

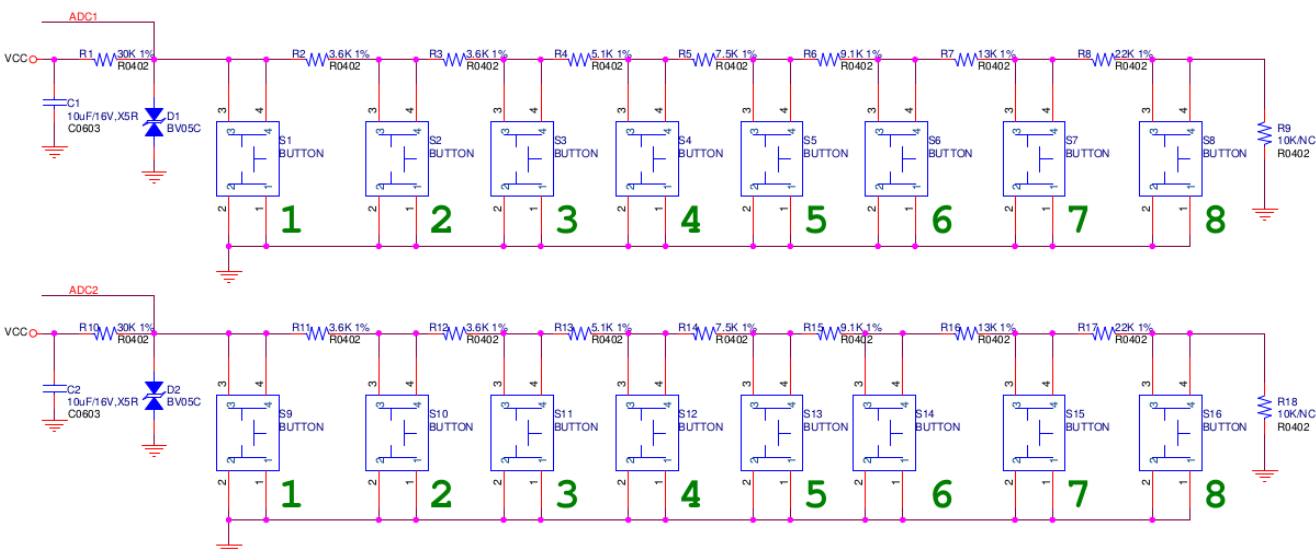
ADC_KEY驱动调试

driver

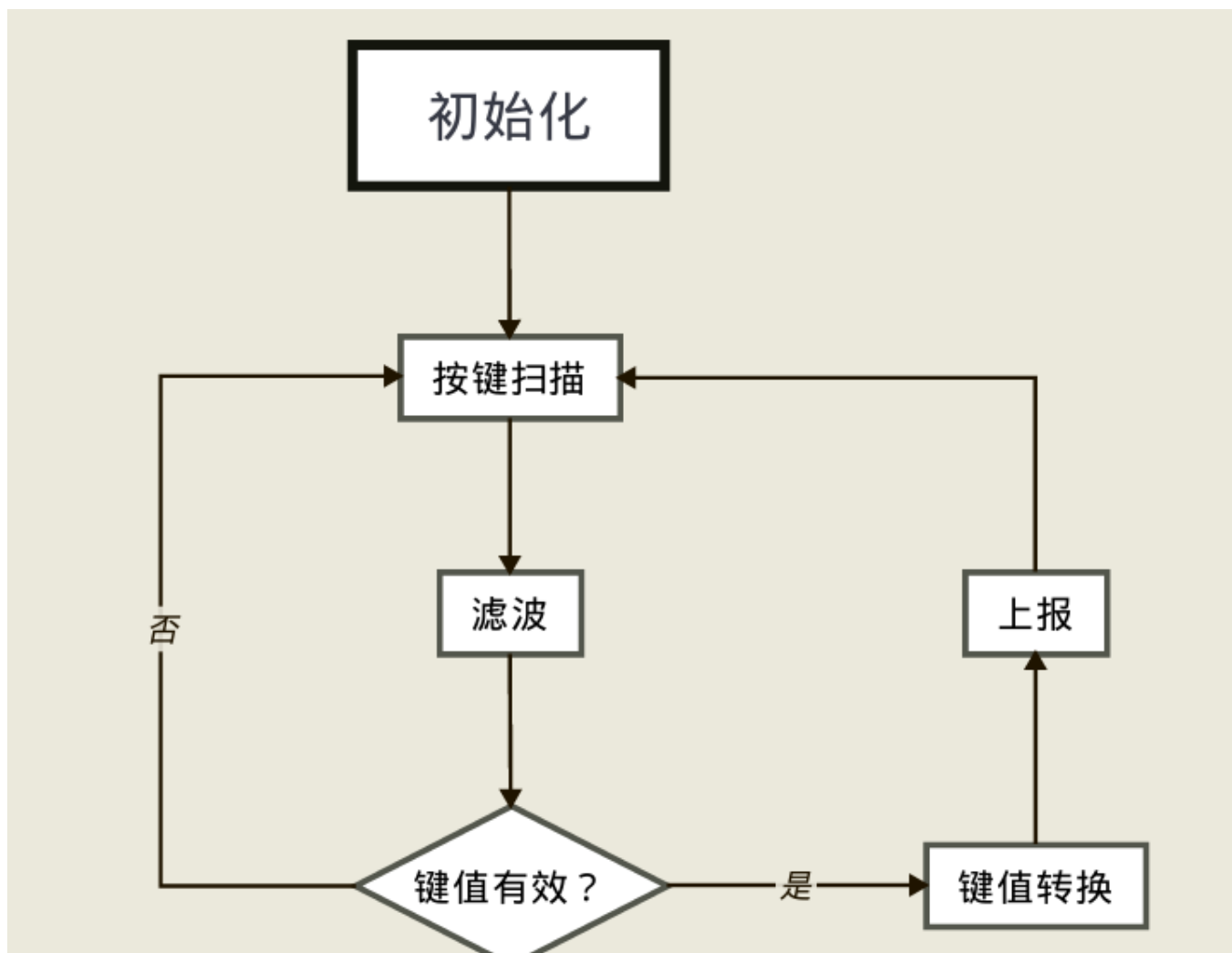
简介

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：

```
1. left_adc_val = <0 80 180 280 380 480 580 680>;
2. 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：

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

```
1. static inline int of_property_read_u32(const struct device_node *np,
2.                                       const char *propname, u32 *out_value)
3. {
4.     return of_property_read_u32_array(np, propname, out_value, 1);
5. }
```

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。

```
1. static int atc260x_adckeypad_config(struct atc260x_adckeypad_dev
2. *atc260x_adckeypad)
3. {
4.     struct atc260x_dev *atc260x = atc260x_adckeypad->atc260x;
5.     const char *default_channel_name[CHANNEL_NUM] = {"REMCON", "AUX0"};
6.     const char *channel_name[CHANNEL_NUM];
7.     const char *of_prop_name[CHANNEL_NUM] = {"adc_channel_name", "adc_c
8. hannel_name1"};
9.     int ret, i;
10.
11.     /* no need to touch the hardware,
12.      * we use the service from the parent device (ie. the core). */
13.
14.     for (i = 0; i < CHANNEL_NUM; ++i) {
15.         ret = of_property_read_string(
```

```

14.         atc260x_adckeypad->dev->of_node, of_prop_name[i], &(channel
        _name[i]));
15.         if (ret) {
16.             dev_warn(atc260x_adckeypad->dev, "%s() can not get of_prop
        %s\n",
17.                     __func__, of_prop_name[i]);
18.             /* use default value */
19.             channel_name[i] = default_channel_name[i];
20.         }
