PHY62XX

SDK Application Note

Version 0.2

Author: Eagle Lao

Security: Public

Date: 2021.3

PhyPlus

Revision History

Revision	Author	Participant	Date	Description
V0.1	Lao Yufeng		03/30/2018	Draft file
V0.2	Lao Yufeng		01/24/2019	Update of files based on PHY62XX_SDK_1.1.8

Table of Contents

Figures and Tables

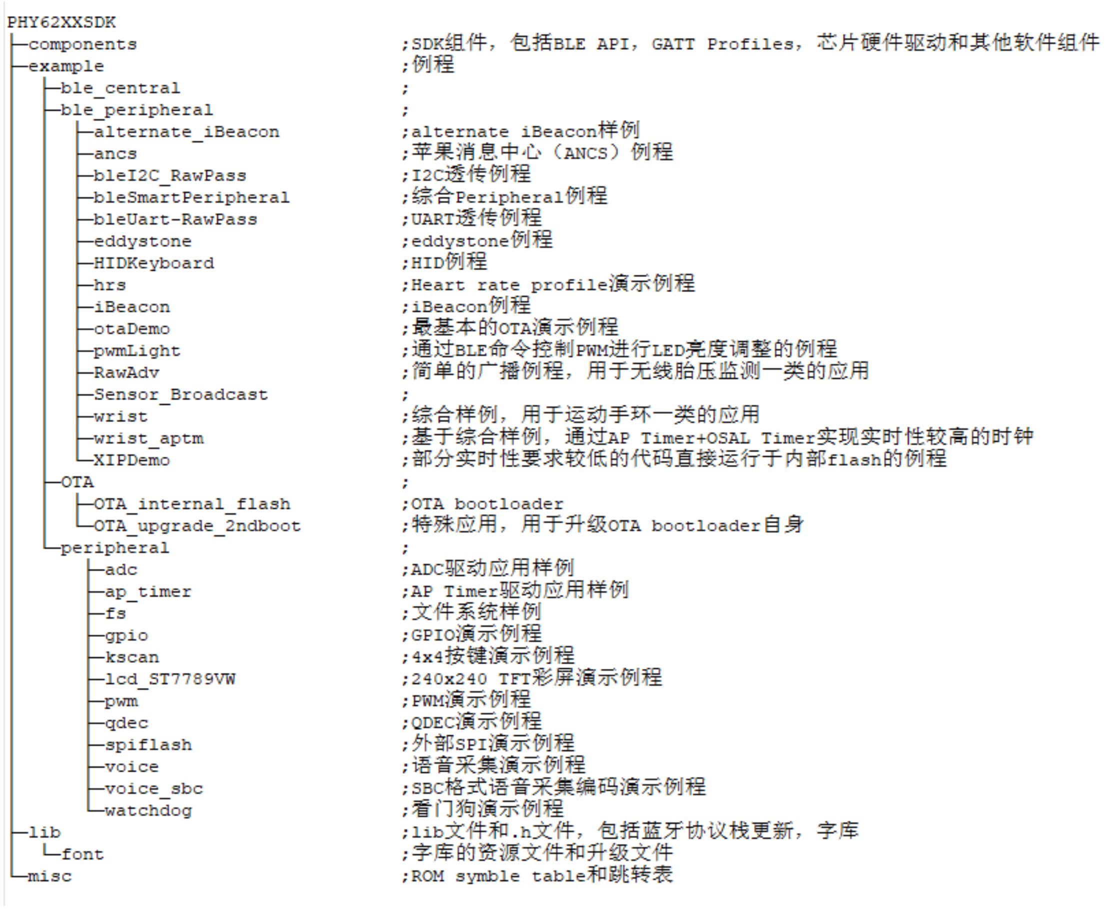
1 Introduction

This file is used to explain how to use the PHY62XX SDK to develop applications. It can help you understand and understand the components provided by the SDK, how to use the samples, and how to start the firmware development of BLE products from the provided samples.
Samples provided by the SDK are only for design reference. For product firmware development, please use the firmware you designed!

2 Quick start

The SDK provides a set of sample programs, which can be directly run on the development board. You can use these sample programs as a reference and start your own application development. By running these pre-compiled samples and conducting interactive experiments with the smartphone through the BLE test program, you can quickly understand the specific functions provided by the development board and samples.

