WARNING: could not open modules.order at /work/linux/nfsroot/lib/modules/4.9.133-at27: No such file or directory
8 depmod: WARNING: could not open modules.builtin at /work/linux/nfsroot/lib/modules/4.9.133-at27: No such file or directory
9 make[1]: ディレクトリ '/home/atmark/linux-4.9-x1-at27_dbg' から出ます
```

## sudo make myinstall

▼ \$ sudo make myinstall

```
1 atmark@atde8:/media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor/drivers/acceler
2 cp -p *.ko /media/sf_ArmadilloX1/hwpwm/dbg/debug_share_hwpwm/R06_2024/04_practice
3 cp -p *.c /media/sf_ArmadilloX1/hwpwm/dbg/debug_share_hwpwm/R06_2024/04_practice
```

## cd

▼ root@armadillo:# cd /lib/modules/4.9.133-at27/extral

```
1 | root@armadillo:~# cd /lib/modules/4.9.133-at27/extral
```

## insmod

▼ #insmod leds.ko, #insmod acceleration.ko と #insmod buttons.ko

```
1 root@armadillo:/lib/modules/4.9.133-at27/extral# insmod leds.ko
2 root@armadillo:/lib/modules/4.9.133-at27/extral# insmod acceleration.ko
3 root@armadillo:/lib/modules/4.9.133-at27/extral# insmod buttons.ko
4 root@armadillo:/lib/modules/4.9.133-at27/extral# lsmod
5 Module           Size  Used by
6 buttons          3065  0
7 acceleration     5941  0
8 leds             2103  0
```

# tail -f /var/log/kern.log で /dev/input/event\* を確認

 acceleration.ko -> buttons.ko の順番で insmod すると、acceleration は event 3、buttons は event 4となる

▼ root@armadillo:~# tail -f /var/log/kern.log

```
1 root@armadillo:~# tail -f /var/log/kern.log
2 Feb 14 19:12:08 armadillo kernel: random: systemd: uninitialized urandom read (16 bytes read)
3 Feb 14 19:12:08 armadillo kernel: random: systemd: uninitialized urandom read (16 bytes read)
4 Feb 14 19:12:08 armadillo kernel: nf_conntrack: default automatic helper assignment has been turned off for security rea
5 Feb 14 19:12:08 armadillo kernel: random: crng init done
6 Feb 14 19:12:08 armadillo kernel: random: 6 urandom warning(s) missed due to ratelimiting
7 Jul  5 07:48:44 armadillo kernel: IPVS: Creating netns size=912 id=1
8 Jul  5 07:48:45 armadillo kernel: IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
9 Jul  5 07:48:46 armadillo kernel: IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
10 Jul  5 09:07:29 armadillo kernel: leds: loading out-of-tree module taints kernel.
11 Jul  5 09:07:35 armadillo kernel: input: armadillo-x1-extension-acceleration as /devices/platform/armadillo-x1-extension-
12 Jul  5 09:28:28 armadillo kernel: input: armadillo-x1-extension-btns as /devices/platform/armadillo-x1-extension-btns/in
```

## 1.4.3. デバイスファイル

"/dev/input/event\*"

\*には連番

**⚠ ボタンスイッチ、センサなど複数の入力デバイスがある場合、ソースファイル内ではデバイスファイル/dev/input/event\* の \* を決め打ちしているので注意すること**

## イベント

読み出したイベントデータは次のinput\_event構造体の形で表示

▼ input\_event構造体

