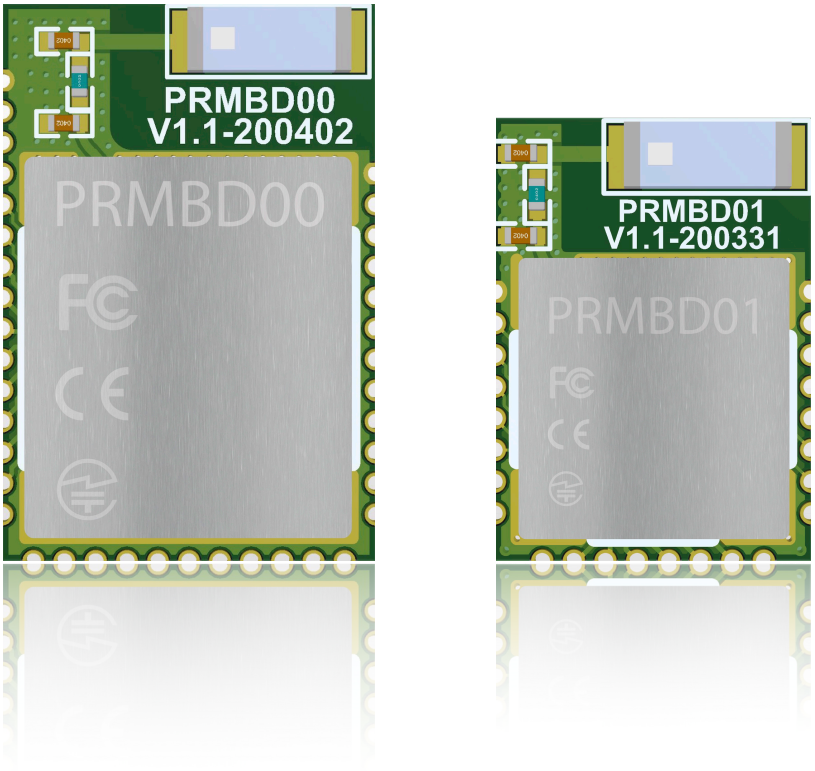
PRBMD0x Application Note

General Purpose I/O



Disclaimer

Liability Disclaimer

K-Solution Consulting Co. Ltd reserves the right to make changes without further notice to the product to improve reliability, function or design. K-Solution Consulting Co. Ltd does not assume any liability arising out of the application or use of any product or circuits described herein.

Life Support Applications

K-Solution Consulting Co. Ltd's products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. K-Solution Consulting Co. Ltd customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify K-Solution Consulting Co. Ltd for any damages resulting from such improper use or sale.

1. Introduction

This document introduce the GPIO of PRBMD0x and the application.

PRBMD00 is up to 35 GPIOs, and the default status is listed as following table.

Many of GPIOs are multiplex with other interface, such as I2C, UART…etc, by configuring IOMUX. Correct configuration is needed to ensure they work properly. Please note that P31-P34 multiplex with proprietary SPI for internal testing purpose only.

PRBMD00	PRBMD01	Default MODE	Default IN_OUT	IRQ	Wakeup	ANA_IO
GPIO_P00	√	jtag_dout	OUT	√	√	
GPIO_P01	√	jtag_din	IN	√	√	
GPIO_P02	√	jtag_tm	IN	√	√	
GPIO_P03	√	jtag_clk	IN	√	√	
GPIO_P04	N/A	GPIO	IN	√	√	
GPIO_P05	N/A	GPIO	IN	√	√	
GPIO_P06	N/A	GPIO	IN	√	√	
GPIO_P07	N/A	GPIO	IN	√	√	
TEST_MODE	√					
GPIO_P09	√	GPIO	IN	√	√	
GPIO_P10	√	GPIO	IN	√	√	
GPIO_P11	N/A	GPIO	IN	√	√	ADC_CH1N_P11
GPIO_P12	N/A	GPIO	IN	√	√	ADC_CH1P_P12
GPIO_P13	N/A	GPIO	IN	√	√	ADC_CH2N_P13
GPIO_P14	√	GPIO	IN	√	√	ADC_CH2P_P14
GPIO_P15	√	GPIO	IN	√	√	ADC_CH3N_P15
GPIO_P16	√	XTALI(ANA)	ANA	√	√	
GPIO_P17	√	XTALO(ANA)	ANA	√	√	
GPIO_P18	√	GPIO	IN		√	
GPIO_P19	N/A	GPIO	IN		√	
GPIO_P20	√	GPIO	IN		√	ADC_CH3P_P20
GPIO_P21	N/A	GPIO	IN		√	
GPIO_P22	N/A	GPIO	IN		√	
GPIO_P23	√	GPIO	IN		√	
GPIO_P24	√	GPIO	IN		√	
GPIO_P25	√	GPIO	IN		√	

