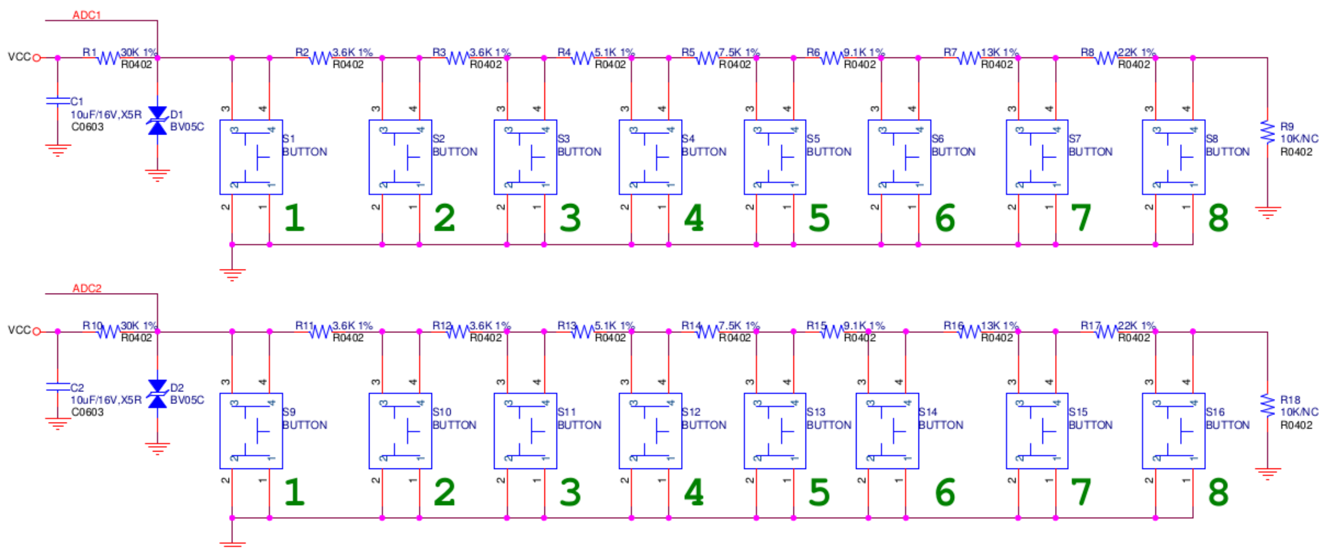
ADC_KEY驱动调试

driver

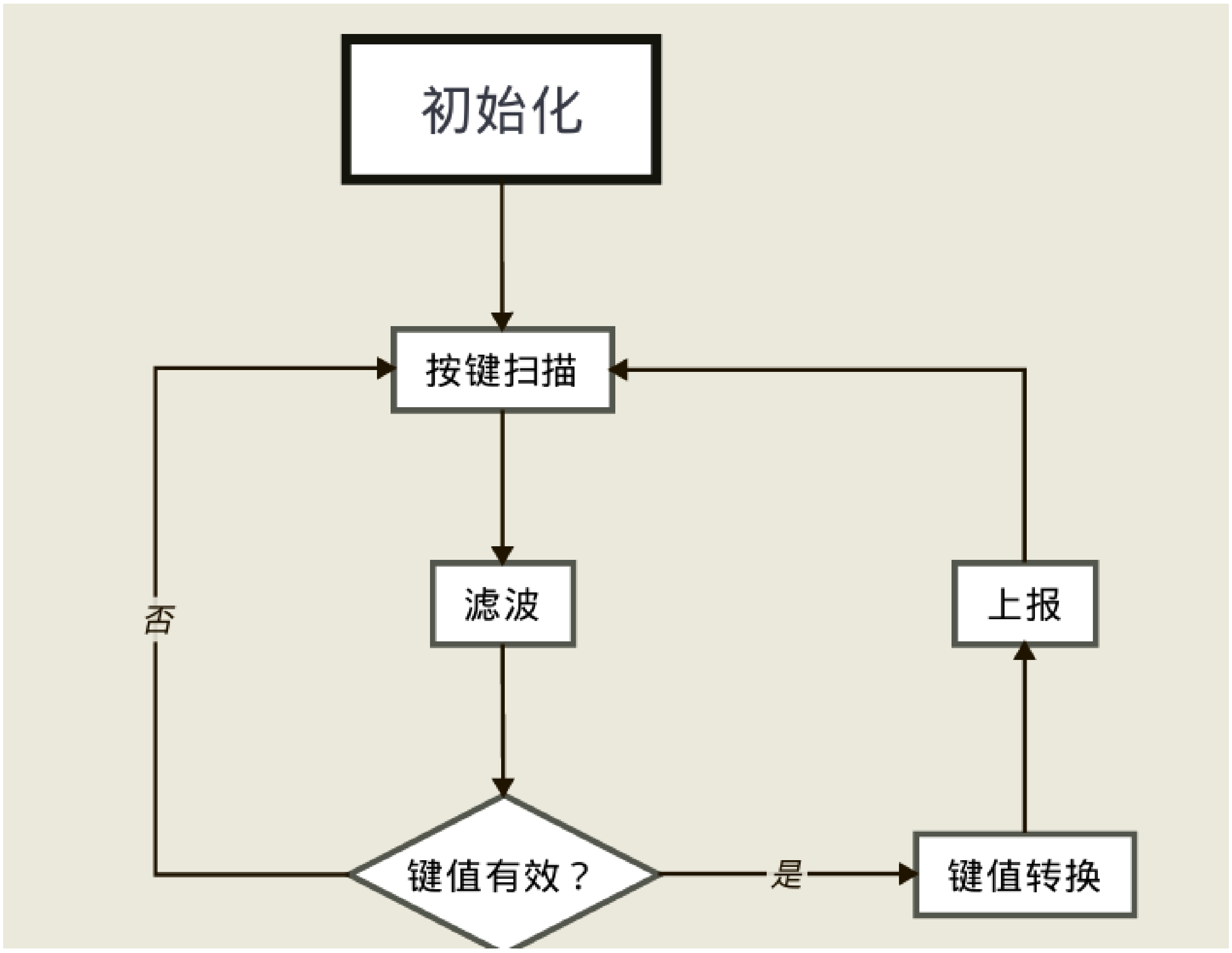
简介

adc按键与gpio按键相比，极大的节省了引脚的使用，在本例中，使用的是PMU（atc260x）的ADC(10bit/16channel)中的两个通道。

原理图



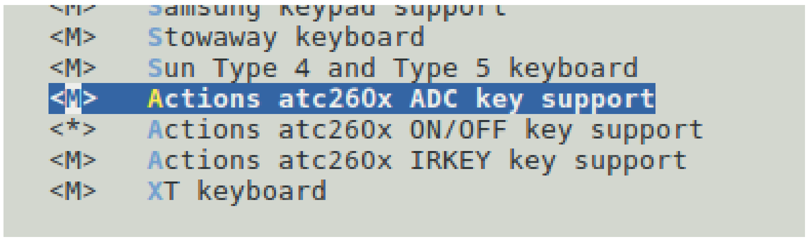
流程图



配置与修改

内核配置

首先，打开内核配置选项：



```
<M>   Samsung keypad support
<M>   Stowaway keyboard
<M>   Sun Type 4 and Type 5 keyboard
<M>   Actions atc260x ADC key support
<*>   Actions atc260x ON/OFF key support
<M>   Actions atc260x IRKEY key support
<M>   XT keyboard
```

dts

Key	Voltage
KEY1	0V
KEY2	0.3V
KEY3	0.6V
KEY4	0.9V
KEY5	1.2V
KEY6	1.5V
KEY7	1.8V
KEY8	2.1V
OFF	3.1V

由于ADC为10位，故相应的adc_val为：0, 99, 198, 297... 为消除误差，故在dts中配置left_adc_val和right_adc_val为：

```
left_adc_val =  < 0  80 180 280 380 480 580 680>;
right_adc_val = <20 120 220 320 420 520 620 720>;
```

keymapsize表示一路通道接多少个按键，相应的，与按键相关的属性如：*adc_val/adc_val/key_val* 的个数要与keymapsize的值相同。

```
keymapsize = <8>;            /* number of ADC key */
```

默认只支持1路通道，所以要添加另一路通道的按键名与键值：

```
key_val  = <1  2  3  4  5  6  7  8>;
key_val1  = <11  22  33  44  55  66  77  88>;
adc_channel_name = "AUX0";
adc_channel_name1 = "REMCON";
```

驱动

在这里添加宏`#define CHANNEL_NUM 2`表示驱动使用的通道数。
并在源码多处使用了`for (i = 0; i < CHANNEL_NUM; ++i)`循环。
把一路改为多路，以下变量有所改变：

```
unsigned int auxadc_channel[CHANNEL_NUM];
unsigned int *adc_buffer[CHANNEL_NUM];
unsigned int *key_values[CHANNEL_NUM];
unsigned int old_key_val[CHANNEL_NUM];
unsigned int key_val[CHANNEL_NUM];
```

atc260x_adckeypad_probe

该函数主要是获取dts配置中的信息，如keymapsize, key_val等。所使用函数的原型为：

```
static inline int of_property_read_u32(const struct device_node *np,
            const char *propname, u32 *out_value)
{
    return of_property_read_u32_array(np, propname, out_value, 1);
}
```

