PRBMD0x Application Note

SDK user guide

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.

Content

1. Introduction

This document is base on PHY62xx SDK from PHYPlus, describing firmware development for PRBMD0x with details description and examples.

Examples in this document is for reference only, facilitating development of firmware, K-Solution Consulting Co. Ltd. takes no responsibility for any example listed in this document.

If there is any question, it is welcomed to drop me an email: sales@k-sol.com.hk

2. Quick Start

SDK provide a groups of sample code, which can be operated on PRBMD0x, user can develop their own firmware base on these examples.

2.1. SDK directory

```
PHY62XXSDK
├─components                ; SDK components, including BLE API, GATT profile, drivers and other components
├─example                   ; example
│  ├─ble_central            ;
│  ├─ble_peripheral         ;
│  │  ├─alternate_iBeacon   ; alternate iBeacon example
│  │  ├─ancs                ; Apple Notification Center Service example
│  │  ├─bleI2C_RawPass      ; I2S tunnelling example
│  │  ├─bleSmartPeripheral  ; General peripheral example
│  │  ├─bleUart-RawPass     ; UART tunnelling example
│  │  ├─eddystone           ; eddystone example
│  │  ├─HIDKeyboard         ; HID example
│  │  ├─hrs                 ; Heart rate profile example
│  │  ├─iBeacon             ; iBeacon example
│  │  ├─otaDemo             ; Basic OTA example
│  │  ├─pwmLight            ; example of LED control by PWM, by BLE command
│  │  ├─RawAdv              ; simple boardcasting example, for tire pressure monitor
│  │  ├─Sensor_Broadcast    ;
│  │  ├─wrist               ; General example for sport bracelet
│  │  ├─wrist_aptm          ; General example, real time timer base on AP Timer + OSAL Timer
│  │  └─XIPDemo             ; Example of running within flash, for application not requiring realtime response
│  ├─OTA                    ;
│  │  ├─OTA_internal_flash  ; OTA bootloader
│  │  └─OTA_upgrade_2ndboot ; Special example for upgrading OTA bootloader
│  └─peripheral             ;
│     ├─adc                ; ADC driver example
│     ├─ap_timer           ; AP timer driving example
│     ├─fs                  ; File system example
│     ├─gpio                ; GPIO demo example
│     ├─kscan               ; 4x4 keypad example
│     ├─lcd_ST7789VW        ; 240x240 TFT display example
│     ├─pwm                 ; PWM demo example
│     ├─qdec                ; QDEC demo example
│     ├─spiflash            ; SPI ext. device example
│     ├─voice               ; Audio sampling example
│     ├─voice_sbc           ; SBC coding format audio sampling example
│     └─watchdog            ; Watchdog example
├─lib                       ; lib and .h document, including Bluetooth stack and Font library
│  └─font                   ; Font resource document
└─misc                      ; ROM symbol table and others
```

example directory inculds sample code for each feature, and a hex file is stored in the bin directory which user can program directly into PRBMD0x.

2.2. Installation of development environment

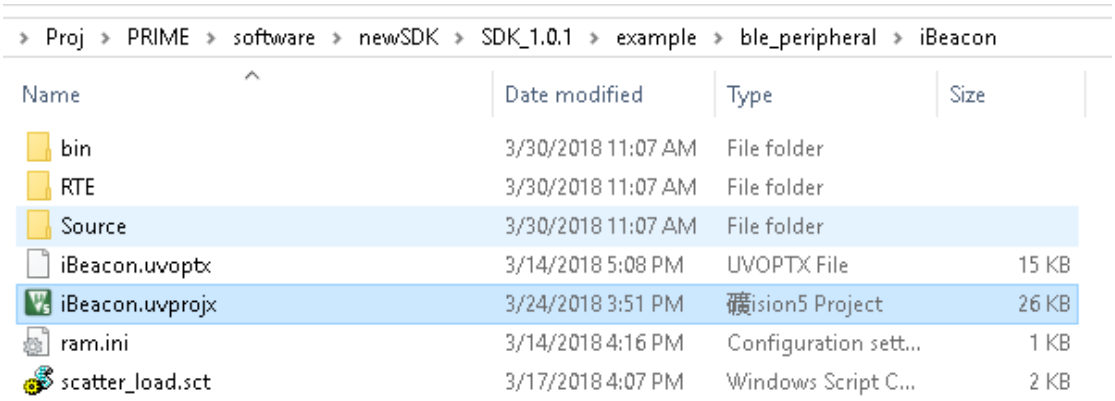
IDE installation step is as following：

- Copy SDK to working directory.
- Install **MDK Keil5 for ARM IDE**.
- Open, tuning and compile example code from MDK IDE.

图 3

2.3. compile and operation example

- Deleted programmed firmware by software tool: PlyPlusKit （Please refers <PhyPlusKit User's Guide.pdf> for details operation） 。
- Open any project in SDK, for example, the iBeacon project in directory example/bleperipheral/iBeacon/ iBeacon.jvprojx：



- Compile the project (or adjust the code if needed).
- Firmware is now ready for operate on PRBMD0x.

2.4. Tuning and programming

- on the MDK button tool bar, click **Option for target** button and open a project dialog box.

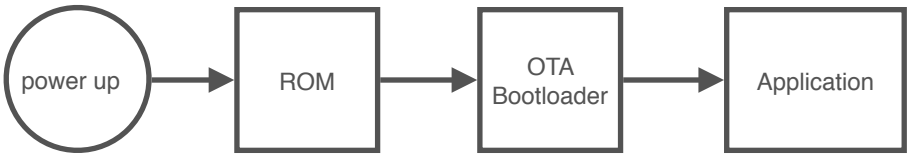
- In the Preprocessor Symbols' Define in the C / C ++ tab, developers can change the corresponding precompiled macro：
  - CFG_SLEEP_MODE=PWR_MODE_SLEEP ：Enable low power mode, during the firmware program execution, it will go to sleep during the idle process, after sleep, the debugger cannot debug and track, and the breakpoint is invalid
  - CFG_SLEEP_MODE=PWR_MOD_ENO_SLEEP ：Turn off the low-power mode, the processor has been awake during the firmware program execution。
  - DEBUG_INFO=1：Enable debugging information, output by default through the serial port P9(Tx),P10(Rx)
  - DEBUG_INFO=0：Turn off debugging information.
- Debug code：
  - Please use the PhyPlusKit tool to erase programmed firmware. Before erasing, please confirm that the TM pin is connected to a high level, and then apply a Reset .
  - Connect TM to low level and apply reset after Flash Erase
  - Clicke Debug button on the tool bar to download Image to SRAM for on line debugging.
- Firmware programming：
  - Erase the Flash and program, details on PhyPlusKit user guide.

3. Platform and driver

3.1. Firmware platform

Firmware for PRBMD0x consists of three major blocks: ROM, OTA Bootloader and application firmware, where OTA Bootloader is optional. Boot up processes are as following:

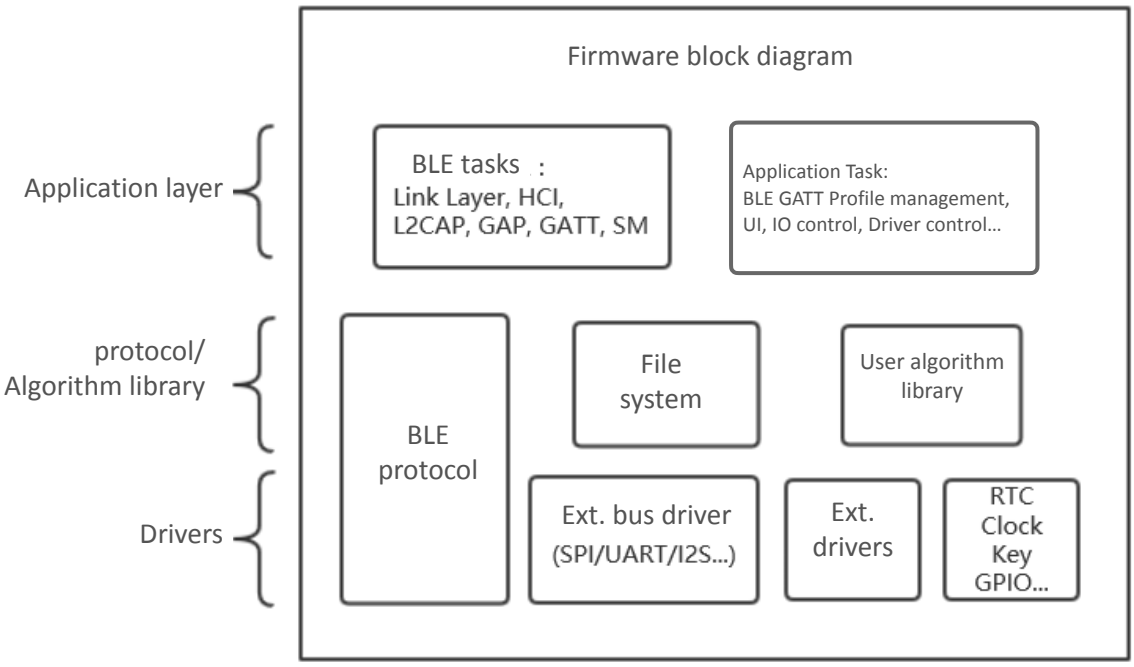
OTA boot up：



non OTA boot up：

- ROM部分：Start Bootloader or Application and BLE protocol API, depends on the TM pin level (High for programming mode; Low for Normal operation mode.
- OTA Boot loader：Used to guide Application and handle OTA upgrade
- Application：Application code, most of the secondary development work is concentrated in the Application section.