```
1 #include <linux/input.h>
2
3 struct input_event {
4     struct timeval time;
5     __u16 type;
6     __u16 code;
7     __s32 value;
8 };
```

type	code	value
3(EV_ABS)	0(ABS_X)	X軸加速度(mG)
3(EV_ABS)	1(ABS_Y)	Y軸加速度(mG)
3(EV_ABS)	2(ABS_Z)	Z軸加速度(mG)

G は重力加速度の単位( $1.0G = 9.8m/s^2$ )

mG は G の 1000 分の 1

### EV\_ABSイベント発生間隔設定ファイル

"/sys/devices/soc0/soc/30800000.aips-bus/30a30000.i2c/i2c-1/1-004c/delay"

イベント発生間隔（初期値）：1000ms

### センサの有効化ファイル

"/sys/devices/soc0/soc/30800000.aips-bus/30a30000.i2c/i2c-1/1-004c/enable"

有効化: 0 以外の値を書き込む

無効化: 0

### その他

- 一連のイベントの終わりに EV\_SYN（同期）
- ボタンイベントと同様に EV\_ABS イベントの直後に EV\_SYN イベントが発生

## 1.5. 例題 asevent

センサイベントの内容を表示する

- X軸加速度が変化したとき --> "X:[X軸valueの値]"
- Y軸加速度が変化したとき --> "Y:[Y軸valueの値]"
- Z軸加速度が変化したとき --> "Z:[Z軸valueの値]"
- 同期イベント発生時 --> "EV\_SYN:-----"

センサを有効化、発生間隔は2000ms

### 1.5.1. ソース

#### asevent.c

▼ 04.sensor/asevent.c

```

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <linux/input.h>
5
6 // イベントデバイスファイル
7 #define AS_EV_FILE      "/dev/input/event3"
8 // 加速度センサ制御用ファイル
9 #define AS_DELAY_FILE   "/sys/devices/soc0/soc/30800000.aips-bus/30a30000.i2c/i2c-1/1-004c/delay"
10 #define AS_ENABLE_FILE  "/sys/devices/soc0/soc/30800000.aips-bus/30a30000.i2c/i2c-1/1-004c/enable"
11 // 通知時間(ms)
12 #define DELAY_TIME      2000
13 // センサの有効設定
14 #define ENABLE_MODE     1
15
16 int main(int argc, char *argv[])
17 {
18     int fd_as, fd_as_delay, fd_as_enable;
19     int ret;
20     int value;
21     char data[32];
22     int n;
23     struct input_event ev;
24
25     // 加速度センサの通知間隔設定属性ファイルをオープンします。
26     fd_as_delay = open(AS_DELAY_FILE, O_WRONLY);
27     // オープンに失敗した場合、エラー終了します。
28     if (fd_as_delay < 0){
29         perror("failed to open device");
30         return 1;
31     }
32
33     // 加速度センサ有効化属性ファイルをオープンします。
34     fd_as_enable = open(AS_ENABLE_FILE, O_WRONLY);
35     // オープンに失敗した場合、エラー終了します。
36     if (fd_as_enable < 0){
37         perror("failed to open device");
38         return 1;
39     }
40
41     // イベントデバイスファイルをオープンします。
42     fd_as = open(AS_EV_FILE, O_RDONLY);
43     // オープンに失敗した場合、エラー終了します。
44     if (fd_as < 0){
45         perror("failed to open device");
46         return 1;
47     }
48
49     // 通知間隔を設定します。
50     n = sprintf(data, "%d", DELAY_TIME);
51     ret = write(fd_as_delay, data, n);
52     // 書き込みが失敗した場合、エラー終了します。
53     if (ret < 0){
54         perror("failed to write");
55         return 1;
56     }
57
58     // 加速度センサを有効にします。
59     n = sprintf(data, "%d", ENABLE_MODE);
60     ret = write(fd_as_enable, data, n);
61     // 書き込みが失敗した場合、エラー終了します。
62     if (ret < 0){
63         perror("failed to write");
64         return 1;
65     }
66
67     // イベント内容を表示する処理。
68     for(;;){
69

```

```

64 // 入力イベント構造体を読み込みます。
65 ret = read(fd_as, &ev, sizeof(ev));
66 // 読み込みに失敗した場合、エラー終了します。
67 if (ret < 0){
68     perror("failed to read events");
69     return 1;
70 }
71 // イベントのタイプで表示内容を変化します。
72 switch (ev.type){
73 // タイプがEV_ABSの時の処理。
74 case EV_ABS:
75     // コードがABS_XならXの値を表示します。
76     if (ev.code == ABS_X){
77         printf("X : %d\n", ev.value);
78     } else if (ev.code == ABS_Y){
79         printf("Y : %d\n", ev.value);
80     } else if (ev.code == ABS_Z){
81         printf("Z : %d\n", ev.value);
82     } else {
83         fprintf(stderr, "invalid event code\n");
84         return 1;
85     }
86     break;
87
88 // タイプがEV_SYNの時の処理。
89 case EV_SYN:
90     printf("EV_SYN: -----\n");
91     break;
92 // タイプが上記以外なら、エラー終了します。
93 default:
94     fprintf(stderr, "unknown event\n");
95     return 2;
96 }
97
98 // 加速度センサ制御用ファイルをクローズします。
99 close(fd_as_delay);
100 close(fd_as_enable);
101 // イベントデバイスファイルをクローズします。
102 close(fd_as);
103
104 return 0;
105 }
106
107
108 // 加速度センサ制御用ファイルをクローズします。
109 close(fd_as_delay);
110 close(fd_as_enable);
111 // イベントデバイスファイルをクローズします。
112 close(fd_as);
113
114 return 0;
115 }

```

## Makefile

### ▼ 04.sensor/Makefile