21.         dev_info(atc260x_adckeypad->dev, "select AUXADC channel %s", ch
        annel_name[i]);
22.
23.         ret = atc260x_auxadc_find_chan(atc260x, channel_name[i]);
24.         if (ret < 0) {
25.             dev_err(atc260x_adckeypad->dev, "%s() unknown channel %s\n"
        ,
26.                     __func__, channel_name[i]);
27.             return ret;
28.         }
29.         atc260x_adckeypad->auxadc_channel[i] = ret;
30.     }
31.     return 0;
32. }

```

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。

```

1.  static void atc260x_adckeypad_poll(struct input_polled_dev *dev)
2.  {
3.      struct atc260x_adckeypad_dev *atc260x_adckeypad = dev->private;
4.      struct input_dev *input_dev = dev->input;
5.      static unsigned int i = 0;
6.      int ret;
7.
8.      ret = atc260x_adckeypad_scan(atc260x_adckeypad->atc260x, dev, i);
9.      if (ret < 0)
10.         return;
11.     atc260x_adckeypad->adc_buffer[i][atc260x_adckeypad->filter_index] =
        ret;
12.     atc260x_adckeypad->filter_index =
13.         (atc260x_adckeypad->filter_index < atc260x_adckeypad->filter_de
        p) ?
14.         atc260x_adckeypad->filter_index + 1 : 0;

```

```

15.     if (atc260x_adckeypad->filter_index == 0)
16.         i = (i == CHANNEL_NUM -1) ? 0: i + 1;
17.     ret = atc260x_adckeypad_filter(dev, i);
18.     if (ret >= 0) {
19.         atc260x_adckeypad->key_val[i] =
20.             atc260x_adckeypad_convert(i, ret, atc260x_adckeypad);
21.         atc260x_adckeypad_report(input_dev, atc260x_adckeypad);
22.     }
23. }

```

atc260x_adckeypad_scan

按键扫描函数，此函数用于获取adc_val.