3.2. firmware block diagram



The SDK does not use a third-party RTOS, but the Task concept is abstracted at the application layer. For BLE applications, the following Tasks are required. Each Task includes an initialisation function and an event processing function.

Applications generally define one or more tasks, and typical scenarios (routines) generally use only one application task. Tasks such as HMI, peripheral control, BLE broadcast and connection configuration, and loading of GATT Profile are all implemented in Task. Within the task and between tasks can interact and communicate through the API provided by OSAL.

Tasks (Task initialization & Task event response function)	Description
LL_Init() LL_ProcessEvent(uint8, uint16)	Link layer initialization and corresponding event processing function.
HCI_Init() HCI_ProcessEvent(uint8, uint16)	HCI layer initialization and corresponding event processing function.
L2CAP_Init() L2CAP_ProcessEvent(uint8, uint16)	L2CAP layer initialization and corresponding event processing function.
GAP_Init() GAP_ProcessEvent(uint8, uint16)	GAP layer initialization and corresponding event processing function.
GATT_Init() GATT_ProcessEvent(uint8, uint16)	GATT layer initialization and corresponding event processing function.
SM_Init() SM_ProcessEvent(uint8, uint16)	SM（Safety management）layer initialization and corresponding event processing function.

GAPRole_Init() GAPRole_ProcessEvent(uint8, uint16)	GAP layer initialization and corresponding event processing function.
GATTServApp_Init() GATTServApp_ProcessEvent(uint8, uint16)	GATT layer initialization and corresponding event processing function.

Bond Manager task will be included if SMP is needed：

GAPBondMgr_Init() GAPBondMgr_ProcessEvent(uint8, uint16)	For supporting SMP feature

3.2.1. OSAL

SDK provides a set of APIs for system-level operations, including message mechanisms, timers, and heap management

APIs

3.2.1.1. uint8 osal_set_event(uint8 task_id, uint16 event_flag)

Used to send an event to a Task, and then the Task event processing function of the Task will respond to the event.

- Parameter

Type	Parameter name	Description
uint8	task_id	Task ID
uint16	event_flag	Event ID, each Bit corresponds to an event, and a Task can declare up to 16 Event Types.

- Return value:

0	Success
Other values	Refers to <comdef.h>

3.2.1.2. uint8 osal_clear_event(uint8 task_id, uint16 event_flag)

Clean an event。

- Parameter

Type	Parameter name	Description
uint8	task_id	Task ID
uint16	event_flag	Event ID, each Bit corresponds to an event, and a Task can declare up to 16 Event Types.

- Return value:

0	Success
Other values	Refers to <comdef.h>

3.2.1.3. uint8 osal_msg_send(uint8 task_id, uint8 *msg_ptr)

Send a message to Task

- Parameter

Type	Parameter name	Description
uint8	task_id	Task ID
Uint8*	msg_ptr	Message pointer

- Return value:

0	Success
Other values	Refers to <comdef.h>

3.2.1.4. uint8 *osal_msg_receive(uint8 task_id)

To receive the message, this function is applied to the event processing function of Task. The corresponding event is **SYS_EVENT_MSG (0x8000)** This event is specifically reserved for message processing

After receiving the event **SYS_EVENT_MSG**, the message pointer is obtained through this function. If the message buffer is allocated through **osal_mem_alloc**, it needs to be released through **osal_mem_free** after use.

- Parameter

Type	Parameter name	Description
uint8	task_id	Task ID

- Return value:

uint8*	Message pointer
NULL	No message received

3.2.1.5.uint8 osal_start_timerEx(uint8 task_id, uint16 event_id, uint32 timeout_value)

Start an application Timer, the system will send an event to the specified task when the timeout is reached, the timer will automatically close after completing an event, and will not be resent.

- Parameter

Type	Parameter name	Description
uint8	task_id	Task ID
uint16	event_id	Event ID, need to be declared in the specified Task
uint32	timeout_value	Timeout time, in milliseconds.

- Return value:

SUCCESS	Success

NO_TIMER_AVAIL	Fail to start timer

3.2.1.6. uint8 osal_start_reload_timerEx(uint8 task_id, uint16 event_id, uint32 timeout_value)

Start an application Timer, the system will send an event to the specified task when the timeout is reached, the **Typetimer** will send an event to the task at the specified time interval after starting, until the function **osal_stop_timerEx ()** stops the timer.

- Parameter

Type	Parameter name	Description
uint8	task_id	Task ID
uint16	event_id	Event ID, need to be declared in the specified Task
uint32	timeout_value	Timeout time, in milliseconds.

- Return value:

SUCCESS	Success
NO_TIMER_AVAIL	Fail to start timer

3.2.1.7. uint8 osal_stop_timerEx(uint8 task_id, uint16 event_id)

Stop an application Timer

- Parameter

Type	Parameter name	Description
uint8	task_id	Task ID
uint16	event_id	Event ID, need to be declared in the specified Task

- Return value:

SUCCESS	Success
INVALID_EVENT_ID	Timer ID not exist （or not started）

3.2.1.8.uint32 osal_get_timeoutEx(uint8 task_id, uint16 event_id)

Get the remaining timeout time of a running Timer in milliseconds.

- Parameter

Type	Parameter name	Description
uint8	task_id	Task ID
uint16	event_id	Event ID, need to be declared in the specified Task

- Return value:

uint32	The remaining waiting time, if the timer has not started, returns 0.

3.3. Hardware driver

3.3.1. Introduction

This section describes the hardware module driver and external bus driver. For the definition of the hardware module, please refer to Section 3.1.3.1. In addition to the actual hardware driver, a virtual module is provided for virtual control. Use case, which includes the application of the virtual module MOD_USR0.

3.3.1.1. MODULE_e

Module ID

Value	Name	Description
0	MOD_NONE MOD_SOFT_RESET MOD_CPU	Normally it is None effect device. The two aliases MOD_SOFT_RESET and MOD_CPU are temporarily reserved and have not taken effect.
1	MOD_LOCKUP_RESET_EN	None temporarily.
2	MOD_WDT_RESET_EN	None temporarily.
3	MOD_DMA	DMA module
4	MOD_AES	AES module
5	MOD_TIMER	Timer module
6	MOD_WDT	Watchdog module
7	MOD_COM	No effect at the moment
8	MOD_UART	UART module
9	MOD_I2C0	I2C Bus 1
10	MOD_I2C1	I2C Bus 2
11	MOD_SPI0	SPI Bus 1
12	MOD_SPI1	SPI Bus 2
13	MOD_GPIO	GPIO module
14	MOD_I2S	I2S module
15	MOD_QDEC	QDEC（Quadrature decoder） module
16	MOD_RNG	Random generator module
17	MOD_ADCC	ADC module
18	MOD_PWM	PWM module
19	MOD_SPIF	internal SPI Flash module
20	MOD_VOC	VOC module
31	MOD_KSCAN	Key Scan module
32	MOD_USR0	Virtual module 0, reserved for system use, is used to manage interrupt priority.

33~39	MOD_USR1~8	The application is used as needed, and can manage the hibernation of the application, and manage the additional operations of the wake-up and sleep of the application.

3.3.2. Clock

Clock module mainly provides system clock related configuration, including module clock switch, 32K clock source selection and other operations.

3.3.2.1. Definition

3.3.2.1.1. CLK32K_e

32K clock source selection

CLK_32K_XTAL	Select external 32.768K oscillator as clock source
CLK_32K_RCOSC	Select internal RC as clock source

3.3.2.2. Data structure: None

3.3.2.3. APIs

3.3.2.3.1. void clk_gate_enable(MODULE_e module)

Enable the clock of the hardware module

- Parameter

Type	Parameter name	Description
MODULE_e	module	Hardware module ID

- Return value: None

3.3.2.3.2. void clk_gate_disable(MODULE_e module)

Disable the clock of the hardware module

- Parameter

Type	Parameter name	Description
MODULE_e	module	Hardware module ID

- Return value: None

3.3.2.3.3. void clk_reset(MODULE_e module)

Reset module

- Parameter

Type	Parameter name	Description
MODULE_e	module	Hardware module ID

- Return value: None

3.3.2.3.4. uint32_t clk_hclk(void)

Obtain HCLK value

- Parameter: None
- Return value: Returns the HCLK value

3.3.2.3.5. uint32_t clk_pclk(void)

Obtain PCLK value

- Parameter: None
- Return value:: Returns PCLK value

3.3.2.3.6. void hal_rtc_clock_config(CLK32K_e clk32Mode)

Configure RTC clock source

- Parameter

Type	Parameter name	Description
CLK32K_e	clk32Mode	RTC clock source

- Return value: None

3.3.2.3.7.uint32_t hal_systick(void)

Obtain the system tick value. The tick value is 32bitNone symbols, and the time unit is 625 microseconds.

- Parameter: None
- Return value:: 32bit None symbolic system tick value

3.3.2.3.8.uint32_t hal_ms_intv(uint32_t tick)

Time difference from the previous time snapshot, in milliseconds, enter Parameter as the system tick.