GPIO_P26	N/A	GPIO	IN	√
GPIO_P27	N/A	GPIO	IN	√
GPIO_P28	N/A	GPIO	IN	√
GPIO_P29	N/A	GPIO	IN	√
GPIO_P30	N/A	GPIO	IN	√
GPIO_P31	√	spi_t_ssn	IN	√
GPIO_P32	√	spi_t_rx	IN	√
GPIO_P33	√	spi_t_tx	OUT	√
GPIO_P34	√	spi_t_sck	IN	√

Table 1 GPIO power up default attribute

i. GPIO mode

Here are some items related to GPIO.

a. Analog and Digital port

Some GPIO can be configure as digital or analog port, such as P16 and P17.

b. Input and output

All GPIO can be configured as digital input or output.

c. Pull-high and Pull-low

Each GPIO can be enabled a pull high/low resistor

- Floating: unknown statu
- Weak pull high: enable a 150KΩ resistor to VDD, small driving current
- Strong pull high: enable a 10KΩ resistor to VDD, higher driving current
- Pull low: enable a 100KΩ resistor to ground

d. Interrupt and wake up

P00~P17 supports GPIO interrupt and wake up

P18~P34 supports wake up only

ii. Functional block

GPIOs are multiplexed with other functional blocks, such as ADC. These functional block will be described in separated documents.

2. API

i. Definition, constants and variables

a. NUMBER_OF_PINS

number of GPIO pins

b. NUMBER_OF_IRQ_PINS

number of GPIO pin supporting interrupt

c. GPIO_Pin_e

GPIO global definition, GPIO_DUMMY is a pseudo pin GPIO

```
typedef enum{
	GPIO_P00  =  0,   P0  =  0,
	GPIO_P01  =  1,   P1  =  1,
	GPIO_P02  =  2,   P2  =  2,
	GPIO_P03  =  3,   P3  =  3,
	GPIO_P04  =  4,   P4  =  4,
	GPIO_P05  =  5,   P5  =  5,
	GPIO_P06  =  6,   P6  =  6,
	GPIO_P07  =  7,   P7  =  7,
	TEST_MODE =  8,   P8  =  8,
	GPIO_P09  =  9,   P9  =  9,
	GPIO_P10  =  10,  P10 =  10,
	GPIO_P11  =  11,  P11 =  11,  Analog_IO_0 = 11,
	GPIO_P12  =  12,  P12 =  12,  Analog_IO_1 = 12,
	GPIO_P13  =  13,  P13 =  13,  Analog_IO_2 = 13,
	GPIO_P14  =  14,  P14 =  14,  Analog_IO_3 = 14,
	GPIO_P15  =  15,  P15 =  15,  Analog_IO_4 = 15,
	GPIO_P16  =  16,  P16 =  16,  XTALI = 16,
	GPIO_P17  =  17,  P17 =  17,  XTALO = 17,
	GPIO_P18  =  18,  P18 =  18,  Analog_IO_7 = 18,
	GPIO_P19  =  19,  P19 =  19,  Analog_IO_8 = 19,
	GPIO_P20  =  20,  P20 =  20,  Analog_IO_9 = 20,
	GPIO_P21  =  21,  P21 =  21,
	GPIO_P22  =  22,  P22 =  22,
	GPIO_P23  =  23,  P23 =  23,
	GPIO_P24  =  24,  P24 =  24,
	GPIO_P25  =  25,  P25 =  25,
	GPIO_P26  =  26,  P26 =  26,
	GPIO_P27  =  27,  P27 =  27,
	GPIO_P28  =  28,  P28 =  28,
	GPIO_P29  =  29,  P29 =  29,
	GPIO_P30  =  30,  P30 =  30,
	GPIO_P31  =  31,  P31 =  31,
	GPIO_P32  =  32,  P32 =  32,
	GPIO_P33  =  33,  P33 =  33,
	GPIO_P34  =  34,  P34 =  34,
	GPIO_DUMMY =  0xff,
}GPIO_Pin_e;
```