同时也申请并注册了input设备。

atc260x_adckeypad_config

在`atc260x_adckeypad_probe`函数中被调用，用于获取所使用的通道信息。

```
static int atc260x_adckeypad_config(struct atc260x_adckeypad_dev *atc260x_adckeypad)
{
    struct atc260x_dev *atc260x = atc260x_adckeypad->atc260x;
    const char *default_channel_name[CHANNEL_NUM] = {"REMCON", "AUX0"};
    const char *channel_name[CHANNEL_NUM];
    const char *of_prop_name[CHANNEL_NUM] = {"adc_channel_name", "adc_channel_name1"};
    int ret, i;

    /* no need to touch the hardware,
     * we use the service from the parent device (ie. the core). */

    for (i = 0; i < CHANNEL_NUM; ++i) {
        ret = of_property_read_string(
```

```c
            atc260x_adckeypad->dev->of_node, of_prop_name[i], &(channel_name[i]));
        if (ret) {
            dev_warn(atc260x_adckeypad->dev, "%s() can not get of_prop %s\n",
                __func__, of_prop_name[i]);
            /* use default value */
            channel_name[i] = default_channel_name[i];
        }
        dev_info(atc260x_adckeypad->dev, "select AUXADC channel %s", channel_name[i]);

        ret = atc260x_auxadc_find_chan(atc260x, channel_name[i]);
        if (ret < 0) {
            dev_err(atc260x_adckeypad->dev, "%s() unknown channel %s\n",
                __func__, channel_name[i]);
            return ret;
        }
        atc260x_adckeypad->auxadc_channel[i] = ret;
    }
    return 0;
}
```

atc260x_adckeypad_poll

轮询函数，在这里调用按键扫描，过滤，转换与上报等函数。

```c
static void atc260x_adckeypad_poll(struct input_polled_dev *dev)
{
    struct atc260x_adckeypad_dev *atc260x_adckeypad = dev->private;
    struct input_dev *input_dev = dev->input;
    static unsigned int i = 0;
    int ret;

    ret = atc260x_adckeypad_scan(atc260x_adckeypad->atc260x, dev, i);
    if (ret < 0)
        return;
    atc260x_adckeypad->adc_buffer[i][atc260x_adckeypad->filter_index] = ret;
    atc260x_adckeypad->filter_index =
        (atc260x_adckeypad->filter_index < atc260x_adckeypad->filter_dep) ?
            atc260x_adckeypad->filter_index + 1 : 0;
    if (atc260x_adckeypad->filter_index == 0)
        i = (i == CHANNEL_NUM -1) ? 0: i + 1;
    ret = atc260x_adckeypad_filter(dev, i);
    if (ret >= 0) {
        atc260x_adckeypad->key_val[i] =
            atc260x_adckeypad_convert(i, ret, atc260x_adckeypad);
        atc260x_adckeypad_report(input_dev, atc260x_adckeypad);
    }
}
```

atc260x_adckeypad_scan

按键扫描函数，此函数用于获取adc_val.

```c
static int atc260x_adckeypad_scan(struct atc260x_dev *atc260x,
    struct input_polled_dev *poll_dev, int index)
{
    struct atc260x_adckeypad_dev *atc260x_adckeypad = poll_dev->private;
    s32 tr_val;
    int ret;
    /* no need to touch the hardware,
     * we use the service from the parent device (ie. the core). */
    ret = atc260x_auxadc_get_translated(atc260x_adckeypad->atc260x,
        atc260x_adckeypad->auxadc_channel[index], &tr_val);
    if (ret) {
        dev_err(atc260x_adckeypad->dev,
            "%s() failed to get raw value of auxadc channel #%u\n",
            __func__, atc260x_adckeypad->auxadc_channel[index]);
        tr_val = 4095; /* use max value instead. */
    }

    /* tr_val is in the range [0, 4095] */
    if (tr_val < 0 || tr_val > 4095) {
        dev_err(atc260x_adckeypad->dev,
            "%s() auxadc channel #%u result out of range\n",
            __func__, atc260x_adckeypad->auxadc_channel[index]);
        tr_val = 4095; /* use max value instead. */
    }
    /* dev_info(atc260x_adckeypad->dev, "%s() adc_val=%u\n", __func__, tr_val); */
    return tr_val;
}
```

atc260x_adckeypad_filter

滤波，减小误差。

```c
static int atc260x_adckeypad_filter(struct input_polled_dev *dev, int index)
{
    struct atc260x_adckeypad_dev *atc260x_adckeypad = dev->private;
    uint tmp, sum_cnt, adc_val_sum;
    uint i, j;
    int diff;

    if (atc260x_adckeypad->adc_buffer[index] == NULL)
        return -EINVAL;
    if (atc260x_adckeypad->filter_dep == 0)
        return -EINVAL;
    sum_cnt = atc260x_adckeypad->filter_dep;

    adc_val_sum = 0;
    for (i = 0; i < sum_cnt; i++) {
        tmp = atc260x_adckeypad->adc_buffer[index][i];
        if (tmp == (typeof(tmp))-1)
            return -EINVAL;
        for (j = i + 1; j < sum_cnt; j++) {
            diff = tmp - atc260x_adckeypad->adc_buffer[index][j];
            diff = (diff >= 0) ? diff : -diff;
            if (diff >= atc260x_adckeypad->variance) {
                return -EINVAL;
            }
        }
        adc_val_sum += tmp;
    }
    return adc_val_sum / sum_cnt;
}
```

