PRBMD02 Application Note

BSP application note

Introduction

This application is to describe the Key-press related routine in PRBMD02 SDK.

Introduction 2

1. What is BSP_Button 4

2. Basic configuration 4

2.1.Document description 4

2.2.Button composition 4

2.3.Button classification 4

2.4.Hardware Configuration 4

2.5.Application example 5

2.5.1.GPIO only 5

2.5.2.KSCAN only 6

2.5.3.Both GPIO and KSCAN button exist 6

3. Expanded configuration 10

3.1.Configurable button type 10

3.2.Configurable timing parameter 10

3.3.Max press key number 10

1. What is BSP_Button

BSP_Button is a set of button middleware processing programs. The hardware depends on GPIO and KSCAN. The upper layer implements the monitoring of buttons through TASK scheduling. It supports button type press, long press to start, long press to hold, and release.

2. Basic configuration

To use BSP_Button, you need to do proper hardware configuration, and add the BSP_Button task Bsp_Btn_Init on the TASK dispatch sequence, and add the BSP_Button event handler Bsp_Btn_ProcessEvent on the TASK event handler.

2.1. Document description

To use BSP_Button, you need to include the following files and add hardware initialization code in the application task.

- **bsp_gpio.c**: GPIO button driver, included when using GPIO buttons.
- **kscan.c**: KSCAN button driver, included when using KSCAN button.
- **bsp_button.c**: BSP_Button button detection logic.
- **bsp_button_task.c**: BSP_Button scheduling mechanism.

2.2. Button composition

The key data type is BYTE, and different bits have different meanings.

- The upper 2 bits indicate the key type, press, long press to start, long press to hold, release, etc.
- The lower 6 bits represent the key value, and the value range is 0x00~0x3F. Among them, 0x30~0x3F are reserved for the system, which can be used to the value is 0x00~0x2F.

2.3. Button classification

The keys support individual keys and combination keys.

Individual buttons: A button corresponds to a GPIO or a KSCAN row cross point.

- Combination button: Multiple single buttons form a combination button, such as multiple GPIO single buttons, multiple KSCAN single buttons
- key, etc. to form a combination key. For example, pressing P1 is a separate button, pressing P2 is a separate button, pressing P1P2 at the same time is combination keys.

Combination keys rely on individual keys. When using combination keys, the individual keys are in the front, and the combination keys are in the back. For example, there are 10 buttons in total, 8 individual buttons and 2 combination buttons, then the button value 0~7 corresponds to 8 individual buttons, and the button value 8~9 corresponds to 2 combination buttons

2.4. Hardware Configuration

Choose the appropriate underlying hardware as needed.

- **BSP_BTN_JUST_GPIO**: Only use GPIO buttons.
- **BSP_BTN_JUST_KSCAN**: Only use KSCAN button.

- **BSP_BTN_GPIO_AND_KSCAN**: Use GPIO button and KSCAN button at the same time.

Select the number of individual keys and combination keys as required.

- **BSP_SINGLE_BTN_NUM**: The number of individual keys.
- **BSP_COMBINE_BTN_NUM**: The number of combination keys, if not defined as 0.
- **BSP_TOTAL_BTN_NUM**: The total number of keys, the total number of keys is equal to the number of individual keys plus the number of combined keys.

If the system has both GPIO and KSCAN, you need to specify the number of individual buttons and the number of combined buttons for each type.

- **BSP_KSCAN_SINGLE_BTN_NUM**: The number of KSCAN single buttons.
- **BSP_GPIO_SINGLE_BTN_NUM:** The number of GPIO single buttons.
- **BSP_KSCAN_COMBINE_BTN_NUM**: The number of KSCAN combination keys, if not defined as 0.
- **BSP_GPIO_COMBINE_BTN_NUM**: The number of GPIO combination buttons, if not defined as 0.

If the system has both GPIO and KSCAN, the keys are arranged in the following way: KSCAN separate press --GPIO separate key -- combined key.