- Parameter

Type	Parameter name	Description
uint32_t	tick	The tick value to be compared, the system tick recorded at a certain time before.

- Return value:: Time difference in milliseconds.

3.3.3. retention SRAM

In sleep mode, SRAM can be configured to keep or not hold data as needed. Each SRAM can be independently switched. If it is configured to keep, RAM data can be kept after sleep wake-up, otherwise the data will be lost.

Usually, the application configures retention according to the actual SRAM usage. Generally configured in the hal_init function of <main.c>, please refer to API: hal_pwrmgr_RAM_retention (uint32_t sram)

There are total of 5 available SRAM, please refer to the following table for specific information：

RAM ID	Size	Address space	Description
SRAM0	32K Byte	1FFF_0000~1FFF_7FFF	The application is shared with the protocol stack and must be turned on in sleep mode.
SRAM1	32K Byte	1FFF_8000~1FFF_FFFF	SRAM1 ~ SRAM4 can be turned off or enabled according to the application. It is recommended to turn off when not needed, which helps reduce power consumption.
SRAM2	64K Byte	2000_0000~2000_FFFF	
SRAM3	8K Byte	2001_0000~2001_1FFF	
SRAM4	2K Byte	2001_2000~2001_27FF	

3.3.4. GPIO

Provide GPIO related operations, including GPIO configuration, pull-down settings, event-driven model of GPIO input mode, GPIO wake-up operation, etc.

3.3.4.1. Definition

3.3.4.1.1.GPIO pin definition

GPIO_DUMMY is defined as a virtual pin, generally used as a None effect pin.

```
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;
```

3.3.4.1.2.GPIO_ioe

GPIO input or output configuration.

IE	Configure as input
OEN	Configure as output

3.3.4.1.3.BitAction_e

IO pin is configured as GPIO mode or as a function pin.

Bit_DISABLE	Configure as GPIO
Bit_ENABLE	Configure as function pin

3.3.4.1.4.IO_Pull_Type_e

Configure pin up and down mode.

FLOATING	The configuration is None, and the pin is floating.
WEAK_PULL_UP	Configure as week pull high
STRONG_PULL_UP	Configure as strong pull high
PULL_DOWN	Configure as pull low

3.3.4.1.5.IO_Wakeup_Pol_e

Configure the wake-up polarity of the pin, wake up on the rising edge or wake up on the falling edge.

POSEDGE	Configure as rising edge wakeup
NEGEDGE	Configure as falling edge wakeup

3.3.4.1.6.Fmux_Type_e

Configure pin function settings.

3.3.4.2. Data structure

3.3.4.2.1.gpioin_Hdl_t

Prototype of callback function in GPIO input mode.

3.3.4.3.APIs

3.3.4.3.1.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 it is generally called in the hal_init () function.

- Parameter: None
- Return value:

PPlus_SUCCESS	Initialisation success
Other values	Refers to <error.h>

3.3.4.3.2.void hal_gpio_wakeup_set (GPIO_Pin_e pin,IO_Wakeup_Pol_e type)

Configure GPIO wake-up mode: rising edge wake-up or falling edge wake-up

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	pin	GPIO pin
IO_Wakeup_Pol_e	type	Wakeup mode

- Return value:: None

3.3.4.3.3.void hal_gpio_pin_init(GPIO_Pin_e pin,GPIO_ioe type)

Configure GPIO mode, input or output.

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	pin	GPIO pin
GPIO_ioe	type	Configure GPIO as input or output.

- Return value:: None

3.3.4.3.4.void hal_gpio_write(GPIO_Pin_e pin, uint8_t en)

Write 1 or 0 to a particular GPIO

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin
uint8_t	en	0: write 0, other value: write 1

- Return value:: None

3.3.4.3.5.uint32_t hal_gpio_read(GPIO_Pin_e pin)

Read a particular GPIO value

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin

- Return value:

0: low level; 1: High level

3.3.4.3.6.int hal_gpio_cfg_analog_io(GPIO_Pin_e pin, BitAction_e value)

Configure IO as analog mode

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin
BitAction_e	value	enable or disable analog mode

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.4.3.7. int hal_gpio_pull_set(GPIO_Pin_e pin,IO_Pull_Type_e type);

Configure GPIO pull high/low.

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin
Pull_Type_e	type	configure IO pull hight/low

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.4.3.8. int hal_gpio_fmux_set(GPIO_Pin_e pin,Fmux_Type_e type)

Configure IO pin as function mode

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin
Fmux_Type_e	type	IO function mode

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

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

Register the GPIO input mode, which supports interrupt and wakeup callbacks, including rising edge callback and falling edge callback functions.

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin
gpioin_Hdl_t	posedgeHdl	The callback function for the rising edge can be NULL.
gpioin_Hdl_t	negedgeHdl	The callback function for the falling edge can be NULL.

- Return value:

PPlus_SUCCESS	Success

Other values	Refers to <error.h>

3.3.4.3.10. int hal_gpioin_unregister(GPIO_Pin_e pin)

Cancel the input mode of GPIO. In this mode, interrupt and wake-up callbacks are supported, including rising edge callback and falling edge callback functions. ◦

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.4.3.11. int hal_gpioin_enable(GPIO_Pin_e pin)

For a pin that has been registered as gpioin, if the pin is disabled (Refers to hal_gpioin_disable), this function can enable the gpioin function.

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.4.3.12. int hal_gpioin_disable(GPIO_Pin_e pin)

For a pin that has been registered as GPIO_IN, if the pin is enabled, the function can stop the GPIO-IN function

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin

- Return value:

PPlus_SUCCESS	Success

Other values	Refers to <error.h>

3.3.5. I2S

3.3.5.1. Data structure

3.3.5.2.APIs

3.3.6. I2C Master

This chapter introduces the I2C Master mode.

PHY62XX has two I2C controllers, one of which can be selected through API.

This chapter mainly introduces the basic usage method of I2C Master by example. The following introduces the application of the module through practical use cases, taking the acceleration sensor KX023 as an example.

3.3.6.1. Reference code

3.3.6.1.1. I2C initialisation

The initialisation process mainly includes IO configuration and I2C initialisation. Initialisation needs to configure the speed of the I2C bus: I2C_CLOCK_100K or I2C_CLOCK_400K.

```
static void* kxi2c_init(void)
{
  void* pi2c;
  hal_i2c_pin_init(I2C_0, P26, P27);
  pi2c = hal_i2c_init(I2C_0,I2C_CLOCK_400K);
  return pi2c;
}
```

3.3.6.1.2. I2C Deinit

To release related resources during the Deinit process, please refer to the following code.

```
static int kxi2c_deinit(void* pi2c)
{
  int ret;
  ret = hal_i2c_deinit(pi2c);
  hal_gpio_pin_init(P26,IE);
  hal_gpio_pin_init(P27,IE);
  return ret;
}
```

3.3.6.1.3. I2C read write

The Slave address of the slave device needs to be input during reading and writing. Refer to the following code for specific implementation. Due to the real-time requirements, the transmission process FIFO cannot be empty, so the process of writing FIFO cannot be interrupted.

```
static int kxi2c_read(void* pi2c, uint8_t reg, uint8_t* data, uint8_t size)
{
 return hal_i2c_read(pi2c, KX023_SLAVE_ADDR, reg, data, size);
}

static int kxi2c_write(void* pi2c, uint8_t reg, uint8_t val)
{
 uint8_t data[2];
 data[0] = reg;
 data[1] = val;
```

3.3.7. I2C Slave

TBD.

3.3.8. PWM

3.3.9. Definitions

3.3.9.1. PWMN_e

Define PWM channel 0~5.

3.3.9.2.PWM_CLK_DIV_e

Clock frequency division

PWM_CLK_NO_DIV	16MHz clock, None frequency division
PWM_CLK_DIV_2	2~128 frequency division
PWM_CLK_DIV_4	
PWM_CLK_DIV_8	
PWM_CLK_DIV_16	
PWM_CLK_DIV_32	
PWM_CLK_DIV_64	
PWM_CLK_DIV_128	

3.3.9.3. Data structure

3.3.9.4. APIs

3.3.9.4.1. void hal_pwm_init(PWMN_e pwmN, PWM_CLK_DIV_e pwmDiv, PWM_CNT_MODE_e pwmMode, PWM_POLARITY_e pwmPolarity)

PWM initialisation, including turning on the clock, configuring channels, configuring crossover parameters, working mode, and polarity.

- Parameter

Type	Parameter name	Description
PWMN_e	pwmN	Channel Parameter
PWM_CLK_DIV_e	pwmDiv	Frequency division parameter
PWM_CNT_MODE_e	pwmMode	Counting mode, increment or decrement.

PWM_POLARITY_e	pwmPolarity	Output polarity

- Return value: None

3.3.9.4.2. void hal_pwm_open_channel(PWMN_e pwmN,GPIO_Pin_e pwmPin)

使能PWM通道，绑定IO pin。

- Parameter

Type	Parameter name	Description
PWMN_e	pwmN	Channel Parameter
GPIO_Pin_e	pwmPin	IO pin

- Return value: None

3.3.9.4.3. void hal_pwm_close_channel(PWMN_e pwmN)

Close PWM channel

- Parameter

Type	Parameter name	Description
PWMN_e	pwmN	Channel Parameter

- Return value: None

3.3.9.4.4. void hal_pwm_destroy(PWMN_e pwmN)

Close the PWM channel and clear the channel configuration

- Parameter

Type	Parameter name	Description
PWMN_e	pwmN	Channel Parameter

- Return value:: None

3.3.9.4.5. void hal_pwm_set_count_val(PWMN_e pwmN, uint16_t cmpVal, uint16_t cntTopVal)

Configure counter data to configure the PWM duty cycle.