d. GPIO input/output status

name	definition
GPIO_PIN_ASSI_NONE	Idle
GPIO_PIN_ASSI_OUT	Output
GPIO_PIN_ASSI_IN_IRQ	Input, support interrupt
GPIO_PIN_ASSI_IN_WAKEUP	Input, support wakeup
GPIO_PIN_ASSI_IN_IRQ_AND_WAKEUP	Input, support wakeup and interrupt

e. GPIO_ioe

GPIO as input or output

IE	Input
OEN	Output

f. BitAction_e

GPIO as Functional block or GPIO

Bit_DISABLE	Disable
Bit_ENABLE	Enable

g. IO_Pull_Type_e

GPIO's pull high/low

FLOATING	Floating
WEAK_PULL_UP	Weak pull high
STRONG_PULL_UP	Strong pull high
PULL_DOWN	Pull low

h. IO_Wakeup_Pol_e

Configure trigger of wakeup or interrupt

POSEDGE	Rising edge for interrupt and wakeup
NEGEDGE	Falling edge for interrupt and wakeup

i. Fmux_Type_e Configure GPIO multiplex function

Related to pin configuration

definition	description
IIC0_SCL	IIC0 clock
IIC0_SDA	IIC0 data
IIC1_SCL	IIC1 clock
IIC1_SDA	IIC1 data
I2S_SCK	I2S clock
I2S_WS	I2S LR
I2S_SDO_0	I2S channel 0 data out
I2S_SDI_0	I2S channel 1 data in
UART_TX	UART Tx, only for GPIO P9
UART_RX	UART Rx, only for GPIO P10
PWM0	PWM channel 0
PWM1	PWM channel 1
PWM2	PWM channel 2
PWM3	PWM channel 3
PWM4	PWM channel 4
PWM5	PWM channel 5
SPI_0_SCK	SPI0 clock
SPI_0_SSN	SPI0 chip selection
SPI_0_TX	SPI0 Tx
SPI_0_RX	SPI0 Rx

definition	description
SPI_1_SCK	SPI1 clock
SPI_1_SSN,	SPI1 chip selection
SPI_1_TX	SPI1 Tx
SPI_1_RX	SPI1 Rx
CHAX	Quadrature decoder
CHBX	Quadrature decoder
CHIX	Quadrature decoder
CHAY	Quadrature decoder
CHBY	Quadrature decoder
CHIY	Quadrature decoder
CHAZ	Quadrature decoder
CHBZ	Quadrature decoder
CHIZ	Quadrature decoder
CLK1P28M	reserved
ADCC	reserved
I2S_SDO_1	I2S channel1 output
I2S_SDO_2	I2S channel2 output
I2S_SDO_3	I2S channel3 output
I2S_SDI_1	I2S channel1 input
I2S_SDI_2	I2S channel2 input
I2S_SDI_3	I2S channel3 input

ii. Data structure

a. gpioin_Hdl_t

GPIO interrupt callback function and wake-up callback function type.

b. gpioin_Ctx_t

GPIO input control structure

type	parameter	description
bool	enable	input pin enable flag
uint8_t	pin_state	pin level status
gpioin_Hdl_t	posedgeHdl	Rising edge callback function pointer
gpioin_Hdl_t	negedgeHdl	Falling edge callback function pointer

c. gpio_Ctx_t

GPIO global control structure

type	parameter	description
bool	state	input pin enable pin
uint8_t	pin_assignments	Pin mode configuration
gpioin_Ctx_t	irq_ctx	Input pin processing structure.

iii. API

a. int hal_gpio_init(void)

GPIO module initialization: initialize hardware, enable interrupts, configure interrupt priority, etc.

This function needs to be set when the system is initialized, and is generally called in the hal_init () function. For specific information, please refer to example code for more details.

- parameter: none
- Return value

PPlus_SUCCESS	initialisation success
other value	refers to <error.h>

b. void hal_gpio_pin_init(GPIO_Pin_e pin,GPIO_ioe type)

Configure GPIO as input or output

- Parameter

type	parameter	description
GPIO_Pin_e	pin	GPIO pin
GPIO_ioe	type	configure GPIO as input or output

- Return value: none

c. int gpio_pin0to3_pin31to34_control(GPIO_Pin_e pin, uint8_t en)

Default setting of PIO_P00~PIO_P03、PIO_P31~ PIO_P34 are non-GPIO, and they can be set as GPIO by this function.