atc260x_adckeypad_convert

转换，根据从得到的adc_val确定相应的按键值。

```c
static inline unsigned int atc260x_adckeypad_convert(unsigned int index,
    unsigned int adc_val, struct atc260x_adckeypad_dev *atc260x_adckeypad)
{
    unsigned int i;
    unsigned int key_val = KEY_RESERVED;

    for (i = 0; i < atc260x_adckeypad->keymapsize; i++) {
        if ((adc_val >= *(atc260x_adckeypad->left_adc_val + i))
            && (adc_val <= *(atc260x_adckeypad->right_adc_val + i))) {
            key_val = *(atc260x_adckeypad->key_values[index] + i);
            break;
        }
    }
    return key_val;
}
```

atc260x_adckeypad_report

按键上报。

```
static void atc260x_adckeypad_report(struct input_dev *input_dev,
    struct atc260x_adckeypad_dev *atc260x_adckeypad)
{
    int i;
    unsigned int changed[CHANNEL_NUM];

    for (i = 0; i < CHANNEL_NUM; ++i) {
        changed[i] = atc260x_adckeypad->old_key_val[i] ^
atc260x_adckeypad->key_val[i];
        if (changed[i]) {
            if (atc260x_adckeypad->key_val[i] != KEY_RESERVED) {
                dev_info(atc260x_adckeypad->dev, "key_code=%d val=1\n",
                    atc260x_adckeypad->key_val[i]);
                input_report_key(input_dev, atc260x_adckeypad->key_val[
i], 1);
                input_sync(input_dev);
            }
            if (atc260x_adckeypad->old_key_val[i]!= KEY_RESERVED) {
                dev_info(atc260x_adckeypad->dev, "key_code=%d val=0\n",
                    atc260x_adckeypad->old_key_val[i]);
                input_report_key(input_dev, atc260x_adckeypad->old_key_
val[i], 0);
                input_sync(input_dev);
            }
            atc260x_adckeypad->old_key_val[i] = atc260x_adckeypad->key_
val[i];
        }
    }
}
```

源代码

dts

```
atc260x-adckeypad{
keymapsize = <8>;                    /* number of ADC key */
filter_dep = <3>;                    /* depth of the moving average filter (length of filter queue) */
variance = <10>;                     /* filter will not output until the difference between every two ADC samples in the filter queue goes below this value */
poll_interval =  <10>;               /* sample period, in ms */
left_adc_val =  < 0  80 180 280 380 480 580 680>;     /* for key code translator, each value defines a lower bound of ADC value of a key */
right_adc_val = <20 120 220 320 420 520 620 720>;  /* for key code translator, each value defines a upper bound of ADC value of a key */
key_val = <1 2 3 4 5 6 7 8>; /* for key code translator, each value defines the key_code of a key */
key_val1 = <11 22 33 44 55 66 77 88>; /* for key code translator, each value defines the key_code of a key */
adc_channel_name = "AUX0";           /* the ADC channel used for sampling, valid names are REMCON AUX0 AUX1 AUX2 (AUX3) */
adc_channel_name1 = "REMCON";           /* the ADC channel used for sampling, valid names are REMCON AUX0 AUX1 AUX2 (AUX3) */
compatible = "actions,atc2603c-adckeypad";
status = "okay";                 /* enable/disable ADC key function (okay or disabled) */
};
```