- Parameter

Type	Parameter name	Description
PWMN_e	pwmN	Channel Parameter
uint16	cmpVal	Counter value
uint16_t	cntTopVal	Counter total value

- Return value: None

3.3.9.4.6. void hal_pwm_start(void)

Start the PWM counter. Because the PWM operation needs to depend on the system clock, it will lock and not enter sleep during the work process.

- Parameter: None
- Return value: None

3.3.9.4.7. void hal_pwm_stop(void)

Turn off the PWM counter. Before that, you need to turn off the open PWM channel.

- Parameter: None
- Return value: None

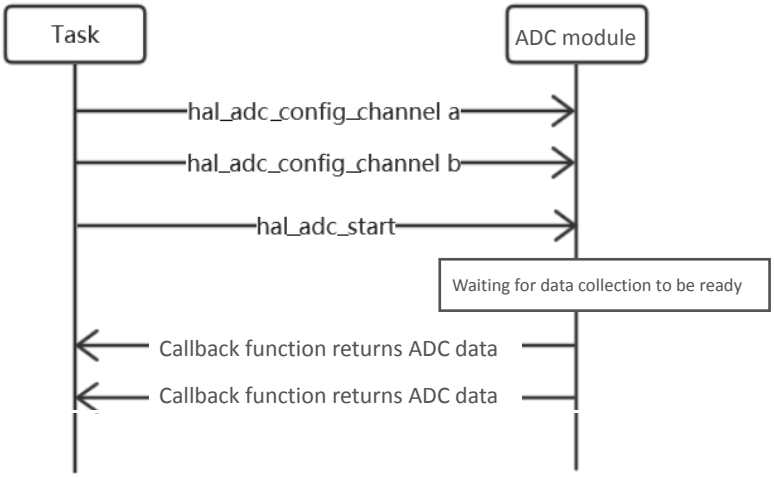
3.3.10. QDEC

TBD

3.3.11. ADC

The ADC driver provides asynchronous AD acquisition function. After the data acquisition is completed, the ADC acquisition result is returned through the callback function. The ADC driver supports both single-ended and differential modes.

If you need to use the ADC module, you generally need to initialize it in the hal_init () function of main.c, and then use it in the Application Task according to the following figure. The following figure assumes that two ADC channels are collected simultaneously.



3.3.11.1. Definition

3.3.11.1.1. adc_CH_t

ADC channel。

ADC_CH0	Not support
ADC_CH1	Not support

ADC_CH1P or ADC_CH1DIFF	In single-ended mode, it works as channel 1P and works independently. In differential mode, it works as 1DIFF channel and 1N channel in combination.
ADC_CH1N	1N channel in single-ended mode, None effect in differential mode
ADC_CH2P or ADC_CH2DIFF	In single-ended mode, it works as channel 2P and works independently. In differential mode, it works as 2DIFF channel and is used in combination with 2N channel.
ADC_CH2N	2N channel in single-ended mode, None effect in differential mode
ADC_CH3P或ADC_CH3DIFF	In single-ended mode, it works as a channel 3P and works independently. In differential mode, it works as a 3DIFF channel and is used in combination with a 3N channel.
ADC_CH3N	As a channel 3N in single-ended mode, None effect in differential mode
ADC_CH_VOICE	Voice channel, used for collecting analog microphones.

3.3.11.1.2. ADC event

ADC driving events will be thrown in the callback function of ADC driving.

HAL_ADC_EVT_DATA	ADC sampling data, if the sampling data is ready, it will call the registered ADC callback function to send the event.
HAL_ADC_EVT_FAIL	ADC sampling fail

3.3.11.2. Data structure

3.3.11.2.1. adc_Evt_t

ADC drive event data structure.

Int	type	ADC event type
adc_CH_t	ch	ADC channel
uint16_t*	data	ADC sampled data
uint8_t	size	The size of the ADC sampling data, the data unit is 16bit.

3.3.11.2.2. adc_Hdl_t

ADC callback function Type.

typedef void (*adc_Hdl_t)(adc_Evt_t* pev)

3.3.11.2.3. adc_Cfg_t

ADC configuration parameter

Bool	is_continue_mode	Whether continuous acquisition mode, if TRUE, ADC acquisition will continue to work until manually stopped

Bool	is_differential_mode	Whether it is differential mode. If it is differential mode, P and N terminals need to be configured in pairs.
Bool	is_high_resolution	Whether to use attenuation, if the value is TRUE, then use attenuation, the range is 0V ~ 1V, otherwise the range is 0V ~ 4V.
Bool	is_auto_mode	Not support at this moment

3.3.11.3.APIs

3.3.11.3.1. void hal_adc_init(void)

The ADC module is initialized, and other functions of the module need to be configured before it can be used, otherwise the None method predicts the result, or returns an error.

- Parameter: None
- Return value:: None

3.3.11.3.2. int hal_adc_config_channel (adc_CH_t channel, adc_Cfg_t cfg, adc_Hdl_t evt_handler)

Configure ADC acquisition channel

- Parameter

Type	Parameter name	Description
adc_CH_t	channel	ADC channel
adc_Cfg_t	cfg	ADC configuration message
adc_Hdl_t	evt_handler	Event response callback function.

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.11.3.3.int hal_adc_start(void)

Start sampling

- Parameter: None
- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.11.3.4.int hal_adc_stop(void)

Stop sampling

- Parameter: None
- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.11.3.5.float hal_adc_value(uint16_t* buf, uint8_t size, bool high_resol, bool diff_mode)

Calculate the ADC value, the output is a floating point number, the voltage value of the sampling point, and the arithmetic average of the input buffer

- Parameter

Type	Parameter name	Description
uint16_t*	buf	sampling raw data
uint8_t	size	size of sampled data
bool	High_resol	Whether to use attenuation (TRUE: range is 0 ~ 1V; FALSE: range is 0 ~ 4V)
bool	diff_mode	Whether the sampled data is differential data.

- Return value:

float	The voltage value at the sampling point.

3.3.12. SPI

SPI provides spi0 and spi1 for communication. When the peripheral circuit uses two sets of SPI, the software can switch between the two sets of SPI in time-sharing.

SPI transmission mode, you can choose polling mode or interrupt mode.

If the amount of data is small, the transfer is completed once, and it is convenient and simple to use the polling method directly.

If the amount of data is large, the transmission is completed multiple times, either in polling mode or interrupt mode. The polling method will transmit each frame of data in turn, and the program will not continue until the data transmission is completed. The interrupt mode is executed downward after the first frame is sent. Every time the data is transferred, the application is requested for data and the data is received at the same time. The data sending buffer and the receiving buffer are configured before transmission. The program can be executed after the first frame is transmitted. But it is not allowed to enter sleep at this time, because the SPI transmission is in progress, and the sleep is enabled after the data transmission is completed, and the SPI notifies the application of the completion of the transmission through the callback function.

When sending data in interrupt mode, you need to allocate memory for Tx, and the memory size is configured according to the maximum value of a single transmission.

The SPI enable signal can be manually configured or automatically configured by the SPI controller. This needs to be determined according to the characteristics of the peripheral.

Refer to Refers to SPI flash example for specific use.

3.3.12.1. Definition

3.3.12.1.1. SPI_INDEX_e

SPI module selection

SPI0	spi 0
SPI1	spi 1

3.3.12.1.2. SPI_TMOD_e

SPI usage scenarios.

SPI_TRXD	Transmit & Receive
SPI_TXD	Transmit Only
SPI_RXD	Receive Only
SPI_EEPROM	EEPROM Read

3.3.12.1.3.SPI_SCMOD_e

SPI working mode

SPI_MODE0	SCPOL=0,SCPH=0
SPI_MODE1	SCPOL=0,SCPH=1
SPI_MODE2	SCPOL=1,SCPH=0
SPI_MODE3	SCPOL=1,SCPH=1

3.3.12.1.4. spi_Hdl_t

SPI transmission completion callback function, which needs to be configured when transmitting in interrupt mode

3.3.12.1.5. spi_Type_t

SPI transmission completion callback function Parameter.

TRANSMIT_COMPLETED	Transmission completed

3.3.12.2. Data structure

3.3.12.2.1.spi_Cfg_t

SPI configuration Parameter

GPIO_Pin_e	sclk_pin	Configure CLK pin
GPIO_Pin_e	ssn_pin	Configure CS pin

GPIO_Pin_e	MOSI	Configure MOSI pin
GPIO_Pin_e	MISO	Configure MISO pin
uint32_t	baudrate	Configure SPI buad
SPI_TMOD_e	spi_tmod	Configure SPI mode
SPI_SCMOD_e	spi_scmod	Configure SPI operating mode
bool	int_mode	Use interrupt mode / non-interrupt mode transfer
bool	force_cs	Use manual setting / IP to pull the CS pin
spi_Hdl_t	evt_handler	When using interrupt transmission, the callback function when the transmission is completed

3.3.12.3.APIs

3.3.12.3.1. void hal_spi_init(void)

The SPI module is initialized, and other functions of the module need to be configured before it can be used, otherwise the None method predicts the result, or returns an error.