- Parameter

type	parameter	description
GPIO_Pin_e	pin	GPIO pin。
uint8_t	en	1：set pin as GPIO pin 0：set pin as default setting

- Return value

PPlus_SUCCESS	Initialisation success
other value	refers to <error.h>

d. int hal_gpio_fmux(GPIO_Pin_e pin, BitAction_e value)

Configure pin function as GPIO or multiplexed function block.

Parameter

type	parameter	description
GPIO_Pin_e	Pin	GPIO pin。
BitAction_e	value	Bit_ENABLE：set IO pin as multiplexed function Bit_DISABLE：set IO pin as GPIO

Return value

PPlus_SUCCESS	Success
other value	refers to <error.h>

e. int hal_gpio_fmux_set(GPIO_Pin_e pin,Fmux_Type_e type)

Configure pin's function

- Parameter

type	parameter	description
GPIO_Pin_e	Pin	GPIO pin。
Fmux_Type_e	type	Function of the IO pin

- Return value

PPlus_SUCCESS	Success
other value	Refers to <error.h>

f. int hal_gpio_cfg_analog_io(GPIO_Pin_e pin, BitAction_e value)

Configure GPIO as digital or analog port.

- Parameter

type	parameter	description
GPIO_Pin_e	pin	GPIO pin。
BitAction_e	value	Bit_ENABLE：set the pin as analog Bit_DISABLE：set the pin as digital

- Return value

PPlus_SUCCESS	initialisation success
other value	refers to <error.h>

g. int hal_gpioin_register(GPIO_Pin_e pin, gpioin_Hdl_t posedgeHdl, gpioin_Hdl_t negedgeHdl)

注册GPIO的输入模式，该模式下支持中断回调和唤醒回调，包括上升沿回调和下降沿回调。

- 参数

type	parameter	description
GPIO_Pin_e	Pin	GPIO pin。
gpioin_Hdl_t	posedgeHdl	上升沿回调函数，可以为NULL。
gpioin_Hdl_t	negedgeHdl	下降沿回调函数，可以为NULL。

- Return value

PPlus_SUCCESS	成功。
其他数值	参考<error.h>

h. int hal_gpioin_unregister(GPIO_Pin_e pin)

Un-register GPIO as input. Interrupt, wakeup rising/falling edge trigger will be disable after un-registered.

- Parameter

type	parameter	description
GPIO_Pin_e	Pin	GPIO pin

- Return value

PPlus_SUCCESS	Success
Other value	Refers to <error.h>

i. int hal_gpio_pull_set(GPIO_Pin_e pin,IO_Pull_Type_e type)

Configure pull high/low of the pin

- Parameter

type	parameter	description
GPIO_Pin_e	Pin	GPIO pin
Pull_Type_e	type	type of pull high or low

- Return value

PPlus_SUCCESS	Success
Other value	refers to <error.h>

j. void hal_gpio_write(GPIO_Pin_e pin, uint8_t en)

Write 1 or 0 to a particular GPIO

- Parameter

type	parameter	description
GPIO_Pin_e	Pin	GPIO pin
uint8_t	en	0：GPIO pin set to 0 other value：GPIO pin set to 1

- Return value: none

k. uint32_t hal_gpio_read(GPIO_Pin_e pin)

Read particular GPIO value

- Parameter

type	parameter	description
GPIO_Pin_e	Pin	GPIO pin。

- Return value

TRUE	the pin is at high level
FALSE	the pin is at low level

l. int hal_gpioin_enable(GPIO_Pin_e pin)

Enable GPIO as input, this function will also configure the pin attribute, feature and return function.

- Parameter

type	parameter	description
GPIO_Pin_e	Pin	GPIO pin。

- Return value

PPlus_SUCCESS	Success
Other value	Refers to <error.h>

m. int hal_gpioin_disable(GPIO_Pin_e pin)

Disable GPIO as input

- Parameter

type	parameter	description
GPIO_Pin_e	Pin	GPIO pin。

- Return value

PPlus_SUCCESS	Success
Other value	refers to <error.h>

n. int gpio_interrupt_enable(GPIO_Pin_e pin, IO_Wakeup_Pol_e type)

Configure GPIO interrupt interrupt，enable interrupt

- Parameter

type	parameter	description
GPIO_Pin_e	pin	GPIO pin
IO_Wakeup_Pol_e	type	Rising or falling edge trigger

