[AN]SNC86xx_ADC_KEY_ApplicationNote

V0.1

History

版本	发布时间	版本说明	作者	核准人
V0.1	21.3.29	适用于 SAR ADC driver V2.01	范紫阳	

目录

1 模块概述

SAR ADC 模块可将输入的信号转换为数字输出，输入范围为 0-3.3V，输出值为 0-4095。共有两个通道 CH0、CH1，其中 CH0 的精度更高，CH1 的转换速率更快。CH0 和 CH1 占用芯片两个不同的引脚，模块工作时一次只能处理一个通道的值，建议固定使用 CH0。

2 应用示例与接口说明

以 ADC 按键功能 app_adc_key 为例：

2.1 按键的配置

在 app_adc_key.h 中，有一组按键配置相关的宏：

```
/**
 * @name KEY_PARAM
 * @brief Key parameters
 * @{
 */
#define KEY_NULL                    0xFF
#define KEY_PARSE_THRES             80
#define KEY_SHORT_PRESS_THRES       8   //8*10 = 80 ms
#define KEY_NULL_THRES              50  //50*10 = 500 ms
/** @} KEY_PARAM */
```

其中，KEY_NULL 代表未检索到有按键按下时的返回值，不需要修改；KEY_PARSE_THRES 表示处理键值时，实际键值与预设按键对应键值的最大偏移量；KEY_SHORT_PRESS_THRES 为计数阈值，当计数超过该阈值时认为按键按下，由于相关处理函数是按 10ms 轮询的，因此将该阈值乘以 10ms 即为按键按下的时间阈值；KEY_NULL_THRES 为计数阈值，如果两次按下之间计数小于该阈值，则认为是一次双击，反之则为两次单击。

除了上述宏定义外，每个按键还有一个结构体做更多定义：

```
typedef struct{
    uint16_t keyVal;
    uint16_t longKeyThr;
    uint16_t longPressThr;
}key_cfg_t;
```

其中 keyVal 为按键键值，例如一个按键对应的电压若为 1.1V，则其对应的键值应设为(1.1/3.3)*4095=1365；longKeyThr 为该按键的长按计数阈值，当按键按下后计数超过该阈值时则认为为长按，反正则为短按；longPressThr 为计数阈值，在按键已经被认为长按的情况下，每经过该阈值，可再次返回 long press 信息。

2.2 添加按键定义

实际使用时，首先根据实际需求，枚举所有按键：

```
/**
 * @enum key_t
 * @brief Define key numbers.
 */
typedef enum{
    KEY_0_VOL_UP = 0,
    KEY_1_VOL_DOWN,
    KEY_2_PLAY_PAUSE,
    KEY_3,
    KEY_4,
    KEY_5,
    //...
    KEY_NUM
}key_t;
```

并且在 app_adc_key.c 中，为每个按键的具体配置赋值：

```
/* Private variables ---------------------------
key_cfg_t keyCfg[KEY_NUM] = {
    //Key 0
    {
        .keyVal = 1360,
        .longKeyThr = 30,    //300ms
        .longPressThr = 4,   //40ms
    },

    //Key 1
    {
        .keyVal = 600,
        .longKeyThr = 30,
        .longPressThr = 4,
    },

    //Key 2
    {
        .keyVal = 600,//0,
        .longKeyThr = 30,
        .longPressThr = 4,
    },

    //Key 3
    {
        .keyVal = 1360,//2000,
        .longKeyThr = 30,
        .longPressThr = 4,
    },

    //Key 4
    {
        .keyVal = 600,//0,//2000,
        .longKeyThr = 30,
        .longPressThr = 4,
    },

    //Key 5
    {
        .keyVal = 0,//2000,
        .longKeyThr = 30,
        .longPressThr = 4,
    },
};
```

2.3 添加按键事件处理

按键的事件包括四类：短按，长按，双击，释放：

```
typedef enum{
    KEY_EVENT_NONE = 0,
    KEY_EVENT_SHORT,
    KEY_EVENT_LONG,
    KEY_EVENT_DOUBLE,
    KEY_EVENT_RELEASE,
}key_event_t;
```

每个按键可在 key_event_handler 中，自由处理各类事件，目前提供如下例子：

（1）音量增减键：

```
case KEY_0_VOL_UP:
    switch(keyEvent)
    {
    case KEY_EVENT_SHORT:
    case KEY_EVENT_LONG:    //identical function of short and long
        uart_printf("vol up\n");
        #ifdef SUPPPORT_HID_KEYBOARD
        is_hid_key(HID_VOL_UP);
        #endif
        break;
    case KEY_EVENT_RELEASE:
        #ifdef SUPPPORT_HID_KEYBOARD
        is_hid_key(HID_NULL);
        #endif
        break;
    default:
        break;
    }
    break;
```

音量增减键的特点是：短按和长按的功能相同，都是发送对应键值；长按的功能在 long press 时需要反复执行；每次键值发送完后，都需要在下一次处理时再发送 release 的键值；不需要双击功能。