2.5. Application example

2.5.1. GPIO only

If Only GPIO button is needed, configure as following:

- **bsp_button_task.h** specifies the hardware configuration method, the number of buttons, etc.
- The number of GPIO pins and the idle level state are specified in **bsp_gpio.h**.
- Configure specific pins, callback functions, etc. in the application. If there is a combination button, you need to specify the corresponding relationship of the group button.

For example, if there are 3 individual buttons (P14, P15, P26), when there is no button, it is a high level, and if there are 2 combined buttons (P14P15, P14 P26), the callback function is **hal_bsp_btn_callback**.

The corresponding buttons are as following:

2.5.2.KSCAN only

If there is KSCAN only, configuration is as following:

- bsp_button_task.h specifies the hardware configuration method, the number of buttons, etc.
- The number of rows and columns used by KSCAN is specified in kscan.h.
- Configure specific pins, callback functions, etc. in the application. If there is a combination button, you need to specify the corresponding relationship of the group button.

For example, there are 4x4 individual keys (ROW: P0, P2, P25, P18) (COL: P1, P3, P24, P20), 2 combined keys (Row0Col0 and Row0Col1, Row0Col2 and Row0Col3), and the callback function is **hal_bsp_btn_callback** .

2.5.3.Both GPIO and KSCAN button exist

If the GPIO button and the KSCAN button exist at the same time, the pins used and the number of pins need to be configured respectively. If there is a combination button, the combination button relationship needs to be assigned.

- **bsp_button_task.h** specifies the hardware configuration method, the number of buttons, etc.
- The number of GPIO pins used and the idle level state are specified in **bsp_gpio.h**.
- The number of rows and columns used by KSCAN is specified in **kscan.h**.
- Configure specific pins, callback functions, etc. in the application. If there is a combination button, you need to specify the corresponding relationship of the group button.

Example: There are 4*4 kscan individual keys (ROW: P0, P2, P25, P18) (COL: P1, P3, P24, P20). There are 3 separate buttons (P14, P15, P26), and it is a high level when there is no button. 1 combination key (row0col2, row0col3), 1 combination key (P14P15). The callback function is **hal_bsp_btn_callback**.

GPIO type key mapping example

Press button value	Corresponding H/W action
0	P14 pressed
1	P15 pressed
2	P26 pressed
3	P14 and P15 pressed at the same time
4	P14 and P26 pressed at the same time

Reference code:

```
bsp_button_task.h:
#define BSP_BTN_HARDWARE_CONFIG                 BSP_BTN_JUST_GPIO
#define BSP_SINGLE_BTN_NUM                      (GPIO_SINGLE_BTN_NUM)
#define BSP_COMBINE_BTN_NUM                     (2)
#define BSP_TOTAL_BTN_NUM                       (BSP_SINGLE_BTN_NUM + BSP_COMBINE_BTN_NUM)

bsp_gpio.h:
#define GPIO_SINGLE_BTN_NUM                     3
#define GPIO_SINGLE_BTN_IDLE_LEVEL              1

bsp_btn_demo.c:
uint32_t usr_combine_btn_array[BSP_COMBINE_BTN] =
 {
        (BIT(0)|BIT(1)),
        (BIT(0)|BIT(2)),
};

void hal_bsp_btn_callback(uint8_t evt)
{
        LOG("gpio evt:0x%x ",evt);
        switch(BSP_BTN_TYPE(evt))
        {
        case BSP_BTN_PD_TYPE:
                LOG("press down ");
                break;
        case BSP_BTN_UP_TYPE:
                LOG("press up ");
                break;
        case BSP_BTN_LPS_TYPE:
                LOG("long press start ");
                break;
        case BSP_BTN_LPK_TYPE:
                LOG("long press keep ");
                break;
        default:
                LOG("unexpected ");
                break;
        }
        LOG("value:%d\n",BSP_BTN_INDEX(evt));
}
```

```
Gpio_Btn_Info gpio_btn_info = {
        {P14,P15,P26},
        hal_bsp_btn_callback,
};
void Demo_Init( uint8 task_id )
{
        if(PPlus_SUCCESS == hal_gpio_btn_init(&gpio_btn_info))
        {
                bsp_btn_gpio_flag = TRUE;
        }
        else
        {
                LOG("hal_gpio_btn_init error:%d\n",__LINE__);
        }
}
```