2.1 SDK directory structure



```
PHY62XXSDK
├─components                      ;SDK组件, 包括BLE API, GATT Profiles, 芯片硬件驱动和其他软件组件
├─example                         ;例程
│  ├─ble_central                  ;
│  ├─ble_peripheral               ;
│  │  ├─alternate_iBeacon         ;alternate iBeacon样例
│  │  ├─ancs                      ;苹果消息中心（ANCS）例程
│  │  ├─bleI2C_RawPass            ;I2C透传例程
│  │  ├─bleSmartPeripheral        ;综合Peripheral例程
│  │  ├─bleUart-RawPass           ;UART透传例程
│  │  ├─eddystone                 ;eddystone例程
│  │  ├─HIDKeyboard               ;HID例程
│  │  ├─hrs                       ;Heart rate profile演示例程
│  │  ├─iBeacon                   ;iBeacon例程
│  │  ├─otaDemo                   ;最基本的OTA演示例程
│  │  ├─pwmLight                  ;通过BLE命令控制PWM进行LED亮度调整的例程
│  │  ├─RawAdv                    ;简单的广播例程, 用于无线胎压监测一类的应用
│  │  ├─Sensor_Broadcast          ;
│  │  ├─wrist                     ;综合样例, 用于运动手环一类的应用
│  │  ├─wrist_aptm                ;基于综合样例, 通过AP Timer+OSAL Timer实现实时性较高的时钟
│  │  └─XIPDemo                   ;部分实时性要求较低的代码直接运行于内部flash的例程
│  ├─OTA                          ;
│  │  ├─OTA_internal_flash        ;OTA bootloader
│  │  └─OTA_upgrade_2ndboot       ;特殊应用, 用于升级OTA bootloader自身
│  └─peripheral                   ;
│      ├─adc                      ;ADC驱动应用样例
│      ├─ap_timer                 ;AP Timer驱动应用样例
│      ├─fs                       ;文件系统样例
│      ├─gpio                     ;GPIO演示例程
│      ├─kscan                    ;4x4按键演示例程
│      ├─lcd_ST7789VW             ;240x240 TFT彩屏演示例程
│      ├─pwm                      ;PWM演示例程
│      ├─qdec                     ;QDEC演示例程
│      ├─spiflash                 ;外部SPI演示例程
│      ├─voice                    ;语音采集演示例程
│      ├─voice_sbc                ;SBC格式语音采集编码演示例程
│      └─watchdog                 ;看门狗演示例程
├─lib                             ;lib文件和.h文件, 包括蓝牙协议栈更新, 字库
│  └─font                         ;字库的资源文件和升级文件
└─misc                            ;ROM symble table和跳转表
```

Figure 1: SDK Directory Structure

2.2 Development board and pre-compiled sample

We provide three kinds of development boards, of which the 32pin packaged development board is the small board shown in Fig. 2 for user extension function verification. The 48pin packaged chip development board corresponds to the development board in Fig.3. Development board integrates an acceleration sensor, a 64*128 resolution OLED screen, a heart rate sensor, an adjustable brightness LED light, a keyboard, SPI Flash and other peripherals. It is used for verification and demonstration of more complex applications. There is also a smaller development board with a 32pin packaged chip as the mesh development board, as shown in Fig. 4.