```

1.  static int atc260x_adckeypad_scan(struct atc260x_dev *atc260x,
2.      struct input_polled_dev *poll_dev, int index)
3.  {
4.      struct atc260x_adckeypad_dev *atc260x_adckeypad = poll_dev->private;
5.      s32 tr_val;
6.      int ret;
7.      /* no need to touch the hardware,
8.       * we use the service from the parent device (ie. the core). */
9.      ret = atc260x_auxadc_get_translated(atc260x_adckeypad->atc260x,
10.          atc260x_adckeypad->auxadc_channel[index], &tr_val);
11.      if (ret) {
12.          dev_err(atc260x_adckeypad->dev,
13.              "%s() failed to get raw value of auxadc channel #%u\n",
14.              __func__, atc260x_adckeypad->auxadc_channel[index]);
15.          tr_val = 4095; /* use max value instead. */
16.      }
17.
18.      /* tr_val is in the range [0, 4095] */
19.      if (tr_val < 0 || tr_val > 4095) {
20.          dev_err(atc260x_adckeypad->dev,
21.              "%s() auxadc channel #%u result out of range\n",
22.              __func__, atc260x_adckeypad->auxadc_channel[index]);
23.          tr_val = 4095; /* use max value instead. */
24.      }
25.      /* dev_info(atc260x_adckeypad->dev, "%s() adc_val=%u\n", __func__,
26.          tr_val); */
27.      return tr_val;
28.  }

```

atc260x_adckeypad_filter

滤波，减小误差。

```
1.  static int atc260x_adckeypad_filter(struct input_polled_dev *dev, int
2.  index)
3.  {
4.      struct atc260x_adckeypad_dev *atc260x_adckeypad = dev->private;
5.      uint tmp, sum_cnt, adc_val_sum;
6.      uint i, j;
7.      int diff;
8.
9.      if (atc260x_adckeypad->adc_buffer[index] == NULL)
10.         return -EINVAL;
11.      if (atc260x_adckeypad->filter_dep == 0)
12.         return -EINVAL;
13.      sum_cnt = atc260x_adckeypad->filter_dep;
14.
15.      adc_val_sum = 0;
16.      for (i = 0; i < sum_cnt; i++) {
17.          tmp = atc260x_adckeypad->adc_buffer[index][i];
18.          if (tmp == (typeof(tmp))-1)
19.              return -EINVAL;
20.          for (j = i + 1; j < sum_cnt; j++) {
21.              diff = tmp - atc260x_adckeypad->adc_buffer[index][j];
22.              diff = (diff >= 0) ? diff : -diff;
23.              if (diff >= atc260x_adckeypad->variance) {
24.                  return -EINVAL;
25.              }
26.          }
27.          adc_val_sum += tmp;
28.      }
29.      return adc_val_sum / sum_cnt;
30.  }
```

atc260x_adckeypad_convert

转换，根据从得到的adc_val确定相应的按键值。

```
1.  static inline unsigned int atc260x_adckeypad_convert(unsigned int index
2.  ,
3.  unsigned int adc_val, struct atc260x_adckeypad_dev *atc260x_adckeyp
4.  ad)
5.  {
```

```

4.     unsigned int i;
5.     unsigned int key_val = KEY_RESERVED;
6.
7.     for (i = 0; i < atc260x_adckeypad->keymapsize; i++) {
8.         if ((adc_val >= *(atc260x_adckeypad->left_adc_val + i))
9.             && (adc_val <= *(atc260x_adckeypad->right_adc_val + i))) {
10.            key_val = *(atc260x_adckeypad->key_values[index] + i);
11.            break;
12.        }
13.    }
14.    return key_val;
15. }

```

atc260x_adckeypad_report

按键上报。

```

1.  static void atc260x_adckeypad_report(struct input_dev *input_dev,
2.      struct atc260x_adckeypad_dev *atc260x_adckeypad)
3.  {
4.      int i;
5.      unsigned int changed[CHANNEL_NUM];
6.
7.      for (i = 0; i < CHANNEL_NUM; ++i) {
8.          changed[i] = atc260x_adckeypad->old_key_val[i] ^
atc260x_adckeypad->key_val[i];
9.          if (changed[i]) {
10.             if (atc260x_adckeypad->key_val[i] != KEY_RESERVED) {
11.                 dev_info(atc260x_adckeypad->dev, "key_code=%d val=1\n",
12.                     atc260x_adckeypad->key_val[i]);
13.                 input_report_key(input_dev, atc260x_adckeypad->key_val[
i], 1);
14.                 input_sync(input_dev);
15.             }
16.             if (atc260x_adckeypad->old_key_val[i] != KEY_RESERVED) {
17.                 dev_info(atc260x_adckeypad->dev, "key_code=%d val=0\n",
18.                     atc260x_adckeypad->old_key_val[i]);
19.                 input_report_key(input_dev, atc260x_adckeypad->old_key_
val[i], 0);
20.                 input_sync(input_dev);
21.             }
22.             atc260x_adckeypad->old_key_val[i] = atc260x_adckeypad->key_
val[i];
23.         }

```



```
24.     }
25. }
```

源代码

dts

```
1. atc260x-adckeypad{
2.     keymapsize = <8>; /* number of ADC key */
3.     filter_dep = <3>; /* depth of the moving average filter (length of filter queue) */
4.     variance = <10>; /* filter will not output until the difference between every two ADC samples in the filter queue goes below this value */
5.     poll_interval = <10>; /* sample period, in ms */
6.     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 */
7.     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 */
8.     key_val = <1 2 3 4 5 6 7 8>; /* for key code translator, each value defines the key_code of a key */
9.     key_val1 = <11 22 33 44 55 66 77 88>; /* for key code translator, each value defines the key_code of a key */
10.    adc_channel_name = "AUX0"; /* the ADC channel used for sampling, valid names are REMCON AUX0 AUX1 AUX2 (AUX3) */
11.    adc_channel_name1 = "REMC0N"; /* the ADC channel used for sampling, valid names are REMCON AUX0 AUX1 AUX2 (AUX3) */
12.    compatible = "actions,atc2603c-adckeypad";
13.    status = "okay"; /* enable/disable ADC key function (okay or disabled) */
14.};
```

驱动源码