```

1 CC = arm-linux-gnueabihf-gcc
2 #TARGET = asevent asled
3 TARGET = asevent
4 CFLAGS = -gdwarf-2 -O0
5
6 all: $(TARGET)
7
8 install :
9     cp -p $(TARGET) /work/linux/nfsroot/debug/04_practice
10    cp -p $(TARGET) /media/sf_ArmadilloX1/hwpwm/dbg/debug_share_hwpwm/R06_2024/04_practice
11    cp -p $(TARGET).c /media/sf_ArmadilloX1/hwpwm/dbg/debug_share_hwpwm/R06_2024/04_practice
12
13 clean:
14     rm -f $(TARGET)
15
16 .PHONY: clean

```

## 1.5.2. 動作確認

### make clean

▼ \$ make clean

```
1 | atmark@atde8:/media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor$ make clean
2 | rm -f asevent
```

### make

▼ \$ make

```
1 | atmark@atde8:/media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor$ make
2 | arm-linux-gnueabihf-gcc -gdwarf-2 -O0 asevent.c -o asevent
```

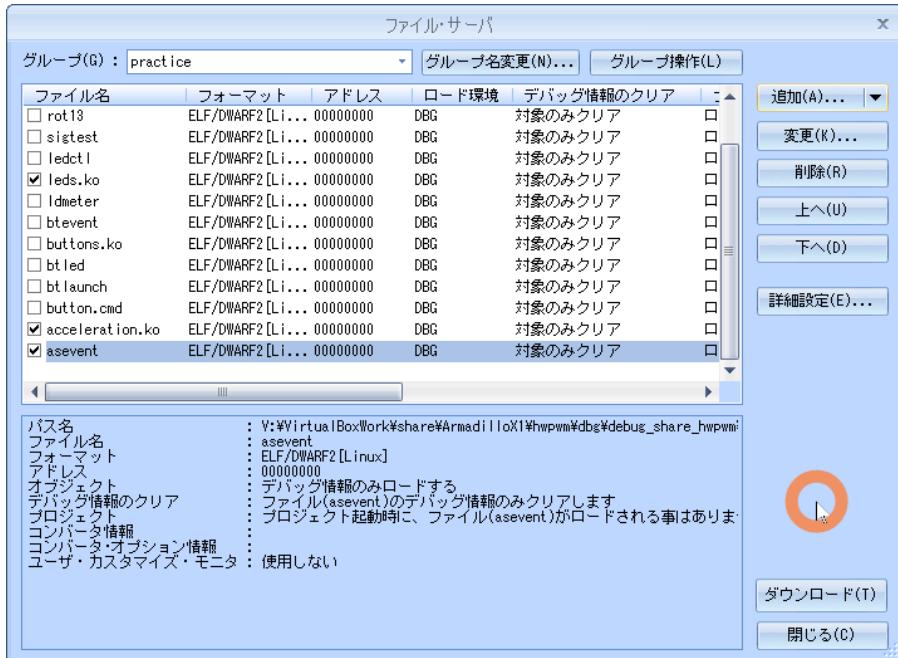
### sudo make install

▼ \$ sudo make install