- Parameter: None
- Return value: None

3.3.12.3.2. int hal_spi_bus_init(hal_spi_t* spi_ptr,spi_Cfg_t cfg)

Configure to enable the SPI module

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle.
spi_Cfg_t	cfg	SPI configuration information

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.12.3.3. int hal_spi_bus_deinit(hal_spi_t* spi_ptr)

Disable SPI module

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.12.3.4. int hal_spi_set_int_mode(hal_spi_t* spi_ptr,bool en)

Set SPI transmission mode, interrupt / non-interrupt

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle.
bool	en	true-interrupt mode false—non-interrupted mode

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.12.3.5. int hal_spi_set_force_cs(hal_spi_t* spi_ptr,bool en)

Set SPI CS pin control mode, manual mode is recommended.

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle.
bool	en	true-manually set CS status false—IP sets CS status

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.12.3.6.int hal_spi_int_set_tx_buf(hal_spi_t* spi_ptr,uint8_t* tx_buf,uint16_t len)

When using interrupt mode to send data, you need to allocate a buffer for it.

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle

uint8_t*	tx_buf	Tx buffer pointer
uint16_t	len	Tx buffer length

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.12.3.7.int hal_spi_transmit(hal_spi_t* spi_ptr,uint8_t* tx_buf,uint8_t* rx_buf,uint16_t len)

SPI starts data transfer.

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle
uint8_t*	tx_buf	Tx buffer pointer
uint8_t*	rx_buf	Rx buffer pointer
uint16_t	len	buffer length. Send buffer and receive buffer length need to be equal

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.12.3.8.bool hal_spi_get_transmit_bus_state(hal_spi_t* spi_ptr)

Query the SPI status to see if any data is being transferred in interrupt mode.

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle

- Return value:

true	SPI idle
false	SPI busy, current data transfer not success

3.3.12.3.9.int hal_spi_module_select(hal_spi_t* spi_ptr)

SPI module selection. Only two sets of SPI will be called when they are used. The two sets of SPI peripheral application circuits are independent, and the software is time-multiplexed.

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle

- Return value:

PPlus_SUCCESS	Success ◦
Other values	Refers to <error.h>

3.3.12.3.10.void hal_spi_send_byte(hal_spi_t* spi_ptr,uint8_t data)

SPI sends a data, the interface is reserved for forward compatibility.

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle
uint8_t	data	Data to be transmit

- Return value:: None

3.3.12.3.11.void hal_spi_TxComplete(hal_spi_t* spi_ptr)

Wait for the SPI transfer to complete, and the interface is reserved for forward compatibility.

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle

- Return value:: None

3.3.13. UART

UART provides a set of APIs for UART synchronous or asynchronous mode control

For Rx, this module only provides the asynchronous mode interface. When Rx data is ready, the **UART_EVT_TYPE_RX_DATA** event or **UART_EVT_TYPE_RX_DATA_TO** will be thrown through the callback function registered during initialization (timeout: when the last data is less than FIFO Threshold)

For Tx, this module provides both synchronous and asynchronous interfaces. If the configuration is cfg at the time of initialization. **Use_tx_buf** is FALSE, Tx synchronization, you can call the function **hal_uart_send_buff (uint8_t * buff, uint16_t len)** to send data in a synchronous manner, the function is blocked.

If cfg. **Use_tx_buf** is TRUE at the time of initialization, and cfg. Use_fifo is TRUE, then Tx is asynchronous. After the initialization is complete, you need to call **hal_uart_set_tx_buf (uint8_t * buf, uint16_t size)** to configure the Tx cache before you can use **hal_uart_send_buff (uint8_t * buff, uint16_t len)** Send data. At this time,

the function is non-blocking. After sending, the **UART_EVT_TYPE_TX_COMPLETED** event will be thrown through the callback function.

3.3.13.1. Definition

3.3.13.1.1. uart_Evt_Type_t

For asynchronous transmission, the callback function will throw one of the following event types:

UART_EVT_TYPE_RX_DATA	UART Rx data ready
UART_EVT_TYPE_RX_DATA_TO	UART Rx data is ready and the reception timed out.
UART_EVT_TYPE_TX_COMPLETED	UART Tx transmit completed

3.3.13.2. Data structure

3.3.13.2.1.uart_Cfg_t

UART setting parameter

GPIO_Pin_e	tx_pin	Configure Tx pin
GPIO_Pin_e	rx_pin	Configure Rx pin
uint32_t	Baudrate	Set baud rate
bool	use_fifo	Use FIFO or not, suggest to use
bool	use_tx_buf	use Tx buffer or not, if use, Tx will be async mode
uart_Hdl_t	evt_handler	Set the callback function.
GPIO_Pin_e	ts_pin	Not yet support
GPIO_Pin_e	cts_pin	Not yet support
bool	hw_fwctrl	Not yet support
bool	parity	Not yet support

3.3.13.3. APIs

3.3.13.3.1. int hal_uart_init (uart_Cfg_t cfg)

It is used to initialize the UART module. Other functions of the module need to be configured before they can be used. Otherwise, the None method predicts the result or returns an error.

- Parameter

Type	Parameter name	Description
uart_Cfg_t	cfg	Serial port configuration information

- Return value:

PPlus_SUCCESS	Success

Other values	Refers to <error.h>

3.3.13.3.2.int hal_uart_set_tx_buf(uint8_t* buf, uint16_t size)

Set the Tx cache. If cfg. Use_tx_buf is TRUE during initialization, this function needs to be executed.

- Parameter

Type	Parameter name	Description
uint8_t*	buf	Need to configure the cache pointer.
uint16_t	size	Buffer size

- Return value:

PPlus_SUCCESS	Buffer configuration success
Other values	Refers to <error.h>

3.3.13.3.3. int hal_uart_get_tx_ready(void)

Used to query whether an asynchronous Tx transmission is complete, or to query whether Tx is idle.

- Parameter: None
- Return value:

PPlus_SUCCESS	Tx transmission completed
PPlus_ERR_BUSY	Tx busy

3.3.13.3.4. hal_uart_send_buff(uint8_t *buff,uint16_t len)

Tx sends data, if it is in synchronous mode, the function is blocked, the transmission is completed, or the transmission returns overtime, if it is in asynchronous mode, the function is non-blocking, and the **UART_EVT_TYPE_TX_COMPLETED** event is thrown through the callback function after the data transmission is completed.

- Parameter

Type	Parameter name	Description
uint8_t*	buf	data to be sent
uint16_t	len	Size of data

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.3.14. Key Scan

TBD.

3.3.15. Flash

3.3.15.1. Definition

3.3.15.2. Data structure

3.3.15.3. APIs

3.3.15.3.1. uint8_t flash_write_ucds(uint32_t addr,uint32_t value)

Used for users to write flash, the user address range is 0x0000 ~ 0x3ffc, and an error will be reported if it exceeds this range.

- Parameter

Type	Parameter name	Description
uint32_t	addr	Flash address （0x0000~0x3ffc）
uint32_t	value	Value write into flash

- Return value:

PPlus_SUCCESS	Write in success
Other values	Refers to <error.h>

3.3.15.3.2. uint32_t flash_read_ucds(uint32_t addr)

It is used for users to read flash, and the user address range is 0x0000 ~ 0x3ffc.

- Parameter

Type	Parameter name	Description
uint32_t	addr	Flash address （0x0000~0x3ffc）

- Return value:

0xffffffff	Flash value is 0xffffffff or incorrect address
Other values	Flash address's value

3.3.15.3.3. void flash_erase_ucds_all(void)

Used for user to erase all flash, user address range is 0x0000 ~ 0x3ffc

- Parameter: None
- Return value:: None

3.3.15.3.4. uint8_t flash_erase_ucds(uint32_t addr)

Sector used for user erasing specified address, user address range is 0x0000 ~ 0x3ffc

- Parameter

Type	Parameter name	Description
uint32_t	addr	Flash address （0x0000~0x3ffc）

- Return value:

PPlus_SUCCESS	Write in success
Other values	Refers to <error.h>

3.4. Power-save management

PRBMD0x provides two power-save modes: sleep mode and Power Off mode

For the sleep mode, the SDK centrally manages sleep and wakeup, and enables or disables sleep through precompiled macros. If the application code requires a process to disable sleep, you can configure it in the Refers to section.

For Power Off mode, SDK provides API for entering Power Off mode, and can be configured by Parameter.

3.4.1. Enable and disable Sleep mode

You can configure it in the MDK project → Option dialog → C / C ++ page → Preprocessor Symbol → Define：

CFG_SLEEP_MODE=PWR_MODE_SLEEP	Enable sleep mode
CFG_SLEEP_MODE=PWR_MODE_NO_SLEEP	Disable sleep mode

3.4.1.1. Application control sleep mode

The application code can obtain the control of dormancy by registering the PwrMgr module. It should be noted that the following modules of MOD_VOC are reserved for actual hardware modules, MOD_USR0 is reserved for rfPhy, and the modules that can be used by applications are MOD_USR1 and above modules, specific For module definition, please refer to Module ID:

The specific code Refers to as follows：

```
//register application module
hal_pwrmgr_register(MOD_USR1, NULL, rf_wakeup_handler);

//Application disable sleep mode
hal_pwrmgr_lock(MOD_USR1);
// Application enable sleep mode
hal_pwrmgr_lock (MOD_USR1);
```

3.4.2. Power Off mode

If you need to enter the Power Off mode, you can call the function **hal_pwrmgr_poweroff (pwroff_cfg_t cfg)** to enter sleep, Parametercfg is used to configure the wake-up pin.