驱动源码

```
/*
 * Asoc adc keypad driver
 *
 * Copyright (C) 2011 Actions Semiconductor, Inc
 * Author:  chenbo <chenbo@actions-semi.com>
 *
 * This program is free software; you can redistribute it and/or modif
y
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */
#include <linux/init.h>
#include <linux/input.h>
#include <linux/input-polldev.h>
#include <linux/slab.h>
#include <linux/interrupt.h>
#include <linux/jiffies.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/platform_device.h>
#include <linux/of_device.h>
#include <asm/delay.h>
#include <asm/io.h>
#include <linux/fb.h>
/*#include <mach/gl5203_gpio.h> */
#include <linux/mfd/atc260x/atc260x.h>


#define ADCKEYPAD_DEBUG     0

#define KEY_VAL_INIT             KEY_UP
#define KEY_VAL_HOLD             SW_RADIO
#define CHANNEL_NUM         2

static const unsigned int left_adc[9] = {
    0x00, 0x32, 0x97, 0xfb, 0x15f, 0x1c3, 0x24e, 0x2b3, 0x317
};
static const unsigned int right_adc[9] = {
    0x00, 0x96, 0xfa, 0x15e, 0x1c2, 0x226, 0x2b2, 0x316, 0x400
};
static const unsigned int key_val[9] = {
    KEY_HOME, KEY_MENU, KEY_VOLUMEUP, KEY_VOLUMEDOWN,
    KEY_RESERVED, KEY_RESERVED, KEY_RESERVED,
    KEY_RESERVED, KEY_UP
};

struct adc_key {
unsigned int min_adc_val;   /*! min adc sample value */
unsigned int max_adc_val;   /*! max adc sample value */
unsigned int keyval;        /*! report key value */
};
```

```c

struct atc260x_adckeypad_dev {
    struct device *dev;
    struct atc260x_dev *atc260x;
    struct input_polled_dev *poll_dev;

    unsigned int auxadc_channel[CHANNEL_NUM];
    unsigned int *adc_buffer[CHANNEL_NUM];

    unsigned int *left_adc_val;
    unsigned int *right_adc_val;
    unsigned int *key_values[CHANNEL_NUM];

    unsigned int filter_dep;
    unsigned int variance;

    unsigned int keymapsize;
    unsigned int old_key_val[CHANNEL_NUM];
    unsigned int key_val[CHANNEL_NUM];
    unsigned int filter_index;
};

static inline unsigned int atc260x_adckeypad_convert(unsigned int index,
    unsigned int adc_val, struct atc260x_adckeypad_dev *atc260x_adckeypad)
{
    unsigned int i;
    unsigned int key_val = KEY_RESERVED;

    for (i = 0; i < atc260x_adckeypad->keymapsize; i++) {
        if ((adc_val >= *(atc260x_adckeypad->left_adc_val + i))
            && (adc_val <= *(atc260x_adckeypad->right_adc_val + i))) {
            key_val = *(atc260x_adckeypad->key_values[index] + i);
            break;
        }
    }
    return key_val;
}

static void atc260x_adckeypad_report(struct input_dev *input_dev,
    struct atc260x_adckeypad_dev *atc260x_adckeypad)
{
    int i;
    unsigned int changed[CHANNEL_NUM];

    for (i = 0; i < CHANNEL_NUM; ++i) {
        changed[i] = atc260x_adckeypad->old_key_val[i] ^ atc260x_adckeypad->key_val[i];
        if (changed[i]) {
            if (atc260x_adckeypad->key_val[i] != KEY_RESERVED) {
                dev_info(atc260x_adckeypad->dev, "key_code=%d val=1\n",
                    atc260x_adckeypad->key_val[i]);
                input_report_key(input_dev, atc260x_adckeypad->key_val[i], 1);
                input_sync(input_dev);
            }
            if (atc260x_adckeypad->old_key_val[i]!= KEY_RESERVED) {
                dev_info(atc260x_adckeypad->dev, "key_code=%d val=0\n",
                    atc260x_adckeypad->old_key_val[i]);
                input_report_key(input_dev, atc260x_adckeypad->old_key_val[i], 0);
                input_sync(input_dev);
            }
            atc260x_adckeypad->old_key_val[i] = atc260x_adckeypad->key_val[i];
        }
    }
}

static int atc260x_adckeypad_scan(struct atc260x_dev *atc260x,
    struct input_polled_dev *poll_dev, int index)
{
    struct atc260x_adckeypad_dev *atc260x_adckeypad = poll_dev->private;
    s32 tr_val;
    int ret;
    /* no need to touch the hardware,
     * we use the service from the parent device (ie. the core). */
    ret = atc260x_auxadc_get_translated(atc260x_adckeypad->atc260x,
        atc260x_adckeypad->auxadc_channel[index], &tr_val);
    if (ret) {
        dev_err(atc260x_adckeypad->dev,
            "%s() failed to get raw value of auxadc channel #%u\n",
            __func__, atc260x_adckeypad->auxadc_channel[index]);
        tr_val = 4095; /* use max value instead. */
    }

    /* tr_val is in the range [0, 4095] */
    if (tr_val < 0 || tr_val > 4095) {
        dev_err(atc260x_adckeypad->dev,
            "%s() auxadc channel #%u result out of range\n",
            __func__, atc260x_adckeypad->auxadc_channel[index]);
        tr_val = 4095; /* use max value instead. */
    }
    /* dev_info(atc260x_adckeypad->dev, "%s() adc_val=%u\n", __func__, tr_val); */
    return tr_val;
}

static int atc260x_adckeypad_filter(struct input_polled_dev *dev, int index)
{
    struct atc260x_adckeypad_dev *atc260x_adckeypad = dev->private;
    uint tmp, sum_cnt, adc_val_sum;
    uint i, j;
    int diff;

    if (atc260x_adckeypad->adc_buffer[index] == NULL)
        return -EINVAL;
    if (atc260x_adckeypad->filter_dep == 0)
        return -EINVAL;
    sum_cnt = atc260x_adckeypad->filter_dep;

    adc_val_sum = 0;
    for (i = 0; i < sum_cnt; i++) {
        tmp = atc260x_adckeypad->adc_buffer[index][i];
        if (tmp == (typeof(tmp))-1)
            return -EINVAL;