因此在编写代码时，长短按共用一段执行代码，release 使用另一段。

（2）播放暂停键：

```
case KEY_2_PLAY_PAUSE:
    switch(keyEvent)
    {
    case KEY_EVENT_SHORT:   //no long, only play/pause once
        uart_printf("play pause\n");
        #ifdef SUPPPORT_HID_KEYBOARD
        is_hid_key(HID_PLAYPAUSE);
        #endif
        break;
    case KEY_EVENT_RELEASE:
        #ifdef SUPPPORT_HID_KEYBOARD
        is_hid_key(HID_NULL);
        #endif
        break;
    default:
        break;
    }
    break;
```

播放暂停键与音量增减键的区别在于，它不需要长按的功能。因此他的事件代码不包括长按的 case。

（3）长按与短按功能不同的按键：

```
case KEY_3:
    switch(keyEvent)
    {
    case KEY_EVENT_SHORT:
        uart_printf("Key 3 short\n");
        break;
    case KEY_EVENT_LONG:
        uart_printf("Key 3 long that only excuted once\n");
        break;
    default:
        break;
    }
    break;

case KEY_4:
    switch(keyEvent)
    {
    case KEY_EVENT_SHORT:
        uart_printf("Key 4 short\n");
        break;
    case KEY_EVENT_LONG:
        uart_printf("Key 4 long\n");
        break;
    default:
        break;
    }
    break;
```

Key3 和 key4 为长短按功能不同的例子，他们之间的区别是长按功能是一次只执行一次还是在 long press 期间需要反复执行。不过他们在事件处理函数中的代码并不会体现这一区别，而是在 2.4 中体现。

（4）支持双击的按键

```
case KEY_5:
    switch(keyEvent)
    {
    case KEY_EVENT_SHORT:
        uart_printf("Key 5 single short\n");
        break;
    case KEY_EVENT_LONG:
        uart_printf("Key 5 long that only excuted once\n");
        break;
    case KEY_EVENT_DOUBLE:
        uart_printf("Key 5 double\n");  //double press func
    default:
        break;
```

Key5 是支持双击的例子。同样的，他在 event 处理函数中只需要多写一个 case 即可。区别于其他按键的配置需要参照 2.4。

2.4 配置状态机

ADC 按键的状态机分为两个函数：key_process_stable 和 key_process_configurable，其中前者不需要用户配置，后者则需要根据按键的具体需求参照例程进行配置。

key_process_configurable 函数中需要根据前级处理得到的中间状态 keyIntrmState，根据每个按键的不同需求执行不同的操作。中间状态 keyIntrmState 包括如下五种类型：

```
typedef enum{
    KEY_INTERMEDIATE_STATE_NONE = 0,
    KEY_INTERMEDIATE_STATE_SHORT,
    KEY_INTERMEDIATE_STATE_LONG,
    KEY_INTERMEDIATE_STATE_LONG_JUSTIFY_FAIL,
    KEY_INTERMEDIATE_STATE_DOUBLE_SHORT_JUSTIFY_FAIL,
}key_intermediate_t;
```

其中，前置的 key_process_stable 函数会在：

(1) 按键连续按下达到 KEY_SHORT_PRESS_THRES 短按阈值时，返回 KEY_INTERMEDIATE_STATE_SHORT。如果该按键不需要区分长短按、单双击功能，那么此时就可以执行 event 事件代码了。

(2) 按键连续按下达到长按阈值 longKeyThr，和长按状态下每次达到 longPressThr 阈值时，返回 KEY_INTERMEDIATE_STATE_LONG。

(3) 如果需要区分长短按功能，那么需要在中间状态返回 KEY_INTERMEDIATE_STATE_SHORT 时将标志位 needJustifyLongKeyFlag 置位，判断该次按键究竟是长按还是短按。如果是长按，会和 (2) 一样返回常规的 KEY_INTERMEDIATE_STATE_LONG；如果最终确认是短按，则会返回 KEY_INTERMEDIATE_STATE_LONG_JUSTIFY_FAIL，这也是这种情况下执行短按功能的正确事机。

(4) 如果需要区分单双击的功能，那么首先需要如(3)一样，区分第一下按键是长按还是短按，如果是长按则应执行长按的功能，如果是短按，则需要将标志位 needJustifySingleShortFlag 置位，判断接下来是否会有第二次按下。如果超出 KEY_NULL_THRES 还没有第二次按下，则认为是一次单击，返回 KEY_INTERMEDIATE_DOUBLE_SHORT_JUSTIFY_FAIL；如果在 KEY_NULL_THRES 阈值内有第二次按下，则不管第二次按键是长按还是短按，都将返回第二次 KEY_INTERMEDIATE_STATE_SHORT，并执行双击功能。