In Power Off mode, the system is in the Power Off state, the BLE broadcast or connection before Power off will be disconnected, and the system is equivalent to a cold start after waking up.

3.4.3. Related APIs

3.4.3.1. APIs

3.4.3.1.1. int hal_pwrmgr_init(void)

Initialise module.

- Parameter: None
- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.4.3.1.2. bool hal_pwrmgr_is_lock(MODULE_e mod)

Query whether a module (hardware or application module) prohibits sleep

- Parameter

Type	Parameter name	Description
MODULE_e*	mod	module ID

- Return value:

TRUE	module prohibit sleep mode
FALSE	module allows sleep mode

3.4.3.1.3. int hal_pwrmgr_lock(MODULE_e mod)

Module prohibit sleep mode

- Parameter

Type	Parameter name	Description
MODULE_e*	mod	Module ID

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.4.3.1.4. int hal_pwrmgr_unlock(MODULE_e mod)

Module allows sleep mode

- Parameter

Type	Parameter name	Description
MODULE_e*	mod	Module ID

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.4.3.1.5. int hal_pwrmgr_register(MODULE_e mod, pwrmgr_Hdl_t sleepHandle, pwrmgr_Hdl_t wakeupHandle)

Register with the sleep management system, including registering the module, registering the sleep callback function, and registering the wake-up callback function, where the module is a mandatory option. The sleep and wake-up callback functions are optional Parameters. If you do not need to set them, set the Parameter to NULL.

- Parameter

Type	Parameter name	Description
MODULE_e*	mod	Module ID
pwrmgr_Hdl_t	sleepHandle	Callback function, which will be called before going to sleep.
pwrmgr_Hdl_t	wakeupHandle	Callback function, which will be called after waking up from sleep.

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.4.3.1.6. int hal_pwrmgr_unregister(MODULE_e mod)

The module is logged off from the hibernation management system.

- Parameter

Type	Parameter name	Description
MODULE_e*	mod	Module ID

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.4.3.1.7. int hal_pwrmgr_RAM_retention(uint32_t sram)

Configure the RAM that can be kept during sleep, Bit 0 ~ 4 are valid, corresponding to RAM0 ~ RAM4, if the corresponding Bit is 1, the data of this RAM can be maintained during sleep, if it is 0, then during sleep, the data will be Lost.

- Parameter

Type	Parameter name	Description
uint32_t	sram	Configure RAM that can be kept during sleep, Bit 0 ~ 4 is valid, corresponding to RAM0 ~ RAM4 respectively.

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.5. Libraries

3.5.1. File system

Lightweight file system based on internal flash, the file system provides a set of synchronization API, supports 16bit file ID, file search, read, write, delete, and file system query, garbage file recycling.

3.5.1.1. Definition

3.5.1.1.1. FS_SETTING

FS each record length, this macro definition can be set in the project configuration, if set the default length of each record is 16 bytes.

FS_ITEM_LEN_16BYTE	Each record is 16 bytes long
FS_ITEM_LEN_32BYTE	Each record is 32 bytes long
FS_ITEM_LEN_64BYTE	Each record is 64 bytes long

3.5.1.2. Data structure

None

3.5.1.3. APIs

3.5.1.3.1.int hal_fs_init(uint32_t fs_start_address,uint8_t sector_num)

Initialize the file system with the configuration parameter. This function needs to be set when the system is initialized. For details, refers to Refers to routine.

- Parameter

Type	Parameter name	Description
uint32_t	fs_start_address	FS start address. It needs to be 4096 bytes aligned, and the space cannot conflict with other uses.
uint8_t	sector_num	The number of FS sectors, the effective value is 3 ~ 78. Examples: Allocate FS to 4 sectors, starting address is 0x11005000 hal_fs_init (0x11005000,4)

- Return value:

PPlus_SUCCESS	Initialisation success
Others	Refers to <error.h>

3.5.1.3.2. int hal_fs_item_read(uint16_t id,uint8_t* buf,uint16_t buf_len,uint16_t* len)

Read FS document

- Parameter

Type	Parameter name	Description
uint16_t	id	Read document ID
uint8_t*	buf	Incoming buffer start address.
buf_len	buf_len	incoming buffer length
uint16_t*	len	Actual length of document

- Return value:

PPlus_SUCCESS	Initialisation success
Others	Refers to <error.h>

3.5.1.3.3. int hal_fs_item_write(uint16_t id,uint8_t* buf,uint16_t len)

Write to FS document

- Parameter

Type	Parameter name	Description
uint16_t	id	File ID to be written
uint8_t*	buf	Incoming buffer start address.
uint16_t	len	Incoming buffer length

- Return value:

PPlus_SUCCESS	Initialisation success
Other values	Refers to <error.h>

3.5.1.3.4. uint32_t hal_fs_get_free_size(void)

The size of the space that FS can use to store file data, in bytes.

- Parameter: None
- Return value: FS available storage file space, in bytes.

3.5.1.3.5. int hal_fs_get_garbage_size(uint32_t* garbage_file_num)

FS deleted file data area size in bytes.

- Parameter

Type	Parameter name	Description
uint32_t*	garbage_file_num	number of deleted files

- Return value:

The space occupied by FS deleted files, in bytes.

3.5.1.3.6. int hal_fs_item_del (uint16_t id)

Delete file

- Parameter

Type	Parameter name	Description
uint16_t	id	Delete file ID

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.5.1.3.7. int hal_fs_garbage_collect(void)

Garbage collection releases the space occupied by deleted files in FS.

This function will traverse the entire FS, and will also erase multiple sectors, which is relatively time-consuming.

It is recommended to execute when the CPU is idle and there are many garbages. The execution time is related to the main frequency and FS size.

- Parameter: None
- Return value:

PPlus_SUCCESS	Initialisation success
Other values	Refers to <error.h>

3.5.1.3.8. int hal_fs_format (uint32_t fs_start_address,uint8_t sector_num)

Format FS, all files will be erased, use caution.

If it must be called, it is recommended to call it when the CPU is idle. The execution time is related to the main frequency and FS size.

- Parameter

Type	Parameter name	Description
uint32_t	fs_start_address	FS start address. It needs to be 4096 bytes aligned, and the space cannot conflict with other uses.
uint8_t	sector_num	The number of FS sectors, the effective value is 3 ~ 78. Examples: Allocate FS to 4 sectors, starting address is 0x11005000 hal_fs_init (0x11005000,4)

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.5.1.3.9. bool hal_fs_initialized(void)

Query the FS initialization status.

- Parameter: None
- Return value:

true	It is initialized and ready for use.
false	Not initialized and cannot be used.

3.5.2. Date and Time

To implement the perpetual calendar function, datetime_t is a time accurate to the second. The library is a service running in the background, driven by osal_timer, which realizes automatic time update, and provides API for time acquisition and time setting.

3.5.2.1. Definition

3.5.2.1.1. USE_SYS_TICK

TRUE	Use **osal_sys_tick** as the clock reference count. When using RC as the 32K clock source, **osal_sys_tick** as the calibrated tick value has higher accuracy
FALSE	Use the RTC counter as the reference count for the clock.

3.5.2.2. Data structure

3.5.2.2.1. datetime_t

Data format of date and time

uint8_t	seconds	second
uint8_t	minutes	minute
uint8_t	hour	Hour（0~23）
uint8_t	day	Day of month
uint8_t	month	Month（1~12）
uint16_t	year	year

3.5.2.3. APIs

3.5.2.3.1. void app_datetime_init(void)

Calendar application initialization, usually called when the application task is initialized.

- Parameter: None
- Return value: None

3.5.2.3.2.void app_datetime_sync_handler(void)

The calendar application synchronization function requires synchronization about one minute (or less than one minute). No precise calls are required.

- Parameter: None
- Return value: None

3.5.2.3.3.int app_datetime_set(datetime_t dtm)

Set the system time according to DTM

- Parameter

Type	Parameter name	Description
datetime_t	dtm	Time value to be set, accurate to second

- Return value:

PPlus_SUCCESS	Success
Other values	Refers to <error.h>

3.5.2.3.4. int app_datetime(datetime_t* pdtm)

Obtain the current system time

- Parameter

Type	Parameter name	Description
datetime_t*	pdtm	Output Parameter. If Success is returned, pdtm will be written to the latest system time.

- Return value:

PPlus_SUCCESS	Success。
Other values	Refers to <error.h>

3.5.2.3.5. int app_datetime_diff(const datetime_t* pdtm_base, const datetime_t* pdtm_cmp)

Compare the two times and get the difference in seconds.

- Parameter

Type	Parameter name	Description
const datetime_t*	pdtm_base	The time being compared.
const datetime_t*	pdtm_cmp)	Price comparison time, the formula is pdtm_cmp – pdtm_base

- Return value:

int	Time difference, if the compared time is newer, then Return value: is negative

3.5.3. Dot matrix font library

There is a Font library provides multi-language, configurable font library solution.