        for (j = i + 1; j < sum_cnt; j++) {
            diff = tmp - atc260x_adckeypad->adc_buffer[index][j];
            diff = (diff >= 0) ? diff : -diff;
//          printk("buffer[%d][%d] = %d, sbuffer[%d][%d] = %d \n ", index,i,tmp, index,j,atc260x_adckeypad->adc_buffer[index][j] );
            if (diff >= atc260x_adckeypad->variance) {
                return -EINVAL;
            }
        }
        adc_val_sum += tmp;
    }
    return adc_val_sum / sum_cnt;
}

static void atc260x_adckeypad_poll(struct input_polled_dev *dev)
{
    struct atc260x_adckeypad_dev *atc260x_adckeypad = dev->private;
    struct input_dev *input_dev = dev->input;
    static unsigned int i = 0;
    int ret;

//  for (i = 0; i < CHANNEL_NUM; ++i) {
        ret = atc260x_adckeypad_scan(atc260x_adckeypad->atc260x, dev, i);
        if (ret < 0)
            return;
        atc260x_adckeypad->adc_buffer[i][atc260x_adckeypad->filter_index] = ret;
        atc260x_adckeypad->filter_index =
            (atc260x_adckeypad->filter_index < atc260x_adckeypad->filter_dep) ?
                atc260x_adckeypad->filter_index + 1 : 0;
        if (atc260x_adckeypad->filter_index == 0)
            i = (i == CHANNEL_NUM -1) ? 0: i + 1;
        ret = atc260x_adckeypad_filter(dev, i);
        if (ret >= 0) {
            atc260x_adckeypad->key_val[i] =
                atc260x_adckeypad_convert(i, ret, atc260x_adckeypad);
            atc260x_adckeypad_report(input_dev, atc260x_adckeypad);
        }
//  }
}

static int atc260x_adckeypad_config(struct atc260x_adckeypad_dev *atc260x_adckeypad)
{
    struct atc260x_dev *atc260x = atc260x_adckeypad->atc260x;
    const char *default_channel_name[CHANNEL_NUM] = {"REMCON", "AUX0"};
    const char *channel_name[CHANNEL_NUM];
    const char *of_prop_name[CHANNEL_NUM] = {"adc_channel_name", "adc_channel_name1"};
    int ret, i;

    /* no need to touch the hardware,
     * we use the service from the parent device (ie. the core). */

    for (i = 0; i < CHANNEL_NUM; ++i) {
        ret = of_property_read_string(
            atc260x_adckeypad->dev->of_node, of_prop_name[i], &(channel_name[i]));
        if (ret) {
            dev_warn(atc260x_adckeypad->dev, "%s() can not get of_prop %s\n",
                __func__, of_prop_name[i]);
            /* use default value */
            channel_name[i] = default_channel_name[i];
        }
        dev_info(atc260x_adckeypad->dev, "select AUXADC channel %s", channel_name[i]);

        ret = atc260x_auxadc_find_chan(atc260x, channel_name[i]);
        if (ret < 0) {
            dev_err(atc260x_adckeypad->dev, "%s() unknown channel %s\n",
                __func__, channel_name[i]);
            return ret;
        }
        atc260x_adckeypad->auxadc_channel[i] = ret;
    }
    return 0;
}

static int atc260x_adckeypad_probe(struct platform_device *pdev)
{
    struct atc260x_dev *atc260x;
    struct atc260x_adckeypad_dev *atc260x_adckeypad;
    struct device_node *np;
    struct input_polled_dev *poll_dev;
    struct input_dev *input_dev;
    const char *dts_status_cfg_str;
    const char * of_key_val[] = {"key_val", "key_val1"};
    int ret = 0;
    int i;

    dev_info(&pdev->dev, "Probing...\n");

    np = pdev->dev.of_node;
    ret = of_property_read_string(np, "status", &dts_status_cfg_str);
    if (ret == 0 && strcmp(dts_status_cfg_str, "okay") != 0) {
        dev_info(&pdev->dev, "disabled by DTS\n");
        return -ENODEV;
    }

    atc260x = atc260x_get_parent_dev(&pdev->dev);

    atc260x_adckeypad = devm_kzalloc(&pdev->dev,
            sizeof(struct atc260x_adckeypad_dev), GFP_KERNEL);
    if (!atc260x_adckeypad) {
        dev_err(&pdev->dev, "%s() no mem\n", __func__);
        return -ENOMEM;
    }
    atc260x_adckeypad->dev = &pdev->dev;
    atc260x_adckeypad->atc260x = atc260x;
    platform_set_drvdata(pdev, atc260x_adckeypad);

    ret = atc260x_adckeypad_config(atc260x_adckeypad);
    if (ret)
        goto of_property_read_err;

    atc260x_adckeypad->filter_index = 0;
    atc260x_adckeypad->old_key_val[0] = KEY_VAL_INIT;
    atc260x_adckeypad->old_key_val[1] = KEY_VAL_INIT;
    /*
     * get configure info from xml
     */
#if (ADCKEYPAD_DEBUG == 1)
    atc260x_adckeypad->keymapsize = 9;
    /*get left adc val*/
    atc260x_adckeypad->left_adc_val = left_adc;
    /*get right adc val*/
    atc260x_adckeypad->right_adc_val = right_adc;
    /*get key values*/
    atc260x_adckeypad->key_values = key_val;

    atc260x_adckeypad->filter_dep = 5;
    atc260x_adckeypad->variance = 50;

    atc260x_adckeypad->adc_buffer = devm_kzalloc(
            atc260x_adckeypad->dev,
            sizeof(unsigned int) * atc260x_adckeypad->filter_dep,
            GFP_KERNEL);
    if (!atc260x_adckeypad->adc_buffer)
        goto free_buffer;
    memset(atc260x_adckeypad->adc_buffer, 0xff,
        sizeof(unsigned int) * atc260x_adckeypad->filter_dep);

#else
    /*get keymapsize*/
    ret = of_property_read_u32(np, "keymapsize", &(atc260x_adckeypad->k
eymapsize));
    if ((ret) || (!atc260x_adckeypad->keymapsize)) {
```

```
        dev_err(&pdev->dev, "Get keymapsize failed ret = %d \r\n", ret);
        goto of_property_read_err;
    }
    dev_info(&pdev->dev, "keymapsize = %d\n", atc260x_adckeypad->keymapsize);

    /*get key filter depth*/