```
1. /*
2.  * Asoc adc keypad driver
3.  *
4.  * Copyright (C) 2011 Actions Semiconductor, Inc
5.  * Author: chenbo <chenbo@actions-semi.com>
6.  */
```

```

7.  * This program is free software; you can redistribute it and/or modif
y
8.  * it under the terms of the GNU General Public License version 2 as
9.  * published by the Free Software Foundation.
10. */
11. #include <linux/init.h>
12. #include <linux/input.h>
13. #include <linux/input-polldev.h>
14. #include <linux/slab.h>
15. #include <linux/interrupt.h>
16. #include <linux/jiffies.h>
17. #include <linux/kernel.h>
18. #include <linux/module.h>
19. #include <linux/platform_device.h>
20. #include <linux/of_device.h>
21. #include <asm/delay.h>
22. #include <asm/io.h>
23. #include <linux/fb.h>
24. /*#include <mach/g15203_gpio.h> */
25. #include <linux/mfd/atc260x/atc260x.h>
26.
27.
28. #define ADCKEYPAD_DEBUG      0
29.
30. #define KEY_VAL_INIT          KEY_UP
31. #define KEY_VAL_HOLD          SW_RADIO
32. #define CHANNEL_NUM           2
33.
34. static const unsigned int left_adc[9] = {
35.     0x00, 0x32, 0x97, 0xfb, 0x15f, 0x1c3, 0x24e, 0x2b3, 0x317
36. };
37. static const unsigned int right_adc[9] = {
38.     0x00, 0x96, 0xfa, 0x15e, 0x1c2, 0x226, 0x2b2, 0x316, 0x400
39. };
40. static const unsigned int key_val[9] = {
41.     KEY_HOME, KEY_MENU, KEY_VOLUMEUP, KEY_VOLUMEDOWN,
42.     KEY_RESERVED, KEY_RESERVED, KEY_RESERVED,
43.     KEY_RESERVED, KEY_UP
44. };
45.
46. struct adc_key {
47.     unsigned int min_adc_val;    /*! min adc sample value */
48.     unsigned int max_adc_val;    /*! max adc sample value */
49.     unsigned int keyval;         /*! report key value */
50. };

```

```

51.
52. struct atc260x_adckeypad_dev {
53.     struct device *dev;
54.     struct atc260x_dev *atc260x;
55.     struct input_polled_dev *poll_dev;
56.
57.     unsigned int auxadc_channel[CHANNEL_NUM];
58.     unsigned int *adc_buffer[CHANNEL_NUM];
59.
60.     unsigned int *left_adc_val;
61.     unsigned int *right_adc_val;
62.     unsigned int *key_values[CHANNEL_NUM];
63.
64.     unsigned int filter_dep;
65.     unsigned int variance;
66.
67.     unsigned int keymapsize;
68.     unsigned int old_key_val[CHANNEL_NUM];
69.     unsigned int key_val[CHANNEL_NUM];
70.     unsigned int filter_index;
71. };
72.
73. static inline unsigned int atc260x_adckeypad_convert(unsigned int index
74. ,
75.     unsigned int adc_val, struct atc260x_adckeypad_dev *atc260x_adckeyp
76. ad)
77. {
78.     unsigned int i;
79.     unsigned int key_val = KEY_RESERVED;
80.
81.     for (i = 0; i < atc260x_adckeypad->keymapsize; i++) {
82.         if ((adc_val >= *(atc260x_adckeypad->left_adc_val + i))
83.             && (adc_val <= *(atc260x_adckeypad->right_adc_val + i))) {
84.             key_val = *(atc260x_adckeypad->key_values[index] + i);
85.             break;
86.         }
87.     }
88.     return key_val;
89. }
90.
91. static void atc260x_adckeypad_report(struct input_dev *input_dev,
92.     struct atc260x_adckeypad_dev *atc260x_adckeypad)
93. {
94.     int i;
95.     unsigned int changed[CHANNEL_NUM];

```