The font program includes two parts:

- Font library + API
  - Font driver library and API functions, provide font registration, UTF-8 parsing and font bitmap interface
  - Font library Bitmap interface supports variable width text
- Multi-language font library
  - Generate multi-language fonts according to customer needs. In the Unicode basic multilingual plane (BMP 0x0000 ~ 0xffff), you can configure any language combination according to your needs, and you can configure the text resolution and scanning method according to customer needs.
  - For Chinese, it can support the 2009 version of the 3500 common word generation scheme.
  - Font files provide binary format files (.bin) and upgrade files (.res)

3.5.3.1.APIs

3.5.3.1.1. void* ui_font_load(uint32_t flash_addr)

Load the font, load the font that has been burned in the internal flash, and after loading, you will get a font handle. You can use this handle to call the function ui_font_unicode () to get the bitmap of Unicode characters.

- Parameter

Type	Parameter name	Description
uint32_t	flash_addr	Flash address for font data storage

- Return value:

Returns the handle of the void * Type font. If the value is NULL, it indicates that the load failed.

3.5.3.1.2. int utf8_to_unicode(const char* utf8, uint16_t *unicode)

Convert UTF-8 strings to Unicode characters.

- Parameter

Type	Parameter name	Description
const char*	utf8	Enter Parameter, and enter the UTF-8 character string, which ends with "\ 0".
uint16_t *	unicode	Output Parameter, a Unicode character obtained by converting UTF-8.

- Return value: Return intType

0	The string is parsed, and the output Parameter is None effect data.
1~n	The conversion obtains a Unicode character, and the corresponding UTF-8 string parsing consumes n bytes.

3.5.3.1.3. int ui_font_unicode(void* font, uint16_t unicode,uint8_t *bitmap)

Get bitmap data based on Unicode characters.

- Parameter

Type	Parameter name	Description
void*	font	Font handle.
uint16_t	unicode	Unicode characters, need to be converted to bitmap
uint8_t *	bitmap	Unicode character bitmap. Among them, bytes 0 ~ 3 are resolution information, and bytes 4 and later data are bitmap data. Byte 0 ~ 3 specific content: Byte0: bitmap data width Byte1: bitmap data height Byte2: actual effective data width of bitmap Byte4: reserved.

- Return value: Return intType ◦

0	Operation success
Other values	Error code

3.5.3.1.4. const char* ui_font_version(void)

Returns the font library lib version information and font file version information.

- Parameter: None ◦
- Return value: const char* Type string

4. BLE

4.1. GAP

Common access profile（GAP）：

The GAP layer in the Ble protocol stack is responsible for processing device access modes, including device discovery, connection establishment, termination, initial security management, and device configuration. Therefore, many functions in the ble protocol stack are prefixed with GAP. These functions will Responsible for the above content.

The GAP layer always serves as one of the following four clock roles:

- Broadcaster ——Unable to connect to devices that have been broadcasting；
- Observer ——Broadcast devices can be scanned, but devices that cannot initiate connection establishment；
- Peripheral ——The broadcast equipment that can be connected can be a slave in a single link layer connection.
- Central ——You can scan broadcast devices and initiate connections, acting as a host in a single link layer or multiple link layers.

In a typical Bluetooth low energy system, the slave device broadcasts specific data to let the host know that it is a connectable device. The broadcast content includes the device address and some additional data, such as device name and service. After receiving the broadcast data, the host will send a scan request ScanRequest to the slave, and then the slave will respond to the specific data to the host, which is called ScanResponse. After the host receives the scan response, it knows that it is an external device that can establish a connection. This is the whole process of device discovery. At this time, the master can initiate a request to establish a connection to the slave. The connection request includes the following parameters.

- Connection interval: The FM mechanism is used in the connection of two BLE devices. The two devices use a specific channel to send and receive data, and then use the new channel after a period of time. (Link layer handles channel switching). Two devices send and receive data after channel switching is called a connection event. Even if no application data is sent or received, the two devices will still maintain the connection by exchanging link layer data. The time interval between connection events, the connection interval is in units of 1.25ms, and the value of the connection interval is 6 (7.5ms) ~ 3200 (4s)。
- Peripheral delay: This Parameter setting can make the slave to skip several connection events, which gives the slave more flexibility. If it has no data to send, it can choose to skip the connection time and continue to sleep to save power.
- Over time management: This is the maximum allowable interval between two Success connection events. If this time is exceeded (the unit of this value is 10ms) and there is no Success connection event, the device is considered to have lost the connection and returns to the unconnected state. The range is 100 (100ms) ~ 3200 (32s). In addition, the timeout value must be greater than the effective connection interval [effective connection interval = connection interval * (1 + slave delay)].
- Safety management: Only in the authenticated connection, the specific data can be read and written. Once the connection is established, the two devices are paired. When the pairing is completed, the key to the encrypted connection is formed. In a typical application, peripheral requests are concentrated The device provides the key to complete the pairing work. The key is a fixed value, such as 000000. It can also randomly generate a data to provide to the user. After the host device sends the correct key, the two devices exchange the security key and encrypt the authentication link. In many cases, the same peripheral and host will be connected and disconnected from time to time. There is a feature in ble's security mechanism that allows long-term security key information to be established between two devices. This feature is called binding. Allows two devices to quickly complete encrypted authentication when connecting, without the need to perform the entire pairing process each time they connect.

4.2. GATT

GATT(Generic Attribute Profile)：The general attribute configuration file is built on the basis of the attribute protocol (ATT), and provides some common operations and frameworks for the attribute protocol to transmit and store data resumes.

4.2.1. How to implement custom service

This section describes how to implement custom services through the SDK.

We introduce how to implement a service through the BLE-Uart routine, which contains three types of feature values: write, Notify, and Indicate.

4.2.1.1. Create an attribute table

The attribute table is a static array of gattAttribute_tType, including:

- Main service
- Eigenvalue 1 attribute (GATT_PROP_WRITE_NO_RSP| GATT_PROP_WRITE)。
- Characteristic value 1
- Eigenvalue 2 attribute(GATT_PROP_NOTIFY| GATT_PROP_INDICATE)。
- Characteristic value 2 client feature configuration descriptor (Client Characteristic Configuration Descriptor)。
- Characteristic value 2 The value of Characteristic value 1.

Please refer to Referers to SDK routine code for details.

```
static gattAttribute_t bleuart_ProfileAttrTbl[] =
{
  // Main service
  {
    { ATT_BT_UUID_SIZE, primaryServiceUUID }, /* type */
    GATT_PERMIT_READ,                   /* permissions */
    0,                                  /* handle */
    (uint8 *)&bleuart_Service           /* pValue */
  },
    // Characteristic value 1: attribute (write, write -None needs to reply)
    {
      { ATT_BT_UUID_SIZE, characterUUID },
      GATT_PERMIT_READ,
      0,
      &bleuart_RxCharProps
    },
      // Characteristic value 1: value
      {
        { ATT_UUID_SIZE, bleuart_RxCharUUID },
        GATT_PERMIT_WRITE,
        0,
        &bleuart_RxCharValue[0]
      },
    // Characteristic value 2: attribute （Notify）
    {
      { ATT_BT_UUID_SIZE, characterUUID },
      GATT_PERMIT_READ,
      0,
      &bleuart_TxCharProps
    },
      // Characteristic value 2: value
      {
        { ATT_UUID_SIZE, bleuart_TxCharUUID },
        0,
        0,
        (uint8 *)&bleuart_TxCharValue
      },
      // Characteristic value 2: Client characteristic configuration descriptor(Client Characteristic
Configuration Descriptor)
      {
        { ATT_BT_UUID_SIZE, clientCharCfgUUID },
        GATT_PERMIT_READ|GATT_PERMIT_WRITE,
        0,
        (uint8*)&bleuart_TxCCCD     },
};
```

4.2.1.2. Functions to be implemented

After completing the configuration of the attribute table, we need to add the service and register the read-write callback function. The following is the description of this part of the function.

4.2.1.2.1. bStatus_t bleuart_AddService(bleuart_ProfileChangeCB_t cb)

Used to add this service, mainly to achieve the following functions:

- Register the link status callback to respond to changes in the connection status.
- Registration service, by passing the attribute table and GATT service callback function to the protocol stack, the registration of the service is realized.
- Save the callback function pointer of the application layer for the application layer.

4.2.1.2.2. static bStatus_t bleuart_WriteAttrCB(uint16 connHandle, gattAttribute_t *pAttr, uint8 *pValue, uint8 len, uint16 offset)

The callback function responds to the Write (Write or Write no response) operation on the Central side, which contains two parts, the write operation on the feature value 1; the write operation on the client's feature value configuration descriptor for the feature value 2.

For specific implementation, please refer to the example code.

4.2.1.2.3. static void bleuart_HandleConnStatusCB (uint16 connHandle, uint8 changeType)

Callback function, respond to link status changes.

4.2.1.2.4. bStatus_t bleuart_Notify(uint16 connHandle, attHandleValueNoti_t *pNoti, uint8 taskId)

Function, used to send Notify to Central.

4.2.1.3. Application layer calls service interface

The application layer calls the bleuart_AddService () function in the application Task initialization function, and is called in the function bleuart_Init () in the BLE-Uart routine.