    ret = of_property_read_u32(np, "filter_dep", &(atc260x_adckeypad->filter_dep));
    if ((ret) || (!atc260x_adckeypad->filter_dep)) {
        dev_err(&pdev->dev, "Get filter_dep failed ret = %d\r\n", ret);
        goto of_property_read_err;
    }
    dev_info(&pdev->dev, "filter_dep = %d\n", atc260x_adckeypad->filter_dep);

    /*get variance val */
    ret = of_property_read_u32(np, "variance", &(atc260x_adckeypad->variance));
    if ((ret) || (!atc260x_adckeypad->variance)) {
        dev_err(&pdev->dev, "Get variance failed ret = %d\r\n", ret);
        goto of_property_read_err;
    }
    dev_info(&pdev->dev, "variance = %d\n", atc260x_adckeypad->variance);

    /*get left adc val*/
    atc260x_adckeypad->left_adc_val = devm_kzalloc(&pdev->dev,
        sizeof(unsigned int) * (atc260x_adckeypad->keymapsize), GFP_KERNEL);
    if (!atc260x_adckeypad->left_adc_val)
        goto free;

    ret = of_property_read_u32_array(np, "left_adc_val",
        (u32 *)atc260x_adckeypad->left_adc_val,
        atc260x_adckeypad->keymapsize);
    if (ret) {
        dev_err(&pdev->dev, "Get left_adc_val failed ret = %d\r\n", ret);
        goto free_left;
    }

    /*get right adc val*/
```

```c
    atc260x_adckeypad->right_adc_val = devm_kzalloc(&pdev->dev,
        sizeof(unsigned int) * (atc260x_adckeypad->keymapsize), GFP_KERNEL);
    if (!atc260x_adckeypad->right_adc_val)
        goto free;

    ret = of_property_read_u32_array(np, "right_adc_val",
        (u32 *)atc260x_adckeypad->right_adc_val,
        atc260x_adckeypad->keymapsize);
    if (ret) {
        dev_err(&pdev->dev, "Get right_adc_val failed ret = %d\r\n", ret);
        goto free_right;
    }

    for (i = 0; i < CHANNEL_NUM; ++i) {
        /*get key val*/
        atc260x_adckeypad->key_values[i] = devm_kzalloc(&pdev->dev,
            sizeof(unsigned int) * (atc260x_adckeypad->keymapsize), GFP_KERNEL);
        if (!atc260x_adckeypad->key_values[i])
            goto free;

        ret = of_property_read_u32_array(np, of_key_val[i],
            (u32 *)atc260x_adckeypad->key_values[i],
            atc260x_adckeypad->keymapsize);
        if (ret) {
            dev_err(&pdev->dev, "Get key_values failed ret = %d\r\n", ret);
            goto free_key_values;
        }
        /*Malloc adc_buffer*/
        atc260x_adckeypad->adc_buffer[i] = devm_kzalloc(&pdev->dev,
            sizeof(unsigned int) * atc260x_adckeypad->filter_dep, GFP_KERNEL);
        if (!atc260x_adckeypad->adc_buffer[i])
            goto free_buffer;
        memset(atc260x_adckeypad->adc_buffer[i], 0xff,
            sizeof(unsigned int) * atc260x_adckeypad->filter_dep);
    }
#endif

    /*
     * poll dev related
     */
    poll_dev = input_allocate_polled_device();
    if (!poll_dev) {
        ret = -ENOMEM;
        goto free_buffer;
    }
    atc260x_adckeypad->poll_dev = poll_dev;

    poll_dev->private = atc260x_adckeypad;
    poll_dev->poll = atc260x_adckeypad_poll;

#if (ADCKEYPAD_DEBUG == 1)
    poll_dev->poll_interval = 5;/* msec */
#else
    /*get poll period*/
    ret = of_property_read_u32(np, "poll_interval", &(poll_dev->poll_interval));
    if ((ret) || (!poll_dev->poll_interval)) {
        dev_err(&pdev->dev, "Get poll_interval failed \r\n");
        goto free_buffer;
    }
    dev_info(&pdev->dev, "poll_interval = %ums\n", poll_dev->poll_interval);
#endif

    input_dev = poll_dev->input;
    input_dev->evbit[0] = BIT(EV_KEY) | BIT(EV_REP) | BIT(EV_SW);
    input_dev->name = pdev->name;
    input_dev->phys = "atc260x_adckeypad/input3";
    input_dev->keycode = atc260x_adckeypad->key_values;
    input_dev->keycodesize = atc260x_adckeypad->keymapsize;
    input_dev->keycodemax = atc260x_adckeypad->keymapsize;
    input_dev->dev.parent = &pdev->dev;
    input_dev->id.bustype = BUS_HOST;

    for (i = 0; i < atc260x_adckeypad->keymapsize; i++) {
        __set_bit(*(atc260x_adckeypad->key_values[0] + i), input_dev->keybit);
        __set_bit(*(atc260x_adckeypad->key_values[1] + i), input_dev->keybit);
    }
    __clear_bit(KEY_RESERVED, input_dev->keybit);
    __set_bit(KEY_POWER, input_dev->keybit);
    __set_bit(KEY_POWER2, input_dev->keybit);
    __set_bit(KEY_VAL_HOLD, input_dev->swbit);

    input_set_capability(input_dev, EV_MSC, MSC_SCAN);
    ret = input_register_polled_device(poll_dev);
    if (ret) {
        dev_err(&pdev->dev, "%s() failed to register_polled_device, ret
=%d\n",
            __func__, ret);
        goto free_polled;
    }

    input_dev->timer.data = (long) input_dev;
    return 0;

free_polled:
    platform_set_drvdata(pdev, NULL);
    input_free_polled_device(poll_dev);

free_buffer:
free_key_values:
free_right:
free_left:
free:
of_property_read_err:

    return ret;
}

static int atc260x_adckeypad_remove(struct platform_device *pdev)
{
    struct atc260x_adckeypad_dev *atc260x_adckeypad =
        platform_get_drvdata(pdev);

    platform_set_drvdata(pdev, NULL);
    input_unregister_polled_device(atc260x_adckeypad->poll_dev);
    input_free_polled_device(atc260x_adckeypad->poll_dev);

    return 0;
}

#ifdef CONFIG_PM
static int atc260x_adckeypad_suspend(struct platform_device *pdev,
    pm_message_t state)
{
    struct atc260x_adckeypad_dev *atc260x_adckeypad =
platform_get_drvdata(pdev);