```
1 | atmark@atde8:/media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor$ sudo make install
2 | cp -p asevent /work/linux/nfsroot/debug/04_practice
3 | cp -p asevent /media/sf_ArmadilloX1/hwpwm/dbg/debug_share_hwpwm/R06_2024/04_practice
4 | cp -p asevent.c /media/sf_ArmadilloX1/hwpwm/dbg/debug_share_hwpwm/R06_2024/04_practice
```

## CSIDEでロード

▼ メニュー「ファイル」 - 「ロード」



### insmod (既にinsmod済みなら割愛)

⚠️ leds.ko も insmod しておくこと

▼ # insmod leds.ko と # insmod acceleration.ko

```
1 | root@armadillo:/lib/modules/4.9.133-at27/extr# insmod leds.ko
2 | root@armadillo:/lib/modules/4.9.133-at27/extr# insmod acceleration.ko
3 | root@armadillo:/lib/modules/4.9.133-at27/extr# lsmod
4 | Module           Size  Used by
5 | acceleration      5941  0
6 | leds              2103  0
7 | root@armadillo:/lib/modules/4.9.133-at27/extr#
```

## 実行結果

---

▼ root@armadillo:/debug/04\_practice# ./asevent

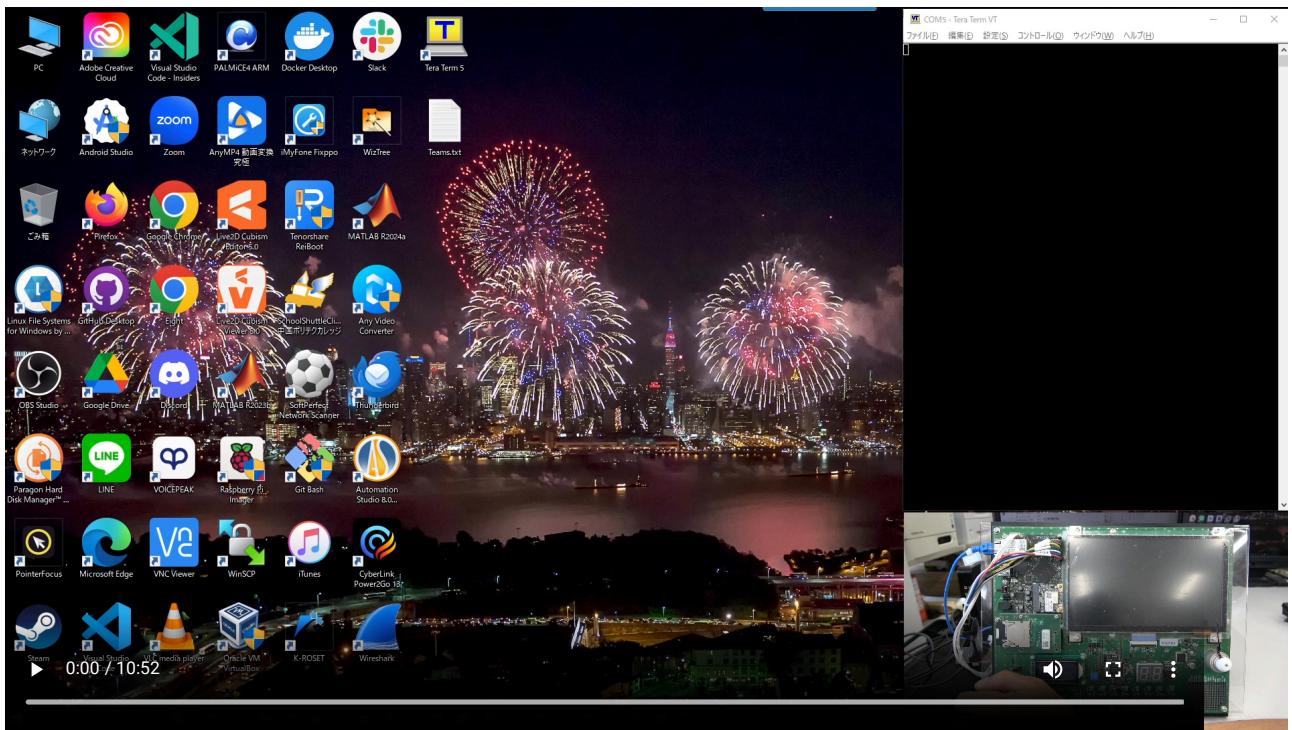
```
1 | root@armadillo:/debug/04_practice# ./asevent
2 | X : -940
3 | EV_SYN: -----
4 | X : -893
5 | Z : 376
6 | EV_SYN: -----
7 | X : -799
8 | Y : -470
9 | Z : 329
10 | EV_SYN: -----
11 | X : -611
12 | Y : -846
13 | Z : 235
14 | EV_SYN: -----
15 | X : -987
16 | Y : 94
17 | Z : 0
18 | EV_SYN: -----
19 | X : -705
20 | Y : 705
21 | Z : 188
22 | EV_SYN: -----
23 | X : -141
24 | Y : 423
25 | Z : 893
26 | EV_SYN: -----
27 | X : 94
28 | Y : 705
29 | Z : 658
30 | EV_SYN: -----
31 | X : -940
32 | Y : -188
33 | Z : 235
34 | EV_SYN: -----
35 | ^C
```

## 実行している様子

---

▼ asevent を実行している動画

<https://youtu.be/1O3wMkOxOCI>



## 1.6. 課題1 asled

センサイベントの値に応じてLEDを点灯する

加速度センサ Y軸	LED
-90 ~ -60	LED4,5,6,7,8 点灯
-60 ~ -30	LED4,5,6,7, 点灯