- Return value

PPlus_SUCCESS	Success
Other value	Refers to <error.h>

o. int gpio_interrupt_disable(GPIO_Pin_e pin)

Configure GPIO interrupt register, disable interrupt

- Parameter

type	parameter	description
GPIO_Pin_e	pin	GPIO pin

- Return value

PPlus_SUCCESS	Success
Other value	Refers to <error.h>

p. void io_wakeup_control(GPIO_Pin_e pin, BitAction_e value)

Configure GPIO wake up feature

- Parameter

type	parameter	description
GPIO_Pin_e	pin	GPIO pin。
BitAction_e	value	Bit_ENABLE：wakeup enable Bit_DISABLE：wakeup disable

- Return value: none

q. void hal_gpio_wakeup_set (GPIO_Pin_e pin,IO_Wakeup_Pol_e type)

Configure GPIO wake up mode：rising or falling edge

- Parameter

type	parameter	description
GPIO_Pin_e	pin	GPIO pin
IO_Wakeup_Pol_e	type	wake up polarity, either rising or falling edge

- Return value: none

r. void gpio_sleep_handler(void)

Callback function before the system enters sleep, you can configure information such as wake-up

- Parameter: none
- Return value: none

s. void gpio_wakeup_handler(void)

Callback function after the system wakes up from sleep, manually call the wake-up processing function.

- Parameter: none
- Return value: none

t. void gpioin_wakeup_trigger(GPIO_Pin_e pin)

Responding to GPIO wakeup event

- Parameter

type	parameter	description
GPIO_Pin_e	pin	GPIO pin

- Return value: none

u. void hal_GPIO_IRQHandler(void)

GPIO interrupt handle function

- 参数: none
- Return value: none

v. void gpioin_event(uint32 int_status, uint32 polarity)

GPIO interrupt event processing, traverse all interrupt events in response to all GPIO.

- Parameter

type	parameter	description
uint32	status	Interrupt flag
uint32	polarity	Either rising or falling edge trigger the flag

- Return value: none

w. void gpioin_event_pin(GPIO_Pin_e pin, IO_Wakeup_Pol_e type)

The interrupt event processing function of a single GPIO will call a user-defined preset callback function

- Parameter

type	parameter	description
GPIO_Pin_e	pin	GPIO pin。
IO_Wakeup_Pol_e	type	Interrupt polarity, rising or falling edge

- Return value: none

3. Application example

i. Digital output

The following sample code is setting IO as digital port

Pin to be configure：set GPIO_P14 as digital port

```
// sample code
hal_gpio_pin_init(GPIO_P14,OEN);// configure as digital port
while(1)
{
	hal_gpio_write(GPIO_P14, 1); //Output high
	WaitMs(50);

	hal_gpio_write(GPIO_P14, 0); //Output low
	WaitMs(50);
}
```

ii. Digital input

This example shows setting IO as digital input

Pin to be configure：set GPIO_P14 as digital port

```
// sample code
hal_gpio_pin_init(GPIO_P14,IE);//P14 is configured as input mode
static bool gpioin_state;
gpioin_state = hal_gpio_read(GPIO_P14);//Read voltage level on P14
LOG("gpioin_state:%d\n",gpioin_state);

while(1)
{
	if(gpioin_state != hal_gpio_read(GPIO_P14))
	{
		gpioin_state = hal_gpio_read(GPIO_P14);
		LOG(“gpioin_state:%d\n",gpioin_state);// printer if there is change on P14
	}
}
```

iii. Interrupt

P00~P07，P09~P17 supports interrupt when configure as digital input.

P14 is set at digital input, enable interrupt

P15 is set as digital output, and toggle at 2s duty cycle

Connect P14 and P15，change of P15 will trigger P14 to generate interrupt

sample code：

```
//P14 rising edge interrupt callback function
void pos_callback(IO_Wakeup_Pol_e type)
{
	if(POSEDGE == type){
			LOG("posedge\n");
	}
}
//P14 rising edge interrupt callback function
void neg_callback(IO_Wakeup_Pol_e type)
{
	if(NEGEDGE == type){
		LOG("negedge\n");
	}
}
```

```
//P15 is configured as a digital output function to output high and low levels at duty cycle of 2s
hal_gpio_pin_init(GPIO_P15,OEN);

//P14 configured as pull down
hal_gpio_pull_set(GPIO_P14,PULL_DOWN);
//configure P14 call back function
hal_gpioin_register(GPIO_P14,pos_callback,neg_callback);

while(1)
{
	hal_gpio_write(GPIO_P15, 1);
	WaitMs(1000);
	hal_gpio_write(GPIO_P15, 0);
	WaitMs(1000);
}
```

iv. Wakeup

P00~P07，P09~P34 support wakeup when configured as digital input

P14 is set as input, change of edge will trigger wakeup

when OSAL processing job, system will enter sleep mode，wake up every 1s and scan process event