```

94.
95.     for (i = 0; i < CHANNEL_NUM; ++i) {
96.         changed[i] = atc260x_adckeypad->old_key_val[i] ^
atc260x_adckeypad->key_val[i];
97.         if (changed[i]) {
98.             if (atc260x_adckeypad->key_val[i] != KEY_RESERVED) {
99.                 dev_info(atc260x_adckeypad->dev, "key_code=%d val=1\n",
100.                    atc260x_adckeypad->key_val[i]);
101.                 input_report_key(input_dev, atc260x_adckeypad->key_val[
i], 1);
102.                 input_sync(input_dev);
103.             }
104.             if (atc260x_adckeypad->old_key_val[i] != KEY_RESERVED) {
105.                 dev_info(atc260x_adckeypad->dev, "key_code=%d val=0\n",
106.                    atc260x_adckeypad->old_key_val[i]);
107.                 input_report_key(input_dev, atc260x_adckeypad->old_key_
val[i], 0);
108.                 input_sync(input_dev);
109.             }
110.             atc260x_adckeypad->old_key_val[i] = atc260x_adckeypad->key_
val[i];
111.         }
112.     }
113. }
114.
115. static int atc260x_adckeypad_scan(struct atc260x_dev *atc260x,
116.     struct input_polled_dev *poll_dev, int index)
117. {
118.     struct atc260x_adckeypad_dev *atc260x_adckeypad = poll_dev->
private;
119.     s32 tr_val;
120.     int ret;
121.     /* no need to touch the hardware,
122.      * we use the service from the parent device (ie. the core). */
123.     ret = atc260x_auxadc_get_translated(atc260x_adckeypad->atc260x,
124.        atc260x_adckeypad->auxadc_channel[index], &tr_val);
125.     if (ret) {
126.         dev_err(atc260x_adckeypad->dev,
127.             "%s() failed to get raw value of auxadc channel #%u\n",
128.             __func__, atc260x_adckeypad->auxadc_channel[index]);
129.         tr_val = 4095; /* use max value instead. */
130.     }
131.
132.     /* tr_val is in the range [0, 4095] */
133.     if (tr_val < 0 || tr_val > 4095) {

```

```

134.         dev_err(atc260x_adckeypad->dev,
135.                 "%s() auxadc channel #%u result out of range\n",
136.                 __func__, atc260x_adckeypad->auxadc_channel[index]);
137.         tr_val = 4095; /* use max value instead. */
138.     }
139.     /* dev_info(atc260x_adckeypad->dev, "%s() adc_val=%u\n", __func__,
tr_val); */
140.     return tr_val;
141. }
142.
143. static int atc260x_adckeypad_filter(struct input_polled_dev *dev, int
index)
144. {
145.     struct atc260x_adckeypad_dev *atc260x_adckeypad = dev->private;
146.     uint tmp, sum_cnt, adc_val_sum;
147.     uint i, j;
148.     int diff;
149.
150.     if (atc260x_adckeypad->adc_buffer[index] == NULL)
151.         return -EINVAL;
152.     if (atc260x_adckeypad->filter_dep == 0)
153.         return -EINVAL;
154.     sum_cnt = atc260x_adckeypad->filter_dep;
155.
156.     adc_val_sum = 0;
157.     for (i = 0; i < sum_cnt; i++) {
158.         tmp = atc260x_adckeypad->adc_buffer[index][i];
159.         if (tmp == (typeof(tmp))-1)
160.             return -EINVAL;
161.         for (j = i + 1; j < sum_cnt; j++) {
162.             diff = tmp - atc260x_adckeypad->adc_buffer[index][j];
163.             diff = (diff >= 0) ? diff : -diff;
164.             // printk("buffer[%d][%d] = %d, sbuffer[%d][%d] = %d \n ",
index,i,tmp, index,j,atc260x_adckeypad->adc_buffer[index][j] );
165.             if (diff >= atc260x_adckeypad->variance) {
166.                 return -EINVAL;
167.             }
168.         }
169.         adc_val_sum += tmp;
170.     }
171.     return adc_val_sum / sum_cnt;
172. }
173.
174. static void atc260x_adckeypad_poll(struct input_polled_dev *dev)
175. {

```

```

176.     struct atc260x_adckeypad_dev *atc260x_adckeypad = dev->private;
177.     struct input_dev *input_dev = dev->input;
178.     static unsigned int i = 0;
179.     int ret;
180.
181.     // for (i = 0; i < CHANNEL_NUM; ++i) {
182.         ret = atc260x_adckeypad_scan(atc260x_adckeypad->atc260x, dev, i
183.     );
184.         if (ret < 0)
185.             return;
186.         atc260x_adckeypad->adc_buffer[i][atc260x_adckeypad->filter_index] = ret;
187.         atc260x_adckeypad->filter_index =
188.             (atc260x_adckeypad->filter_index < atc260x_adckeypad->filter_index_max) ?
189.                 atc260x_adckeypad->filter_index + 1 : 0;
190.         if (atc260x_adckeypad->filter_index == 0)
191.             i = (i == CHANNEL_NUM - 1) ? 0 : i + 1;
192.         ret = atc260x_adckeypad_filter(dev, i);
193.         if (ret >= 0) {
194.             atc260x_adckeypad->key_val[i] =
195.                 atc260x_adckeypad_convert(i, ret, atc260x_adckeypad);
196.             atc260x_adckeypad_report(input_dev, atc260x_adckeypad);
197.         }
198.     // }
199. }
200.
201. static int atc260x_adckeypad_config(struct atc260x_adckeypad_dev
202. *atc260x_adckeypad)
203. {
204.     struct atc260x_dev *atc260x = atc260x_adckeypad->atc260x;
205.     const char *default_channel_name[CHANNEL_NUM] = {"REMCON", "AUX0"};
206.     const char *channel_name[CHANNEL_NUM];
207.     const char *of_prop_name[CHANNEL_NUM] = {"adc_channel_name", "adc_channel_name1"};
208.     int ret, i;
209.
210.     /* no need to touch the hardware,
211.      * we use the service from the parent device (ie. the core). */
212.
213.     for (i = 0; i < CHANNEL_NUM; ++i) {
214.         ret = of_property_read_string(
215.             atc260x_adckeypad->dev->of_node, of_prop_name[i], &(channel_name[i]));
216.         if (ret) {

```