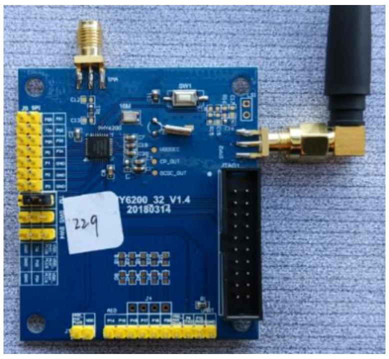


Figure 2: 32pin packaged development board

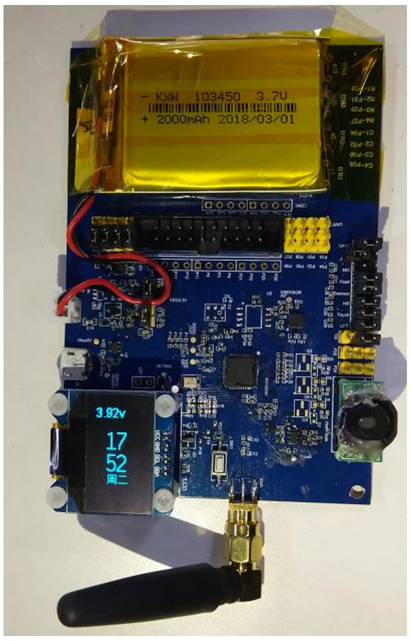


Figure 3: 48pin packaged chip development board

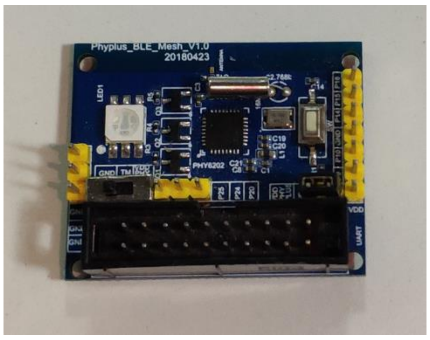


Figure 4: 32pin packaged mesh development board

In the directory where the SDK is located, the example directory contains sample projects. Each sample has a bin directory. Hex file is executable program firmware. Users who use this SDK for the first time can directly burn the file to perform experiment. For the burning method of the program, please refer to the chapter: Debugging and Programming

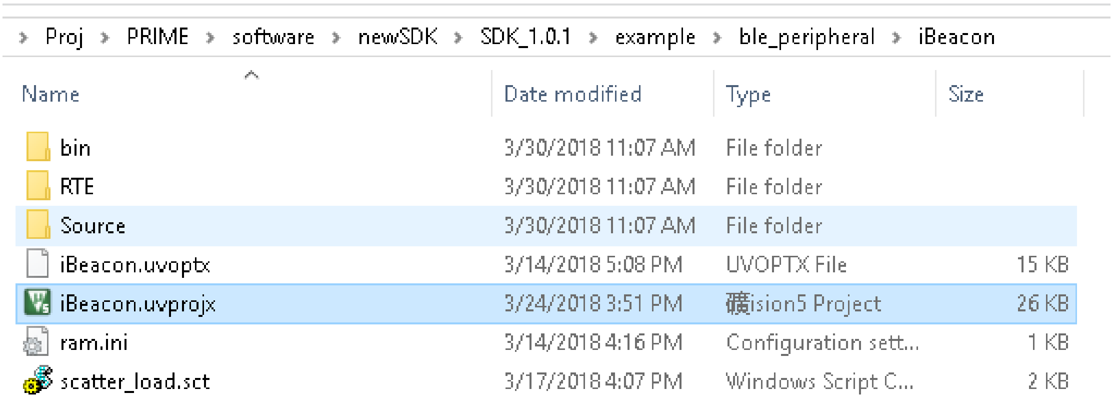
2.3 Install the development environment

Please install in development environment according to the following steps:

1. Copy the SDK to the working directory.
2. Install MDK Keil5 for ARM development environment.
3. Open the sample item file in the SDK directory through MDK to compile and debug the project.

2.4 Compile and run the sample

1. Use the PlyPlusKit tool to delete the firmware that has been burned on the development board (please refer to the tool user guide for the usage of the PlyPlusKit tool:< PhyPlusKit User's Guide.pdf>).
2. Select a sample from the SDK installation directory → example → ble_perpheral of the browser, such as iBeacon, and open the MDK item file:



› Proj › PRIME › software › newSDK › SDK_1.0.1 › example › ble_peripheral › iBeacon

Name	Date modified	Type	Size
bin	3/30/2018 11:07 AM	File folder	
RTE	3/30/2018 11:07 AM	File folder	
Source	3/30/2018 11:07 AM	File folder	
iBeacon.uvoptx	3/14/2018 5:08 PM	UVOPTX File	15 KB
iBeacon.uvprojx	3/24/2018 3:51 PM	礦ision5 Project	26 KB
ram.ini	3/14/2018 4:16 PM	Configuration sett...	1 KB
scatter_load.sct	3/17/2018 4:07 PM	Windows Script C...	2 KB

Figure 5: Select a sample from the SDK installation directory

3. Compile the project, then click the debug button to load the firmware for debugging, and then the stand-alone run button to run the program.
4. At this time, the firmware program is already running on the development board and can be interacted with through the BLE tool of the mobile phone.

2.5 Debugging and burning

- In the MDK toolbar button, click the Option for target button to open the option dialog box of the project.