-30 ~ 0	LED4,5,6, 点灯
0	LED4,5 点灯
0 ~ 30	LED3,4,5 点灯
30 ~ 60	LED2,3,4,5 点灯
60 ~ 90	LED1,2,3,4,5 点灯

※角度はおおよそで良い

※センサイベントは100ms間隔で発生

### 1.6.1. ソース

#### asled.c

▼ 04.sensor/asled.c

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <linux/input.h>
5
6 // イベントデバイスファイル
7 #define AS_EV_FILE      "/dev/input/event3"
8 // 加速度センサ制御用ファイル
9 #define AS_DELAY_FILE   "/sys/devices/soc0/soc/30800000.aips-bus/30a30000.i2c/i2c-1/1-004c/delay"
10 #define AS_ENABLE_FILE  "/sys/devices/soc0/soc/30800000.aips-bus/30a30000.i2c/i2c-1/1-004c/enable"
11 // LED制御用ファイル
12 #define LED_FILE        "/sys/class/leds/led_ext/brightness"
13 // 通知時間(ms)
14 #define DELAY_TIME      100
15 // センサの有効設定
16 #define ENABLE_MODE     1
17 // LED書き込み用定義
18 #define LED1            0x01
19 #define LED2            0x02
20 #define LED3            0x04
21 #define LED_CENTER      0x18
22 #define LED6            0x20
23 #define LED7            0x40
24 #define LED8            0x80
25 // Y軸の閾値
26 #define THRESHOLD1      200
27 #define THRESHOLD2      400
28 #define THRESHOLD3      600
29
30 int main(void)
31 {
32
33
34
35
36
37
38
39     // 加速度センサの通知間隔設定属性ファイルをオープンします。
40
41
42
43
44
45
46
47     // 加速度センサ有効化属性ファイルをオープンします。
48
49
50
51
52
53
54
55     // イベントデバイスファイルをオープンします。
56
57
58
59
60
61
62     // LEDデバイスファイルをオープンします。
63
64
65
66
67
68
69
```

```
64
70
71
72
73
74
75
76
77
78
79    // 加速度センサを有効にします。
80
81
82
83
84
85
86
87
88    // LEDを点灯させる処理。
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
```

```
139
140
141
142
143
144
145
146
147     // 加速度センサ制御用ファイルをクローズします。
148
149
150     // LED制御用ファイルをクローズします。
151
152     // イベントデバイスファイルをクローズします。
153
154
155     return 0;
156 }
```