```

215.         dev_warn(atc260x_adckeypad->dev, "%s() can not get of_prop
%s\n",
216.             __func__, of_prop_name[i]);
217.         /* use default value */
218.         channel_name[i] = default_channel_name[i];
219.     }
220.     dev_info(atc260x_adckeypad->dev, "select AUXADC channel %s", ch
annel_name[i]);
221.
222.     ret = atc260x_auxadc_find_chan(atc260x, channel_name[i]);
223.     if (ret < 0) {
224.         dev_err(atc260x_adckeypad->dev, "%s() unknown channel %s\n"
,
225.             __func__, channel_name[i]);
226.         return ret;
227.     }
228.     atc260x_adckeypad->auxadc_channel[i] = ret;
229. }
230. return 0;
231. }
232.
233. static int atc260x_adckeypad_probe(struct platform_device *pdev)
234. {
235.     struct atc260x_dev *atc260x;
236.     struct atc260x_adckeypad_dev *atc260x_adckeypad;
237.     struct device_node *np;
238.     struct input_polled_dev *poll_dev;
239.     struct input_dev *input_dev;
240.     const char *dts_status_cfg_str;
241.     const char * of_key_val[] = {"key_val", "key_val1"};
242.     int ret = 0;
243.     int i;
244.
245.     dev_info(&pdev->dev, "Probing...\n");
246.
247.     np = pdev->dev.of_node;
248.     ret = of_property_read_string(np, "status", &dts_status_cfg_str);
249.     if (ret == 0 && strcmp(dts_status_cfg_str, "okay") != 0) {
250.         dev_info(&pdev->dev, "disabled by DTS\n");
251.         return -ENODEV;
252.     }
253.
254.     atc260x = atc260x_get_parent_dev(&pdev->dev);
255.
256.     atc260x_adckeypad = devm_kzalloc(&pdev->dev,

```

```

257.         sizeof(struct atc260x_adckeypad_dev), GFP_KERNEL);
258.     if (!atc260x_adckeypad) {
259.         dev_err(&pdev->dev, "%s() no mem\n", __func__);
260.         return -ENOMEM;
261.     }
262.     atc260x_adckeypad->dev = &pdev->dev;
263.     atc260x_adckeypad->atc260x = atc260x;
264.     platform_set_drvdata(pdev, atc260x_adckeypad);
265.
266.     ret = atc260x_adckeypad_config(atc260x_adckeypad);
267.     if (ret)
268.         goto of_property_read_err;
269.
270.     atc260x_adckeypad->filter_index = 0;
271.     atc260x_adckeypad->old_key_val[0] = KEY_VAL_INIT;
272.     atc260x_adckeypad->old_key_val[1] = KEY_VAL_INIT;
273.     /*
274.      * get configure info from xml
275.      */
276.     #if (ADCKEYPAD_DEBUG == 1)
277.         atc260x_adckeypad->keymapsize = 9;
278.         /*get left adc val*/
279.         atc260x_adckeypad->left_adc_val = left_adc;
280.         /*get right adc val*/
281.         atc260x_adckeypad->right_adc_val = right_adc;
282.         /*get key values*/
283.         atc260x_adckeypad->key_values = key_val;
284.
285.         atc260x_adckeypad->filter_dep = 5;
286.         atc260x_adckeypad->variance = 50;
287.
288.         atc260x_adckeypad->adc_buffer = devm_kzalloc(
289.             atc260x_adckeypad->dev,
290.             sizeof(unsigned int) * atc260x_adckeypad->filter_dep,
291.             GFP_KERNEL);
292.         if (!atc260x_adckeypad->adc_buffer)
293.             goto free_buffer;
294.         memset(atc260x_adckeypad->adc_buffer, 0xff,
295.             sizeof(unsigned int) * atc260x_adckeypad->filter_dep);
296.
297.     #else
298.         /*get keymapsize*/
299.         ret = of_property_read_u32(np, "keymapsize", &(atc260x_adckeypad->k
eymapsize));
300.         if ((ret) || (!atc260x_adckeypad->keymapsize)) {

```