4.3. OTA

OTA is Over the Air, which refers to the function of upgrading firmware through the BLENone line. The OTA of PHY62XX provides a complete and reliable firmware over-the-air upgrade solution. The upgrade includes application firmware upgrade, resource file upgrade, OTA bootloader upgrade.

- Apply firmware upgrade

  Upgrade application firmware, support non-encrypted upgrade and encrypted upgrade.

- Resource file upgrade

  To upgrade resource files, the resource file storage area is a non-program area and a system protection area. For the NVM area, users need to protect themselves. The OTA upgrade process will not protect this area.

- OTA Bootloader upgrade

  Upgrading the OTA boot program can be regarded as a special application firmware upgrade. Generally, this part of the area does not need to be upgraded. If you need to upgrade this area, please refer to the routine <OTA_upgrade_2ndboot>. After the OTA bootloader is upgraded, the application firmware will also be invalidated. Therefore, after the OTA bootloader is upgraded, the application firmware needs to be upgraded.

4.3.1. OTA mode

The application firmware upgrade process needs to be performed in OTA mode.

There are three ways to enter OTA mode:

- The application firmware is not detected. If the application firmware is damaged or only the OTA bootloader firmware is programmed, the device enters the OTA mode by default after reset or power-on.
- In the application mode, the system can enter the OTA mode through the instruction of the OTA App Service. The application firmware supporting the OTA requires the OTA App Service to be loaded. Through the Service, the host sends a control command to the characteristic value under the service to make the application firmware jump to OTA mode
- The third mode needs to be customized according to user needs, usually by identifying the state of a specific IO to enter the OTA mode.

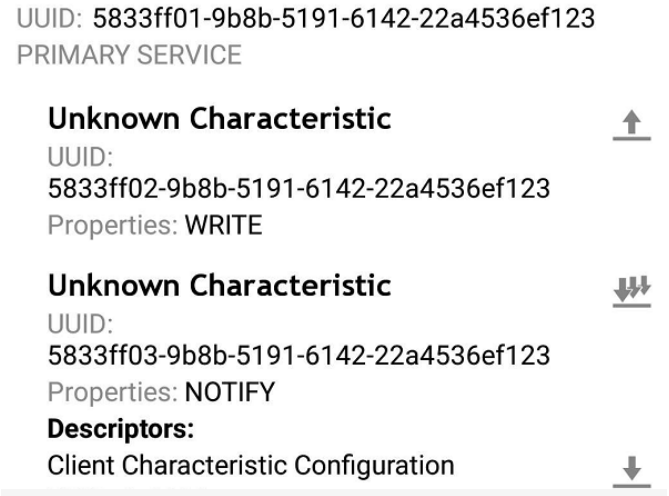
4.3.2. OTA Resource mode

The resource file upgrade process needs to be performed in the OTA Resource mode.

The application mode can enter the OTA Resource mode through the instructions of the OTA App Service.

4.3.3. OTA Service

The application firmware needs to load the OTA Service as shown in the figure below. Through this service, the host can control the device to jump to the OTA mode through commands.



OTA App Service source code locates at: SDK\components\profiles\ota_app

4.3.4. OTA Bootloader

OTA bootloader is a second-level boot program, through which the application firmware loading, encrypted application firmware loading, encrypted and non-encrypted OTA functions, and OTA resource functions can be realized. The firmware will run in the high-end address area of the RAM in the OTA mode. After the application firmware is booted, the control of the RAM will be released, and all the RAM will be managed by the application.

The OTA bootloader is divided into the following modes for different application scenarios (for 512KB Flash and above). For different OTA modes, the application firmware does not need to make any adjustments.

4.3.4.1. OTA Dual Bank Has FCT

Two symmetric 128K spaces are opened on the internal flash for application firmware, and 120K spaces are opened separately for FCT firmware (functional test firmware). The Flash address mapping in this mode is as follows:

	512K version (Dual bank)(Has FCT)	
Reserved By PhyPlus	00000~09fff	40k
OTA bootloader	0a000~11fff	32k
FCT App	12000~2ffff	120k
App Bank0	30000~4ffff	128k
App Bank1	50000~6ffff	128k
NVM	70000~7ffff	64k

4.3.4.2. OTA Dual Bank No FCT

Open up two symmetrical 128K spaces on the internal flash for application firmware, but FCT firmware is not provided separately. The Flash address mapping in this mode is as follows:

	512K version (Dual bank) (No FCT)	
Reserved By PhyPlus	00000~09fff	40k
OTA bootloader	0a000~11fff	32k
App Bank0	12000~31fff	128k
App Bank1	32000~51fff	128k
NVM	52000~7ffff	184k

4.3.4.3. OTA Single Bank Has FCT

This mode only opens up a 128K flash space for application firmware, and also provides 120K space for functional test firmware. The flash space mapping of this mode is as follows:

	512K version (Dual bank)(Has FCT)	
Reserved By PhyPlus	00000~09fff	40k
OTA bootloader	0a000~11fff	32k
FCT App	12000~2ffff	120k
App Bank0	30000~4ffff	128k
NVM	50000~7ffff	192k

4.3.4.4. OTA Single Bank No FCT

This mode only opens up a 128K Flash space for application firmware, and does not provide space for functional test firmware. The Flash space mapping of this mode is as follows:

	512K version (Dual bank) (No FCT)	
Reserved By PhyPlus	00000~09fff	40k
OTA bootloader	0a000~11fff	32k
App Bank0	12000~31fff	128k
NVM	32000~7ffff	312k

4.3.5. Encrypted OTA

OTA supports encrypted transmission and encrypted storage. For OTA in encrypted mode, application firmware needs to be encrypted through PC tools, and the key is embedded inside the chip. From the application point of view, the upgrade process is no different from non-encrypted OTA. If they do not match, the upgrade process will report an error message. ◦

4.3.6. How to implement OTA

If the chip is programmed with OTA Bootloader, it has the ability to perform OTA. For the application firmware, the OTA App Service needs to be loaded so that the application firmware can interact with the OTA Bootloader.

4.3.6.1. Application firmware and OTA

The application firmware needs to load the OTA App Service. Generally speaking, we will load the service during the initialization of the application task.

The specific code snippet is as follows：

```
void otaDemo_Init( uint8 task_id )
{
  otaDemo_TaskID = task_id;

  // Part of the code is omitted here


  // Initialize GATT attributes
  GGS_AddService( GATT_ALL_SERVICES );         // GAP
  GATTServApp_AddService( GATT_ALL_SERVICES ); // GATT attributes
```

4.3.7. Programming application firmware and OTA bootloader

The application firmware and OTA bootloader can be programmed through the PhyPlusKit tool or the Phyplus programmer hardware. For specific use, please refer to the Refers to PhyPlusKit user guide and programmer documentation.

4.3.8. OTA summary

In general, three steps are required to achieve OTA:

- Compile OTA Bootloader to get ota.hex, the routine provides four modes of hex file (For 512K flash version of hardware).
- Apply firmware to add OTA App Service: ota_app_AddService ();
- Use PhyPlusKit or PlyPlus programmer to program the chip.

After completing the above steps, you can update the application firmware or resource files (such as font files) through the Android or iOS version of PHYApp.

5. Sample code

Provide the application routines shown in the following table in the SDK:

○1：The routine can be run on the development board PHY6200_32_V1.4 and above;

○2：The routine can be run on the development board PHY6200_48_Bracelet_V1.0 and above；

○3：The routine can be run on the development board Phyplus_BLE_Mesh_V1.0 and above。

ble_peripheral		
alternate_iBeacon	alternate iBeacon example	○1○2○3
ancs	Apple Message Center (ANCS) routine	○1○2○3
bleI2C_RawPass	I2C transparent transmission routine	○1○2○3
bleSmartPeripheral	General BLE Peripheral examples	○1○2○3
bleUart-RawPass	UART tunnelling example	○1○2○3
eddystone	eddystone example	○1○2○3
HIDKeyboard	HID example	○1○2○3
hrs	Heart rate profile example	○1○2○3
iBeacon	iBeacon example	○1○2○3
otaDemo	Basic OTA example	○1○2○3
pwmLight	Routine for adjusting the brightness of LED by controlling PWM through BLE command	○2○3
RawAdv	Simple broadcast routine for applications like None line tire pressure monitoring	○1○2○3
Sensor_Broadcast		○1○2○3
wrist	Comprehensive example for applications such as sports bracelets	○2
wrist_aptm	Comprehensive example, for applications such as sports bracelets, based on the integrated example, a real-time clock is realized through AP Timer + OSAL Timer	○2
XIPDemo	Some real-time codes require lower code to run directly in the internal flash routine	○2○3
OTA		
OTA_internal_flash	OTA bootloader	○1○2○3
OTA_upgrade_2ndboot	Special application for upgrading the OTA bootloader itself	○1○2○3
peripheral		
adc	ADC Driver example	○1○2○3
ap_timer	AP Timer driver application example	○1○2○3

fs	File system	o1o2o3
gpio	GPIO sample code	o1o2o3
kscan	4x4 keypad example	o2
lcd_ST7789VW	240x240 TFT display example	o1
pwm	PWM sample example	o1o2o3
qdec	QDEC sample example	o1o2o3
spiflash	External SPI sample example	o2
voice	Audio sampling example	o2
voice_sbc	SBC format sampling example	o2
watchdog	Watchdog example	o1o2o3