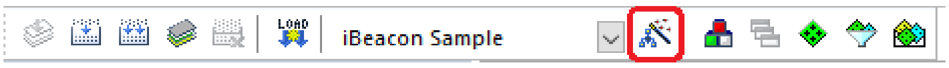


Figure 6: The option for target button in MDK toolbar

- In the Preprocessor Symbols → Define of the C/C++ tab, developers can change the corresponding pre-compiled macros:

- CFG_SLEEP_MODE=PWR_MODE_SLEEP: Enable low power consumption mode. During the execution of the firmware program, it will go to sleep during the idle process. After sleep, the debugger cannot debug and trace, and the breakpoint is also invalid
- CFG_SLEEP_MODE=PWR_MOD_ENO_SLEEP: Turn off the low power consumption mode. During the execution of the firmware program, the processor is always awake.
- DEBUG_INFO=1: Enable debugging information, output through serial port by default: P9(Tx),P10(Rx)
- DEBUG_INFO=0: Turn off debugging information.

Debugging code:

- Please use PhyPlusKit tool to erase the firmware that has been burned in Flash. Before erasing, please make sure that the TM jumper of the development board is connected to a high level, and press the Reset button.
- After confirming that the Flash Erase is successful, connect the TM jumper to low level, and press the Reset button to restart the chip
- Click the Debug button on the MDK toolbar to download the Image file to SRAM for online debugging.

Firmware burning:

- You need to erase the Flash first, and then perform burning. Refer to the PhyPlusKit documentation for the burning method.

3 Platforms and drivers

3.1 Introduction to the platform

Firmware architecture of PHY62XX is divided into three parts, ROM part, OTA Bootloader, and application firmware. Among them, OTA Boot loader is an optional process. The following figure shows two startup process flows.

Support OTA mode start:

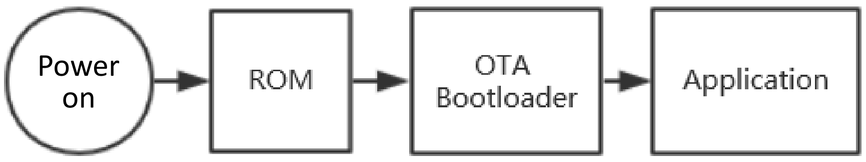


Figure 7: Flow of support OTA mode

Start in non-OTA mode:

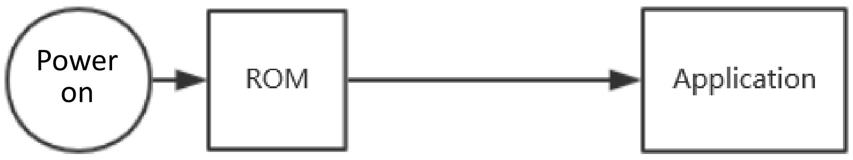


Figure 8: Flow of non-OTA mode

- ROM part: start and boot-or Boot loader or Application and BLE protocol stack API, start by selecting the burning mode (high level) or normal startup mode (low level) through the high and low level of the TM pin.
- OTA Boot loader: used to guide Application and handle OTA upgrades.
- Application: Application code, most of the secondary development work is concentrated in the Application part.