Through the gpio wakeup callback function, you can know whether gpio wakes up the system

it is possible to use jumper to change P14 status and observe the callback function

```
//OSAL_gpio.c
const pTaskEventHandlerFn tasksArr[] =
{
  LL_ProcessEvent,
  Gpio_Wakeup_ProcessEvent,//Event response function entry
};

void osalInitTasks( void )
{
  uint8 taskID = 0;

  tasksEvents = (uint16 *)osal_mem_alloc( sizeof( uint16 ) * tasksCnt);
  osal_memset( tasksEvents, 0, (sizeof( uint16 ) * tasksCnt));

  LL_Init( taskID++);
  Gpio_Wakeup_Init(taskID);//initialisation function entry
}
```

```
//gpio_demo.c
static uint8 gpio_wakeup_TaskID;

//callback function
void posedge_callback_wakeup(GPIO_Pin_e pin,IO_Wakeup_Pol_e type){
	if(type == POSEDGE){
		LOG("pos:io:%d type:%d\n",pin,type);
	}
	else{
		LOG("err\n");
	}
}

void negedge_callback_wakeup(GPIO_Pin_e pin,IO_Wakeup_Pol_e type){
	if(type == NEGEDGE){
		LOG("neg:io:%d type:%d\n",pin,type);
	}
	else{
		LOG("err\n");
	}
}

//structure used
typedef struct gpioin_wakeup_t{
	GPIO_Pin_e pin;
	gpioin_Hdl_t posedgeHdl;
	gpioin_Hdl_t negedgeHdl;
}gpioin_wakeup;

#define GPIO_WAKEUP_PIN_NUM 3
gpioin_wakeup gpiodemo[GPIO_WAKEUP_PIN_NUM] = {
	GPIO_P14,posedge_callback_wakeup,negedge_callback_wakeup,
	GPIO_P23,posedge_callback_wakeup,negedge_callback_wakeup,
	GPIO_P31,posedge_callback_wakeup,negedge_callback_wakeup,
};

//function initialisation
void Gpio_Wakeup_Init(uint8 task_id ){
	uint8_t i = 0;
	static bool gpioin_state[GPIO_WAKEUP_PIN_NUM];
```

```
//Points to notice
1. P00~P03：The default is JTAG, and the GPIO can be configured to wake up the system.
2. P08: Select the pin for the mode, it can not be used for other purposes.
3. P04~P07,P11~P15,P18~P30：The default GPIO can wake up the system.
3. P09~P10：The default is GPIO, and the SDK configures it as UART by default. If P09 ~ P10 are
   used as GPIO, please pay attention to mapping the UART function to other IO.
4. P16~P17：default setting is 32K crystal oscillator. When using internal rc, it can be set to GPIO.
5. P31~P34：Default bit SPIF interface, can configure bit GPIO, support wake up。
```

v. Pulse measurement

Pulse_measure in GPIO demonstrates how to measure pulse type and width

```
P14 set as digital input, determine the pulse width
```

```
//OSAL_gpio.c
const pTaskEventHandlerFn tasksArr[] =
{
  LL_ProcessEvent,
  Pulse_Measure_ProcessEvent, //Event response function entry
};

void osalInitTasks( void )
{
  uint8 taskID = 0;

  tasksEvents = (uint16 *)osal_mem_alloc( sizeof( uint16 ) * tasksCnt);
  osal_memset( tasksEvents, 0, (sizeof( uint16 ) * tasksCnt));

  LL_Init( taskID++);
  /* Application */
  Pulse_Measure_Init( taskID); //Initialization function entry
}
```

```
//gpio_demo.c
static uint8 pulseMeasure_TaskID; //Pulse measurement task ID
typedef struct {
    bool        enable;
    bool        pinstate;
    uint32_t    edge_tick;
}gpioin_Trig_t;

typedef struct {
    GPIO_Pin_e   pin;//IO used in the test
    bool        type;//脉冲type
    uint32_t    ticks;//脉冲长度
}gpioin_pulse_Width_measure_t;

gpioin_pulse_Width_measure_t measureResult ={//Configure the GPIO used
    .pin = GPIO_P14,
};

static gpioin_Trig_t gpioTrig = {
  .enable = FALSE,
  .edge_tick = 0,
};

void plus_edge_callback(void){
    LOG("pulse:%d %d\n",measureResult.type,measureResult.ticks);
}

//回调函数
void pulse_measure_callback(GPIO_Pin_e pin,IO_Wakeup_Pol_e type)
{
    if(gpioTrig.enable == FALSE)
    {
        gpioTrig.enable = TRUE;
        gpioTrig.edge_tick = hal_systick();
        return;
    }
    measureResult.type = type;
    measureResult.ticks = hal_ms_intv(gpioTrig.edge_tick);

    if(measureResult.ticks < 10)
```