## Makefile

### ▼ 04.sensor/Makefile

```
1 CC = arm-linux-gnueabihf-gcc
2 #TARGET = asevent asled
3 TARGET = asled
4 CFLAGS = -gdwarf-2 -O0
5
6 all: $(TARGET)
7
8 install :
9     cp -p $(TARGET) /work/linux/nfsroot/debug/04_practice
10    cp -p $(TARGET) /media/sf_ArmadilloX1/hwpwm/dbg/debug_share_hwpwm/R06_2024/04_practice
11    cp -p $(TARGET).c /media/sf_ArmadilloX1/hwpwm/dbg/debug_share_hwpwm/R06_2024/04_practice
12
13 clean:
14     rm -f $(TARGET)
15
16 .PHONY: clean
```

## 1.6.2. 動作確認

### make clean

#### ▼ \$ make clean

```
1 | atmark@atde8:/media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor$ make clean
2 | rm -f asled
```

### make

#### ▼ \$ make

```
1 | atmark@atde8:/media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor$ make
2 | arm-linux-gnueabihf-gcc -gdwarf-2 -O0    asled.c   -o asled
```

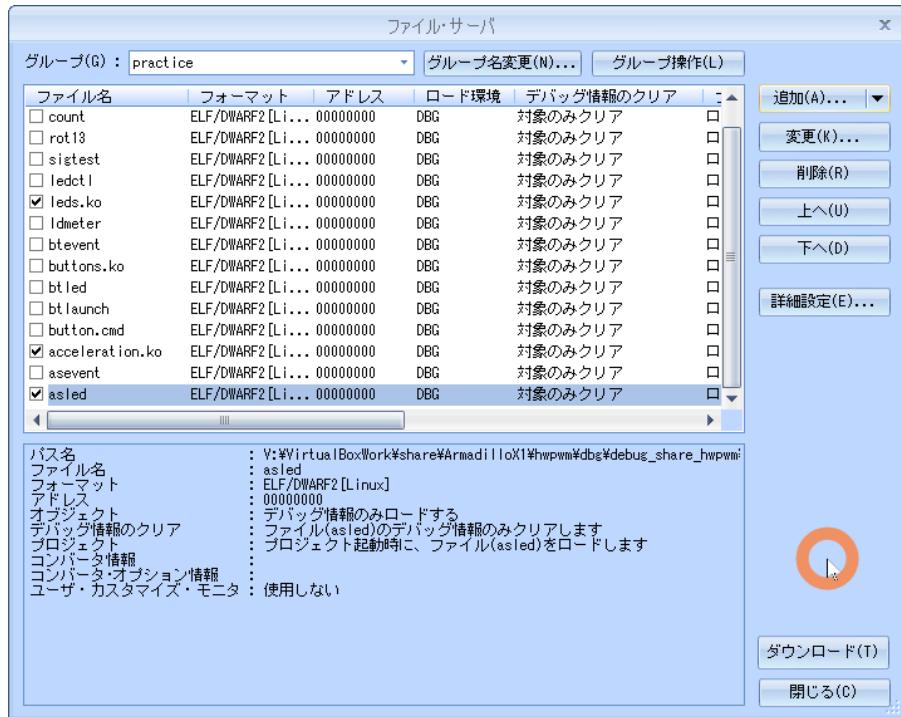
### sudo make install

#### ▼ \$ sudo make install

```
1 | atmark@atde8:/media/sf_ArmadilloX1/hwpwm/work/R06_2024/Application_debug/text/practice-example/04.sensor$ sudo make install
2 | [sudo] atmark のパスワード:
3 | cp -p asled /work/linux/nfsroot/debug/04_practice
4 | cp -p asled /media/sf_ArmadilloX1/hwpwm/dbg/debug_share_hwpwm/R06_2024/04_practice
5 | cp -p asled.c /media/sf_ArmadilloX1/hwpwm/dbg/debug_share_hwpwm/R06_2024/04_practice
```

## CSIDEでロード

### ▼ メニュー「ファイル」 - 「ロード」



### insmod (既にinsmod済みなら割愛)

#### ▼ # insmod leds.ko と # insmod acceleration.ko

```
1 | root@armadillo:/lib/modules/4.9.133-at27/extra# insmod leds.ko
2 | root@armadillo:/lib/modules/4.9.133-at27/extra# insmod acceleration.ko
3 | root@armadillo:/lib/modules/4.9.133-at27/extra# lsmod
4 | Module           Size  Used by
5 | acceleration      5941  0
6 | leds              2103  0
```

## 実行

#### ▼ root@armadillo:/debug/04\_practice# ./asled

```
1 | root@armadillo:/debug/04_practice# ./asled
```

## 実行している様子

#### ▼ asledによる加速度センサとLEDの運動動画

<https://youtu.be/91Cou1sUmjA>