3.2 Software block diagram

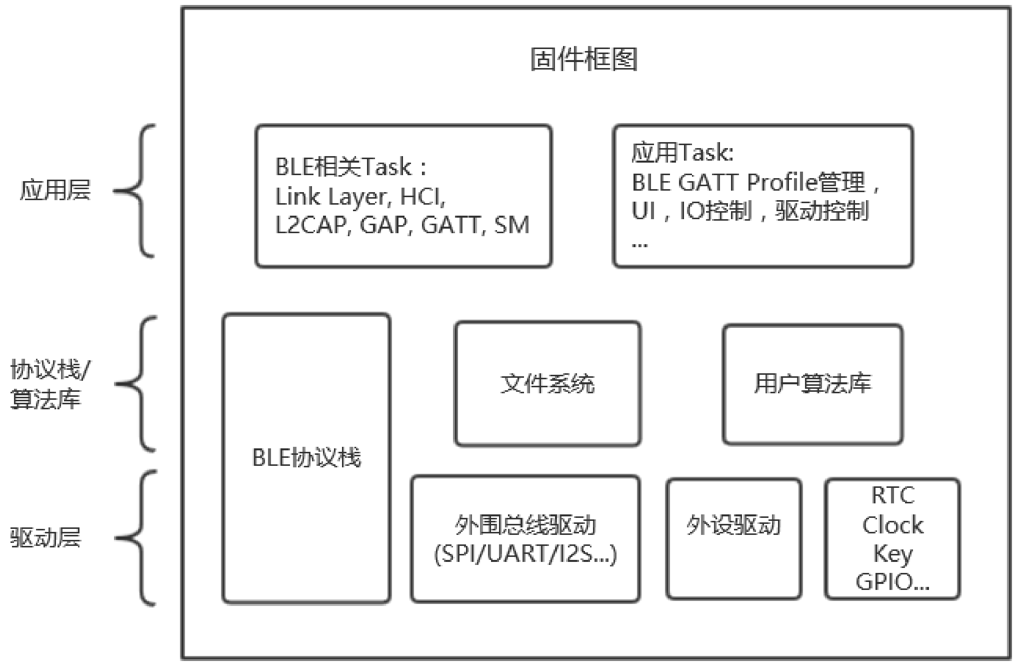


Figure 9: Software block diagram

The SDK does not use a third-party RTOS, but abstracts the concept of Task at the application layer. For BLE applications, the following tasks are necessary. Each task includes an initialization function and an event processing function. For details, please refer to the following table. Applications generally define one or more tasks, and typical scenarios (routines) generally use only one application task. Human-computer interaction, peripheral control, BLE broadcast and connection configuration, GATT profile loading and other tasks are all implemented in the application Task. Within tasks 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 handling function.

HCI_Init()
HCI_ProcessEvent(uint8, uint16)
	HCI layer initialization and corresponding event handling function

L2CAP_Init()
L2CAP_ProcessEvent(uint8, uint16)
	L2CAP initialization and corresponding event handling function

GAP_Init()
GAP_ProcessEvent(uint8, uint16)
	GAP initialization and corresponding event handling function

GATT_Init()
GATT_ProcessEvent(uint8, uint16)
	GATT initialization and corresponding event handling function

SM_Init()
SM_ProcessEvent(uint8, uint16)
	SM (Safety Management) initialization and corresponding event handling function

GAPRole_Init()
GAPRole_ProcessEvent(uint8, uint16)
	GAP configuration initialization and corresponding event handling function

GATTServApp_Init()
GATTServApp_ProcessEvent(uint8, uint16)
	GATT service initialization and corresponding event handling function

- If you need to support SMP, you also need to join the Bond Manager task:

GAPBondMgr_Init()
GAPBondMgr_ProcessEvent(uint8, uint16)
	Used to support SMP

3.2.1 OSAL

The 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 event handler 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. A Task can declare up to 16 event types.

- Return value

0	Succeeded.
Other values	Reference<comdef.h>

3.2.1.2 uint8 osal_clear_event(uint8 task_id, uint16 event_flag)

Clear an event.

- Parameter

Type	Parameter name	Description
uint8	task_id	Task ID.
uint16	event_flag	Event ID. Each Bit corresponds to an event. A Task can declare up to 16 event types.

- Return value

0	Succeeded.
Other values	Reference<comdef.h>

3.2.1.3 uint8 osal_msg_send(uint8 task_id, uint8 *msg_ptr)

Send a message to a Task.

- Parameter

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

- Return value

0	Succeeded.
Other values	Reference<comdef.h>

3.2.1.4 uint8 *osal_msg_receive(uint8 task_id)

When receiving a message, this function is used in the event processing function of Task. Corresponding event is SYS_EVENT_MSG (0x8000). This event is reserved for message processing. After receiving the event SYS_EVENT_MSG, use this function to obtain the message pointer. 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	Message not received.

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

Start an application Timer. When the timeout period is reached, the system will send an event to the specified task. After the timer completes an event, it will be automatically closed without retransmission.

- Parameter

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

- Return value

SUCCESS	Succeeded.
NO_TIMER_AVAIL	Failed 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. When the timeout period is reached, the system will send an event to the specified task. After this type of timer starts, it will send events to the task at the specified time interval until the function osal_stop_timerEx() stops the timer.

- Parameter

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

- Return value

SUCCESS	Succeeded.
NO_TIMER_AVAIL	Failed 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, which needs to be declared in the specified Task.

- Return value

SUCCESS	Succeeded.
INVALID_EVENT_ID	Timer ID does not exist (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, which needs to be declared in the specified Task.