```

301.         dev_err(&pdev->dev, "Get keymapsize failed ret = %d \r\n", ret)
302.     ;
303.         goto of_property_read_err;
304.     }
305.     dev_info(&pdev->dev, "keymapsize = %d\n", atc260x_adckeypad->keymap
306. size);
307.
308.     /*get key filter depth*/
309.     ret = of_property_read_u32(np, "filter_dep", &(atc260x_adckeypad->f
310. ilter_dep));
311.     if ((ret) || (!atc260x_adckeypad->filter_dep)) {
312.         dev_err(&pdev->dev, "Get filter_dep failed ret = %d\r\n",
313. ret);
314.         goto of_property_read_err;
315.     }
316.     dev_info(&pdev->dev, "filter_dep = %d\n", atc260x_adckeypad->filter
317. _dep);
318.
319.     /*get variance val */
320.     ret = of_property_read_u32(np, "variance", &(atc260x_adckeypad->var
321. iance));
322.     if ((ret) || (!atc260x_adckeypad->variance)) {
323.         dev_err(&pdev->dev, "Get variance failed ret = %d\r\n", ret);
324.         goto of_property_read_err;
325.     }
326.     dev_info(&pdev->dev, "variance = %d\n", atc260x_adckeypad-
327. >variance);
328.
329.     /*get left adc val*/
330.     atc260x_adckeypad->left_adc_val = devm_kzalloc(&pdev->dev,
331. sizeof(unsigned int) * (atc260x_adckeypad->keymapsize), GFP_KER
332. NEL);
333.     if (!atc260x_adckeypad->left_adc_val)
334.         goto free;
335.
336.     ret = of_property_read_u32_array(np, "left_adc_val",
337. (u32 *)atc260x_adckeypad->left_adc_val,
338. atc260x_adckeypad->keymapsize);
339.     if (ret) {
340.         dev_err(&pdev->dev, "Get left_adc_val failed ret = %d\r\n",
341. ret);
342.         goto free_left;
343.     }
344.
345.     /*get right adc val*/

```

```

337.     atc260x_adckeypad->right_adc_val = devm_kzalloc(&pdev->dev,
338.         sizeof(unsigned int) * (atc260x_adckeypad->keymapsize), GFP_KERNEL);
339.     if (!atc260x_adckeypad->right_adc_val)
340.         goto free;
341.
342.     ret = of_property_read_u32_array(np, "right_adc_val",
343.         (u32 *)atc260x_adckeypad->right_adc_val,
344.         atc260x_adckeypad->keymapsize);
345.     if (ret) {
346.         dev_err(&pdev->dev, "Get right_adc_val failed ret = %d\r\n", ret);
347.         goto free_right;
348.     }
349.
350.     for (i = 0; i < CHANNEL_NUM; ++i) {
351.         /*get key val*/
352.         atc260x_adckeypad->key_values[i] = devm_kzalloc(&pdev->dev,
353.             sizeof(unsigned int) * (atc260x_adckeypad->keymapsize), GFP_KERNEL);
354.         if (!atc260x_adckeypad->key_values[i])
355.             goto free;
356.
357.         ret = of_property_read_u32_array(np, of_key_val[i],
358.             (u32 *)atc260x_adckeypad->key_values[i],
359.             atc260x_adckeypad->keymapsize);
360.         if (ret) {
361.             dev_err(&pdev->dev, "Get key_values failed ret = %d\r\n", ret);
362.             goto free_key_values;
363.         }
364.         /*Malloc adc_buffer*/
365.         atc260x_adckeypad->adc_buffer[i] = devm_kzalloc(&pdev->dev,
366.             sizeof(unsigned int) * atc260x_adckeypad->filter_dep, GFP_KERNEL);
367.         if (!atc260x_adckeypad->adc_buffer[i])
368.             goto free_buffer;
369.         memset(atc260x_adckeypad->adc_buffer[i], 0xff,
370.             sizeof(unsigned int) * atc260x_adckeypad->filter_dep);
371.     }
372. #endif
373.
374.     /*
375.      * poll dev related
376.      */

```

```

377.     poll_dev = input_allocate_polled_device();
378.     if (!poll_dev) {
379.         ret = -ENOMEM;
380.         goto free_buffer;
381.     }
382.     atc260x_adckeypad->poll_dev = poll_dev;
383.
384.     poll_dev->private = atc260x_adckeypad;
385.     poll_dev->poll = atc260x_adckeypad_poll;
386.
387.     #if (ADCKEYPAD_DEBUG == 1)
388.         poll_dev->poll_interval = 5; /* msec */
389.     #else
390.         /*get poll period*/
391.         ret = of_property_read_u32(np, "poll_interval", &(poll_dev->poll_in
392. terval));
393.         if ((ret) || (!poll_dev->poll_interval)) {
394.             dev_err(&pdev->dev, "Get poll_interval failed \r\n");
395.             goto free_buffer;
396.         }
397.         dev_info(&pdev->dev, "poll_interval = %ums\n", poll_dev-
398. >poll_interval);
399.     #endif
400.
401.     input_dev = poll_dev->input;
402.     input_dev->evbit[0] = BIT(EV_KEY) | BIT(EV_REP) | BIT(EV_SW);
403.     input_dev->name = pdev->name;
404.     input_dev->phys = "atc260x_adckeypad/input3";
405.     input_dev->keycode = atc260x_adckeypad->key_values;
406.     input_dev->keycodesize = atc260x_adckeypad->keymapsize;
407.     input_dev->keycodemax = atc260x_adckeypad->keymapsize;
408.     input_dev->dev.parent = &pdev->dev;
409.     input_dev->id.bustype = BUS_HOST;
410.
411.     for (i = 0; i < atc260x_adckeypad->keymapsize; i++) {
412.         __set_bit(*(atc260x_adckeypad->key_values[0] + i), input_dev->k
413. eybit);
414.         __set_bit(*(atc260x_adckeypad->key_values[1] + i), input_dev->k
415. eybit);
416.     }
417.     __clear_bit(KEY_RESERVED, input_dev->keybit);
418.     __set_bit(KEY_POWER, input_dev->keybit);
419.     __set_bit(KEY_POWER2, input_dev->keybit);
420.     __set_bit(KEY_VAL_HOLD, input_dev->swbit);

```