Press button value	H/W action
0	Row0, Col0 pressed
1	Row0, Col1 pressed
2	Row0, Col2 pressed
3	Row0, Col3 pressed
4	Row1, Col0 pressed
5	Row1, Col1 pressed
6	Row1, Col2 pressed
7	Row1, Col3 pressed
8	Row2, Col0 pressed
9	Row2, Col1 pressed

Press button value	H/W action
10	Row2, Col2 pressed
11	Row2, Col3 pressed
12	Row3, Col0 pressed
13	Row3, Col1 pressed
14	Row3, Col2 pressed
15	Row3, Col3 pressed
16	Row0Col0 and Row0Col1 pressed together
17	Row0Col2 and Row0Col3 pressed together

bsp_button_task.h:

```
#define BSP_BTN_HARDWARE_CONFIG                 BSP_BTN_JUST_KSCAN
#define BSP_SINGLE_BTN_NUM                      (NUM_KEY_ROWS * NUM_KEY_COLS)
#define BSP_COMBINE_BTN_NUM                     (2)
#define BSP_TOTAL_BTN_NUM                       (BSP_SINGLE_BTN_NUM + BSP_COMBINE_BTN_NUM)
```

kscan.h:

```
#define NUM_KEY_ROWS 4
#define NUM_KEY_COLS 4
```

bsp_btn_demo.c:

```
KSCAN_ROWS_e rows[NUM_KEY_ROWS] =
{KEY_ROW_P00,KEY_ROW_P02,KEY_ROW_P25,KEY_ROW_P18};
KSCAN_COLS_e cols[NUM_KEY_COLS] =
{KEY_COL_P01,KEY_COL_P03,KEY_COL_P24,KEY_COL_P20};
uint32_t usr_combine_btn_array[BSP_COMBINE_BTN_NUM] =
{
        (BIT(0)|BIT(1)),
        (BIT(2)|BIT(3)),
};
void hal_bsp_btn_callback(uint8_t evt)
{
        LOG("kscan evt:0x%x ",evt);
        switch(BSP_BTN_TYPE(evt))
        {
                case BSP_BTN_PD_TYPE:
                        LOG("press down ");
                        break;
                case BSP_BTN_UP_TYPE:
                        LOG("press up ");
                        break;
                case BSP_BTN_LPS_TYPE:
                        LOG("long press start ");
                        break;
                case BSP_BTN_LPK_TYPE:
                        LOG("long press keep ");
                        break;
                default:
                        LOG("unexpected ");
                        break;
        }
        LOG("value:%d\n",BSP_BTN_INDEX(evt));
}

void Demo_Init( uint8 task_id )
{
        hal_kscan_btn_check(hal_bsp_btn_callback);
        if(bsp_btn_kscan_flag != TRUE)
        {
                LOG("hal_kscan_btn_check error:%d\n",__LINE__);
        }
}
```

3. Expanded configuration

3.1. Configurable button type

- By default, only key press is supported. If you need to support key release and long key press, you need to configure it.
- **BSP_BTN_PRESS_RELEASE_ENABLE:** Support button release after definition.
- **BSP_BTN_LONG_PRESS_ENABLE:** After the definition, it supports long keys. The long keys include long press to start and long press to hold.

3.2. Configurable timing parameter

Four time parameters is able to be configured:

BTN_SYS_TICK:

Adjust to the cycle, the unit is ms

BTN_FILTER_TICK_COUNT:

Debounce time, BTN_SYS_TICK* BTN_FILTER_TICK_COUNT.