- Return value

uint32	Remaining waiting time, if the timer is not started, return 0.

3.3 Hardware driver

3.3.1 Overview

This chapter introduces hardware module drivers and external bus drivers. In addition to the actual hardware drivers, a virtual module is also provided for virtual control of applications. For virtual control, please refer to the comprehensive use case of hand ring, which has applications for the virtual module MOD_USR0.

3.3.1.1 Module ID

3.3.1.2 MODULE_e

Module ID.

Value	Name	Description
0	MOD_NONE MOD_SOFT_RESET MOD_CPU	Usually it is an invalid device. Two aliases, MOD_SOFT_RESET and MOD_CPU, are temporarily reserved and do not take effect.
1	MOD_LOCKUP_RESET_EN	Temporarily invalid.
2	MOD_WDT_RESET_EN	Temporarily invalid.
3	MOD_DMA	DMA module.
4	MOD_AES	AES module.
5	MOD_TIMER	Timer module.
6	MOD_WDT	Watchdog module.
7	MOD_COM	Temporarily invalid.
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 number module
17	MOD_ADCC	ADC module
18	MOD_PWM	PWM module.
19	MOD_SPIF	Built-in 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	Application can be used as needed, and can manage the sleep of the application, and manage the additional operations of the wakeup and sleep of the application.

Table 1: Hardware module ID

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 Enumeration & Macro

3.3.2.1.1 CLK32K_e

Selection of 32K clock source.

CLK_32K_XTAL	Select the external crystal oscillator as the 32K clock source.
CLK_32K_RCOSC	Select the internal RC as the 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)

Turn off 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)

Get HCLK value.

- Parameter

  None.

- Return value

  Return the HCLK value.

3.3.2.3.5 uint32_t clk_pclk(void)

Get PCLK value.

- Parameter

  None.

- Return value

  Return the PCLK value.

3.3.2.3.6 void hal_rtc_clock_config(CLK32K_e clk32Mode)

Configure the 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 a 32bit unsigned number, and the time unit is 625 microseconds.

- Parameter

  None.

- Return value

  32bit unsigned system tick value.

3.3.2.3.8 uint32_t hal_ms_intv(uint32_t tick)

Time difference before the last time snapshot, in milliseconds. Input parameter is system tick.

- Parameter

Type	Parameter name	Description
uint32_t	tick	Tick value to be compared, the system tick recorded at some time before.

- Return value

  Time difference in milliseconds.

3.3.3 retention SRAM

In sleep mode, SRAM can be configured to retain or not retain data as required. Each SRAM can be independently switched. If configured to retain, RAM data can be retained after sleep wakeup, otherwise the data will be lost.
Usually the application will configure retention according to the actual SRAM usage. Generally at< main.c> hal_init function for configuration, please refer to API:
hal_pwrmgr_RAM_retention(uint32_t sram)
PHY62XX has 5 pieces of SRAM available in total, please refer to the table below for specific information:

RAM ID	Size	Address space	Description
SRAM0	32K Byte	1FFF_0000~1FFF_7FFF	Application and the protocol stack are shared, 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 to help reduce power consumption.
SRAM2	64K Byte	2000_0000~2000_FFFF	
SRAM3	8K Byte	2001_0000~2001_1FFF	
SRAM4	2K Byte	2001_2000~2001_27FF	

Table 2: SRAM specific information

3.3.4 GPIO

Provides GPIO-related operations, including GPIO configuration, pull-up/down settings, event-driven model of GPIO input mode, and GPIO wakeup operations.

3.3.4.1 Enumeration & Macro

3.3.4.1.1 GPIO pin definition

Among them, GPIO_DUMMY is defined as a virtual pin, which is generally used as an invalid 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

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

Bit_DISABLE	Configure as GPIO mode
Bit_ENABLE	Configured as a functional pin.

3.3.4.1.4 IO_Pull_Type_e

Configure the pin pull-up/down mode.

FLOATING	Configuration is Without pull-up/down, and the pin is suspended.
WEAK_PULL_UP	Configured as a weak pull-up.
STRONG_PULL_UP	Configured as strong pull-up.
PULL_DOWN	Configure as pull-down.

3.3.4.1.5 IO_Wakeup_Pol_e

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

POSEDGE	Configure to wake up on rising edge.
NEGEDGE	Configure to wake up on falling edge.