```

```c
    cancel_delayed_work_sync(&atc260x_adckeypad->poll_dev->work);
    return 0;
}
static int atc260x_adckeypad_resume(struct platform_device *pdev)
{
    struct atc260x_adckeypad_dev *atc260x_adckeypad = platform_get_drvdata(pdev);

    schedule_delayed_work(&atc260x_adckeypad->poll_dev->work,
        msecs_to_jiffies(atc260x_adckeypad->poll_dev->poll_interval));
    return 0;
}
#else
    # define atc260x_adckeypad_suspend NULL
    # define atc260x_adckeypad_resume  NULL
#endif

static const struct of_device_id atc260x_adckey_of_match[] = {
    {.compatible = "actions,atc2603a-adckeypad",},
    {.compatible = "actions,atc2603c-adckeypad",},
    {.compatible = "actions,atc2609a-adckeypad",},
    {}
};

static struct platform_driver atc260x_adckeypad_driver = {
    .driver = {
        .name = "atc260x-adckeypad",
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(atc260x_adckey_of_match),
    },
    .probe = atc260x_adckeypad_probe,
    .remove = atc260x_adckeypad_remove,
    .suspend = atc260x_adckeypad_suspend,
    .resume = atc260x_adckeypad_resume,
};

module_platform_driver(atc260x_adckeypad_driver)

/*static int __init atc260x_adckeypad_init(void) */
/*{ */
/*  return platform_driver_register(&atc260x_adckeypad_driver); */
/*} */
/*subsys_init(atc260x_adckeypad_init); */
/*//late_initcall(atc260x_adckeypad_init); */
/*static void __exit atc260x_adckeypad_exit(void) */
```

```c
/*{ */
/*  platform_driver_unregister(&atc260x_adckeypad_driver); */
/*} */
/*module_exit(atc260x_adckeypad_exit); */


MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Asoc adckey  drvier");
MODULE_AUTHOR("sall.xie/Actions Semi, Inc");
```

测试源码

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <sys/ioctl.h>
#include <sys/fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <linux/input.h>

int main(void)
{
    struct input_event ev_key;
    int fd;
    fd = open("/dev/input/event2", O_RDWR);

    if (fd < 0) {
        perror("open device buttons");
        exit(1);
    }
    while(1) {
        read(fd, &ev_key, sizeof(struct input_event));
        if (EV_KEY == ev_key.type)
            printf("type:%d,code:%d,value:%d\n", ev_key.type,ev_key.code,ev_key.value);
    }
    close(fd);
    return 0;
}
```