vi. Timer

Timer_Task in the example GPIO demonstrates the use of timers

```
this example demonstrate the use of timer
osal_start_timerEx( timer_TaskID, TIMER_1S_ONCE , 1000);
set timer at 1000ms，work once
osal_start_reload_timer( timer_TaskID, TIMER_2S_CYCLE , 2000);
set the timer at 2000ms, repetitive, until function osal_stop_timerEx stop the timer
```

```
//OSAL_gpio.c
const pTaskEventHandlerFn tasksArr[] =
{
  LL_ProcessEvent,
Timer_Demo_ProcessEvent,
};

void osalInitTasks( void )
{
  uint8 taskID = 0;

  tasksEvents = (uint16 *)osal_mem_alloc( sizeof( uint16 ) * tasksCnt);
  osal_memset( tasksEvents, 0, (sizeof( uint16 ) * tasksCnt));

  LL_Init( taskID++);
  /* Application */
Timer_Demo_Init(taskID);
}
```

```
//gpio_demo.c
static uint8 timer_TaskID;
#define TIMER_1S_ONCE                    0x0001
#define TIMER_2S_CYCLE                   0x0004

//initialisation
void Timer_Demo_Init( uint8 task_id ){
        timer_TaskID = task_id;

        osal_start_timerEx( timer_TaskID, TIMER_1S_ONCE , 1000 );
    osal_start_reload_timer( timer_TaskID, TIMER_2S_CYCLE , 2000);
}

//event handle
uint16 Timer_Demo_ProcessEvent( uint8 task_id, uint16 events ){
        static uint8 count1 = 0,count2 = 0;
        static bool  timer_cycle_enable = TRUE;

        if(task_id != timer_TaskID){
                return 0;
        }

  if ( events & TIMER_1S_ONCE ){
                LOG("1s:once only mode\n");
                osal_start_timerEx( timer_TaskID, TIMER_1S_ONCE , 1000);

                if(timer_cycle_enable == FALSE){
                        if(++count1 >= 10 ){
                                osal_start_reload_timer( timer_TaskID, TIMER_2S_CYCLE , 2000);

                                LOG("2s:recycle mode start\n");
                                timer_cycle_enable = TRUE;
                                count1 = 0;
                        }
                }
    return (events ^ TIMER_1S_ONCE);
  }

        if ( events & TIMER_2S_CYCLE ){
                        LOG("2s:recycle mode\n");
```

g. numeric button

When GPIO is used as digital input, it can be configured as a button。

key.h contains the following definition：

HAL_KEY_NUM：Number of keys

HAL_KEY_EVENT：The application layer needs to be assigned an event handling key, and the event handling code cannot be modified.

HAL_KEY_SUPPORT_LONG_PRESS：Support long press

HAL_KEY_DEBOUNCD：Debounce timer, unit is ms。

HAL_KEY_LONG_PRESS_TIME：Time needs long press valid, unit is ms。

Point to note

1.P00~P03 default setting is JTAG, need to disable if use as press button

2.P08 is test mode selection pin only

3.P09~P10 is default as UART port, if use as button, UART can map to other IO

4.P16~P17 is supposed to connect to 32.768k crystal by default, need to set to use internal RC if using them as buttons

5.P18~P34 supports wakeup but not interrupt, not suitable for push button

Buttons can be configured with information：

1. the pin to be used
2. level on the pin when idle
3. Callback function, there are short press to press, short press to release, long press to release branch.
4. The application layer needs to assign an event HAL_KEY_EVENT to the key, and the event handling code is not changed.
5. If you need to long press the function, you need to open HAL_KEY_SUPPORT_LONG_PRESS, and add the long press processing code.