3.3.4.1.6 Fmux_Type_e

Configure the function settings of pin.

3.3.4.2 Data structure

3.3.4.2.1 gpioin_Hdl_t

Callback function prototype in GPIO input mode.

3.3.4.3 APIs

3.3.4.3.1 int hal_gpio_init(void)

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

This function needs to be set during system initialization, and is generally called in the hal_init() function. For details, please refer to the routine.

- Parameter

  None.

- Return value

PPlus_SUCCESS	Initialization succeeded.
Other values	Reference<error.h>

3.3.4.3.2 void hal_gpio_wakeup_set (GPIO_Pin_e pin,IO_Wakeup_Pol_e type)

Configure GPIO wakeup mode: Wake up on rising edge or wake up on falling edge.

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	pin	GPIO pin.
IO_Wakeup_Pol_e	type	Awakening 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 GPIO.

- Parameter

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

- Return value

  None.

3.3.4.3.5 3.3.4.3.5 uint32_t hal_gpio_read(GPIO_Pin_e pin)

Read the value of a certain GPIO.

- 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 to analog mode.

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin.
BitAction_e	value	Enable or disable the analog mode of IO

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.3.4.3.7 int hal_gpio_pull_set(GPIO_Pin_e pin,IO_Pull_Type_e type);

Set the pull-up/down of IO.

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin.
Pull_Type_e	type	IO pull-up/down settings.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.3.4.3.8 int hal_gpio_fmux_set(GPIO_Pin_e pin,Fmux_Type_e type)

Configure the functional mode of IO.

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin.
Fmux_Type_e	type	Functional mode of IO

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<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 input mode of GPIO. In this mode, interrupt and wakeup callbacks are supported, including rising edge callback and falling edge callback functions.

- Parameter

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

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.3.4.3.10 int hal_gpioin_unregister(GPIO_Pin_e pin)

Log off the GPIO input mode. In this mode, interrupt and wakeup callbacks are supported, including rising edge callback and falling edge callback functions. These functions are invalid after logging off.

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.3.4.3.11 int hal_gpioin_enable(GPIO_Pin_e pin)

For the pin that has been registered as gpioin, if the pin has been disabled (refer to hal_gpioin_disable), the gpioin function can be enabled through this function.

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.3.4.3.12 int hal_gpioin_disable(GPIO_Pin_e pin)

For the pin that has been registered as gpioin, if the pin has been enabled, the gpioin function can be stopped through this function.

- Parameter

Type	Parameter name	Description
GPIO_Pin_e	Pin	GPIO pin.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.3.5 I2S

3.3.5.1 Enumeration & Macro

3.3.5.2 Data structure

3.3.5.3 APIs

3.3.6 I2C Master

This chapter introduces the I2C Master mode.
PHY62XX has two I2C controllers that can be used, one of which can be selected through API.
This chapter mainly introduces the basic usage method of I2C Master with examples. Application of the module is introduced through actual use cases, taking the acceleration sensor KX023 as an example.

3.3.6.1 Reference code

3.3.6.1.1 I2C initialization

Initialization process mainly includes IO configuration and I2C initialization. Initialization 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 Deinit, 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 and Write