BTN_LONG_PRESS_START_TICK_COUNT:

Long press start time, BTN_SYS_TICK*BTN_LONG_PRESS_START_TICK_COUNT

BTN_LONG_PRESS_KEEP_TICK_COUNT:

Press button value	H/W action
0	Row0, Col0 pressed
1	Row0, Col1 pressed
2	Row0, Col2 pressed
3	Row0, Col3 pressed
4	Row1, Col0 pressed
5	Row1, Col1 pressed
6	Row1, Col2 pressed
7	Row1, Col3 pressed
8	Row2, Col0 pressed
9	Row2, Col1 pressed
10	Row2, Col2 pressed

Press button value	H/W action
11	Row2, Col3 pressed
12	Row3, Col0 pressed
13	Row3, Col1 pressed
14	Row3, Col2 pressed
15	Row3, Col3 pressed
16	P14 pressed
17	P15 pressed
18	P26 pressed
19	Row0Col2 and Row0Col3 pressed together
20	P14 and P15 pressed together

Long press hold time, BTN_SYS_TICK*BTN_LONG_PRESS_KEEP_TICK_COUNT.

3.3.Max press key number

The default configuration supports up to 48 keys, if not enough, you can modify the code as needed.

bsp_button_task.h:

```
#define BSP_BTN_HARDWARE_CONFIG

#define BSP_KSCAN_SINGLE_BTN_NUM
#define BSP_GPIO_SINGLE_BTN_NUM

#define BSP_KSCAN_COMBINE_BTN_NUM
#define BSP_GPIO_COMBINE_BTN_NUM
#define BSP_SINGLE_BTN_NUM          (BSP_KSCAN_SINGLE_BTN_NUM +\
                                        BSP_GPIO_SINGLE_BTN_NUM)
#define BSP_COMBINE_BTN_NUM         (BSP_KSCAN_COMBINE_BTN_NUM +\
                                        BSP_GPIO_COMBINE_BTN_NUM)
#define BSP_TOTAL_BTN_NUM           (BSP_SINGLE_BTN_NUM + \
                                        BSP_COMBINE_BTN_NUM)
```

kscan.h:

```
#define NUM_KEY_ROWS 4
#define NUM_KEY_COLS 4
```

bsp_btn_demo.c:

```
KSCAN_ROWS_e rows[NUM_KEY_ROWS] =
{KEY_ROW_P00,KEY_ROW_P02,KEY_ROW_P25,KEY_ROW_P18};
KSCAN_COLS_e cols[NUM_KEY_COLS] = {KEY_COL_P01,KEY_COL_P03,KEY_COL_P24,KEY_COL_P20};

BTN_T usr_sum_btn_array[BSP_TOTAL_BTN_NUM];
uint32_t usr_combine_btn_array[BSP_COMBINE_BTN_NUM] =
{
        (BIT(2)|BIT(3)),
        (BIT(16)|BIT(17)),
};

void hal_bsp_btn_callback(uint8_t evt)
{
        LOG("kscan evt:0x%x ",evt);
        switch(BSP_BTN_TYPE(evt))
        {
        case BSP_BTN_PD_TYPE:
                LOG("press down ");
                break;
        case BSP_BTN_UP_TYPE:
                LOG("press up ");
                break;
        case BSP_BTN_LPS_TYPE:
                LOG("long press start ");
                break;
        case BSP_BTN_LPK_TYPE:
                LOG("long press keep ");
                break;
        default:
                LOG("unexpected ");
                break;
        }
        LOG("value:%d\n",BSP_BTN_INDEX(evt));
}
```

```
Gpio_Btn_Info gpio_btn_info = { {
        P14,P15,P26},
        hal_bsp_btn_callback,
};
void Demo_Init( uint8 task_id )
 {
        hal_kscan_btn_check(hal_bsp_btn_callback);
        if(bsp_btn_kscan_flag != TRUE)
        {
                LOG("hal_kscan_btn_check error:%d\n",__LINE__);
        }
        if(PPlus_SUCCESS == hal_gpio_btn_init(&gpio_btn_info))
        {
                bsp_btn_gpio_flag = TRUE;
        }
        else
        {
                LOG("hal_gpio_btn_init error:%d\n",__LINE__);
        }
}
```