The key_Task in the example GPIO demonstrates how to use the key in OSAL.

```
/*
Two keys are configured
The pins used are P14 and P15
Low level when idle
The callback function is key_press_evt*/
void Key_Demo_Init(uint8 task_id)
{
    uint8_t i = 0;

    key_TaskID = task_id;
    LOG("gpio key demo start...\n");

    key_state.key[0].pin = GPIO_P14;
    key_state.key[1].pin = GPIO_P15;

    for(i = 0; i < HAL_KEY_NUM; ++i){
        key_state.key[i].state = HAL_STATE_KEY_IDLE;
        key_state.key[i].idle_level = HAL_LOW_IDLE;

        if(key_state.key[i].pin == GPIO_P16){
            hal_pwrmgr_register(MOD_USR2,NULL,P16_wakeup_handler);
            hal_gpio_cfg_analog_io(key_state.key[i].pin,Bit_DISABLE);
            LOG("P16 is used\n");
        }
            else if(key_state.key[i].pin == GPIO_P17){
            hal_pwrmgr_register(MOD_USR3,NULL,P17_wakeup_handler);
            hal_gpio_cfg_analog_io(key_state.key[i].pin,Bit_DISABLE);
            LOG("P17 is used\n");
        }
        else if((key_state.key[i].pin == GPIO_P09) || (key_state.key[i].pin == GPIO_P10)){
            uart_port_reconfig();
        }
    }

    key_state.task_id = key_TaskID;
    key_state.key_callbank = key_press_evt;
    key_init();
}
```

```
/*
button press callback function
HAL_KEY_EVT_PRESS：short press
HAL_KEY_EVT_RELEASE：short press release
HAL_KEY_EVT_LONG_RELEASE：long press release
*/
static void key_press_evt(uint8_t i,key_evt_t key_evt)
{
    LOG("\nkey index:%d gpio:%d ",i,key_state.key[i].pin);
    switch(key_evt)
    {
        case HAL_KEY_EVT_PRESS:
            LOG("key(press down)\n");
#ifdef HAL_KEY_SUPPORT_LONG_PRESS
    osal_start_timerEx(key_TaskID,KEY_DEMO_LONG_PRESS_EVT,HAL_KEY_LONG_PRESS_TIME);
#endif
            break;

        case HAL_KEY_EVT_RELEASE:
            LOG("key(press release)\n");
            break;

#ifdef HAL_KEY_SUPPORT_LONG_PRESS
        case HAL_KEY_EVT_LONG_RELEASE:
            hal_pwrmgr_unlock(MOD_USR1);
            LOG("key(long press release)\n");
            break;
#endif

        default:
            LOG("unexpect\n");
            break;
    }
}
```

```
/*
HAL_KEY_EVENT：The middleware code used for the key and event handling code cannot be modified
KEY_DEMO_LONG_PRESS_EVT：Long press and press, can be configured as needed
*/
uint16 Key_ProcessEvent( uint8 task_id, uint16 events )
{
    if(task_id != key_TaskID){
        return 0;
    }

    if( events & HAL_KEY_EVENT)
{                                                                    //do not modify,key will use it
        for (uint8 i = 0; i < HAL_KEY_NUM; ++i){
            if ((key_state.temp[i].in_enable == TRUE)||
            (key_state.key[i].state == HAL_STATE_KEY_RELEASE_DEBOUNCE)){
                gpio_key_timer_handler(i);
            }
        }
        return (events ^ HAL_KEY_EVENT);
    }

#ifdef HAL_KEY_SUPPORT_LONG_PRESS
    if( events & KEY_DEMO_LONG_PRESS_EVT){

        for (int i = 0; i < HAL_KEY_NUM; ++i){
            if(key_state.key[i].state == HAL_KEY_EVT_PRESS){
                LOG("key:%d gpio:%d      ",i,key_state.key[i].pin);
                LOG("key(long press down)");

osal_start_timerEx(key_TaskID,KEY_DEMO_LONG_PRESS_EVT,HAL_KEY_LONG_PRESS_TIME);//2s

                //user app code long press down process
            }
        }
        return (events ^ KEY_DEMO_LONG_PRESS_EVT);
    }
#endif

 // Discard unknown events
 return 0;
```

4. Reference

This document is base on Application note PHY62XX_GPIO_Application_Note_v0.5 from PHYPlus semiconductors.