```

418.     input_set_capability(input_dev, EV_MSC, MSC_SCAN);
419.     ret = input_register_polled_device(poll_dev);
420.     if (ret) {
421.         dev_err(&pdev->dev, "%s() failed to register_polled_device, ret
422.         __func__, ret);
423.         goto free_polled;
424.     }
425.
426.     input_dev->timer.data = (long) input_dev;
427.     return 0;
428.
429. free_polled:
430.     platform_set_drvdata(pdev, NULL);
431.     input_free_polled_device(poll_dev);
432.
433. free_buffer:
434. free_key_values:
435. free_right:
436. free_left:
437. free:
438. of_property_read_err:
439.
440.     return ret;
441. }
442.
443. static int atc260x_adckeypad_remove(struct platform_device *pdev)
444. {
445.     struct atc260x_adckeypad_dev *atc260x_adckeypad =
446.         platform_get_drvdata(pdev);
447.
448.     platform_set_drvdata(pdev, NULL);
449.     input_unregister_polled_device(atc260x_adckeypad->poll_dev);
450.     input_free_polled_device(atc260x_adckeypad->poll_dev);
451.
452.     return 0;
453. }
454.
455. #ifdef CONFIG_PM
456. static int atc260x_adckeypad_suspend(struct platform_device *pdev,
457.     pm_message_t state)
458. {
459.     struct atc260x_adckeypad_dev *atc260x_adckeypad =
460.     platform_get_drvdata(pdev);

```

```

461.     cancel_delayed_work_sync(&atc260x_adckeypad->poll_dev->work);
462.     return 0;
463. }
464. static int atc260x_adckeypad_resume(struct platform_device *pdev)
465. {
466.     struct atc260x_adckeypad_dev *atc260x_adckeypad =
platform_get_drvdata(pdev);
467.
468.     schedule_delayed_work(&atc260x_adckeypad->poll_dev->work,
469.         msecs_to_jiffies(atc260x_adckeypad->poll_dev->poll_interval));
470.     return 0;
471. }
472. #else
473.     # define atc260x_adckeypad_suspend NULL
474.     # define atc260x_adckeypad_resume  NULL
475. #endif
476.
477. static const struct of_device_id atc260x_adckey_of_match[] = {
478.     {.compatible = "actions,atc2603a-adckeypad", },
479.     {.compatible = "actions,atc2603c-adckeypad", },
480.     {.compatible = "actions,atc2609a-adckeypad", },
481.     {}
482. };
483.
484. static struct platform_driver atc260x_adckeypad_driver = {
485.     .driver = {
486.         .name = "atc260x-adckeypad",
487.         .owner = THIS_MODULE,
488.         .of_match_table = of_match_ptr(atc260x_adckey_of_match),
489.     },
490.     .probe = atc260x_adckeypad_probe,
491.     .remove = atc260x_adckeypad_remove,
492.     .suspend = atc260x_adckeypad_suspend,
493.     .resume = atc260x_adckeypad_resume,
494. };
495.
496. module_platform_driver(atc260x_adckeypad_driver)
497.
498. /*static int __init atc260x_adckeypad_init(void) */
499. /*{ */
500. /*    return platform_driver_register(&atc260x_adckeypad_driver); */
501. /*} */
502. /*subsys_init(atc260x_adckeypad_init); */
503. /*//late_initcall(atc260x_adckeypad_init); */
504. /*static void __exit atc260x_adckeypad_exit(void) */

```

```

505.  /*{ */
506.  /*  platform_driver_unregister(&atc260x_adckeypad_driver); */
507.  /*} */
508.  /*module_exit(atc260x_adckeypad_exit); */
509.
510.
511.  MODULE_LICENSE("GPL");
512.  MODULE_DESCRIPTION("Asoc adckey drvier");
513.  MODULE_AUTHOR("sall.xie/Actions Semi, Inc");

```

测试源码

```

1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <unistd.h>
4.  #include <stdint.h>
5.  #include <sys/ioctl.h>
6.  #include <sys/fcntl.h>
7.  #include <sys/types.h>
8.  #include <sys/stat.h>
9.  #include <linux/input.h>
10.
11.  int main(void)
12.  {
13.      struct input_event ev_key;
14.      int fd;
15.      fd = open("/dev/input/event2", O_RDWR);
16.
17.      if (fd < 0) {
18.          perror("open device buttons");
19.          exit(1);
20.      }
21.      while(1) {
22.          read(fd, &ev_key, sizeof(struct input_event));
23.          if (EV_KEY == ev_key.type)
24.              printf("type:%d,code:%d,value:%d\n", ev_key.type, ev_key.cod
e, ev_key.value);
25.      }
26.      close(fd);
27.      return 0;
28.  }

```