(5)其他的任何时候，都会返回 KEY_INTERMEDIATE_STATE_NONE。

下面是几种不同情况的例程：

（1）音量增减键

```
case KEY_0_VOL_UP:      //Case 0, 2, simplest
case KEY_1_VOL_DOWN:
    switch(keyIntrmState)
    {
    case KEY_INTERMEDIATE_STATE_SHORT:
    case KEY_INTERMEDIATE_STATE_LONG:
        keyHandle.rlsFlag = 1;
        break;
    default:
        if(keyHandle.rlsFlag)
        {
            keyHandle.rlsFlag = 0;
            return KEY_EVENT_RELEASE;
        }
        break;
    }
    return keyIntrmState;
```

不需要判断长短按、单双击，因此在 short 和 long 中间状态到来时直接将其返回给 event_handler 处理即可；但需要注意由于 hid 按键的特性，每次执行完相应功能后都需要

将 hid 键值复位，因此需要在遇到 short 和 long 中间状态时将 rlsFlag 置位，中间状态返回 none 时对 rlsFlag 进行判断并执行 release 功能。

（2）播放暂停键

```
case KEY_2_PLAY_PAUSE:  //no long function
    switch(keyIntrmState)
    {
    case KEY_INTERMEDIATE_STATE_SHORT:
        keyHandle.rlsFlag = 1;
        break;
    default:
        if(keyHandle.rlsFlag)
        {
            keyHandle.rlsFlag = 0;
            return KEY_EVENT_RELEASE;
        }
        break;
    }
    return keyIntrmState;
```

播放暂停键和(1)的区别在于不需要在中间状态返回 long 时将 rlsFlag 置位。因为他并不会执行长按功能。

（3）区分长短按，长按功能只执行一次：

```
case KEY_3:     //Case 1, 2, need justify long
    switch(keyIntrmState)
    {
    case KEY_INTERMEDIATE_STATE_SHORT:
        keyHandle.needJustifyLongKeyFlag = 1;
        break;
    case KEY_INTERMEDIATE_STATE_LONG:
        if(!keyHandle.longKeyFlag)  //Long Key function that excuted only once a time
        {
            keyHandle.longKeyFlag = 1;
            return KEY_EVENT_LONG;
        }
        break;
    case KEY_INTERMEDIATE_STATE_LONG_JUSTIFY_FAIL:
        return KEY_EVENT_SHORT;
        break;
    default:
        break;
    }
    break;
```

如前所述，当需要区分长短按时，需要在中间状态返回 short 时，将 needJustifyLongKeyFlag 置位，在返回 long_justify_fail 时才执行真正的短按功能。

至于长按功能只执行一次的做法则是在长按中将 longKeyFlag 置位。

（4）区分长短按，长按功能反复执行：

```
case KEY_4:
    switch(keyIntrmState)
    {
    case KEY_INTERMEDIATE_STATE_SHORT:
        keyHandle.needJustifyLongKeyFlag = 1;
        break;
    case KEY_INTERMEDIATE_STATE_LONG:
        return KEY_EVENT_LONG;  //Long Key function that excuted continuously
        break;
    case KEY_INTERMEDIATE_STATE_LONG_JUSTIFY_FAIL:
        return KEY_EVENT_SHORT;
        break;
    default:
        break;
    }
    break;
```

和 Key3 的区别是不需要置位 longKeyFlag。

（5）区分单双击：

```
case KEY_5:     //Case 1, 3, need justify long and double
    switch(keyIntrmState)
    {
    case KEY_INTERMEDIATE_STATE_SHORT:
        if(!keyHandle.needJustifySingleShortFlag)
        {
            keyHandle.needJustifyLongKeyFlag = 1;
        }
        else
        {
            keyHandle.needJustifySingleShortFlag = 0;
            keyHandle.rlsFlag = 1;
            return KEY_EVENT_DOUBLE;
        }
        break;
    case KEY_INTERMEDIATE_STATE_LONG:
        if(!keyHandle.longKeyFlag)  //Long Key function that excuted only once a time
        {
            keyHandle.longKeyFlag = 1;
            if(!keyHandle.rlsFlag)
                return KEY_EVENT_LONG;
        }
        break;
    case KEY_INTERMEDIATE_STATE_LONG_JUSTIFY_FAIL:
        keyHandle.needJustifySingleShortFlag = 1;
        break;
    case KEY_INTERMEDIATE_STATE_DOUBLE_SHORT_JUSTIFY_FAIL:
        return KEY_EVENT_SHORT; //single short
        break;
    default:
        break;
    }
    break;
```

这是最复杂的一种情形。如前所述，在中间状态 short 时，首先将 needJustifyLongKeyFlag 置位，判断第一次按键是否是长按；如果返回 long_justify_fail，代表第一次按键是短按，此时将 needJustifySingleShortFlag 置位，判断是否会有第二次按键。如果超出阈值还未有第二次按键，则会返回 double_short_justify_fail，此时才应执行短按的功能；如果阈值内有第二次按下，则会第二次返回 short，此时应执行双击功能。