Slave address of the slave device needs to be input during the reading and writing process. Please refer to the following code for specific implementation. Due to the real-time requirements, the FIFO cannot be empty during the transmission process, so the process of writing the FIFO is not allowed to 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;
    hal_i2c_addr_update(pi2c, KX023_SLAVE_ADDR);
    {
        HAL_ENTER_CRITICAL_SECTION();
        hal_i2c_tx_start(pi2c);
        hal_i2c_send(pi2c, data, 2);
        HAL_EXIT_CRITICAL_SECTION();
    }
    return hal_i2c_wait_tx_completed(pi2c);
}
```

3.3.7 I2C Slave

TBD.

3.3.8 PWM

3.3.9 Enumeration & Macro

3.3.9.1 PWMN_e

Enumerate PWM physical channels 0~5.

3.3.9.2 PWM_CLK_DIV_e

Clock frequency division.

PWM_CLK_NO_DIV	16MHz clock, no 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 initialization includes turning on the clock, configuring the channel, configuring the frequency division parameters, working mode, and polarity.

- Parameter

Type	Parameter name	Description
PWMN_e	pwmN	Channel parameters.
PWM_CLK_DIV_e	pwmDiv	Frequency division parameters.
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)

Enable the PWM channel and bind the IO pin.

- Parameter

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

- Return value

  None.

3.3.9.4.3 void hal_pwm_close_channel(PWMN_e pwmN)

Close the PWM channel.

- Parameter

Type	Parameter name	Description
PWMN_e	pwmN	Channel parameters.

- 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 parameters.

- Return value

  None.

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

Configure the counter data to configure the PWM duty cycle.

- Parameter

Type	Parameter name	Description
PWMN_e	pwmN	Channel parameters.
uint16	cmpVal	Counter
uint16_t	cntTopVal	Total count

- Return value

  None.

3.3.9.4.6 void hal_pwm_start(void)

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

- 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 opened PWM channel.

- Parameter
  None.

- Return value
  None.

3.3.10 QDEC

TBD.

3.3.11 ADC

ADC driver provides asynchronous AD acquisition function. After data acquisition is completed, the ADC acquisition result is returned through the callback function. 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 10 below assumes that two ADC channels are acquired at the same time.

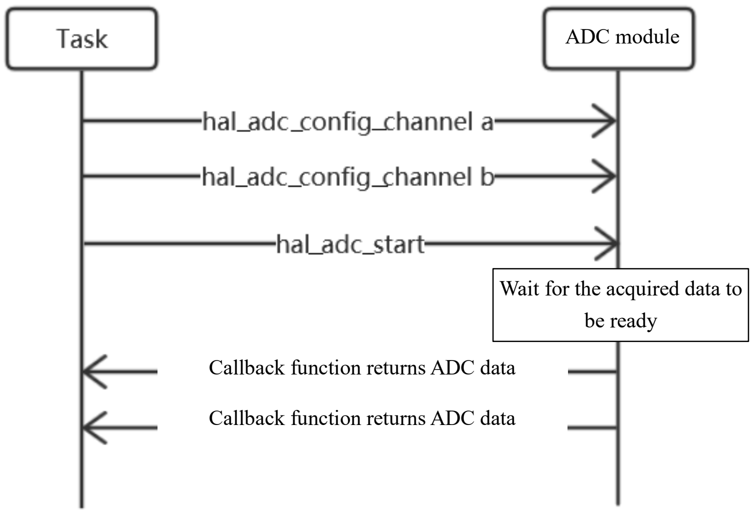

Task
ADC module
hal_adc_config_channel a
hal_adc_config_channel b
hal_adc_start
Wait for the acquired data to be ready
Callback function returns ADC data
Callback function returns ADC data

Figure 10: Two ADC channels are acquired at the same time

3.3.11.1 Enumeration & Macro

3.3.11.1.1 adc_CH_t

ADC physical channel.

ADC_CH0	This channel is not currently supported.
ADC_CH1	This channel is not currently supported.
ADC_CH1P or ADC_CH1DIFF	In single-ended mode, it is used as channel 1P and works independently. In differential mode, it is used as 1DIFF channel and used in combination with 1N channel.
ADC_CH1N	As channel 1N in single-ended mode, invalid in differential mode.
ADC_CH2P or ADC_CH2DIFF	In single-ended mode, it is used as a channel 2P, which works independently, and in differential mode, it is used as a 2DIFF channel and used in combination with 2N channels.
ADC_CH2N	As channel 2N in single-ended mode, invalid in differential mode.
ADC_CH3P or ADC_CH3DIFF	In the single-ended mode, it is used as the channel 3P, working independently, and in the differential mode as the 3DIFF channel, and used in combination with the 3N channel.
ADC_CH3N	As channel 3N in single-ended mode, invalid in differential mode.
ADC_CH_VOICE	Voice channel is used for acquiring analog microphones.

3.3.11.1.2 ADC events

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

HAL_ADC_EVT_DATA	ADC sampling data, if the sampling data is ready, the registered ADC callback function will be called to send the event.
HAL_ADC_EVT_FAIL	ADC sampling failed.

3.3.11.2 Data structure

3.3.11.2.1 adc_Evt_t

Data structure of the ADC driving event.

Int	type	ADC event type.
adc_CH_t	ch	ADC channel
uint16_t*	data	ADC sampling data.
uint8_t	size	Size of 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 parameters.

Bool	is_continue_mode	Whether the continuous acquisition mode, if it is TRUE, ADC acquisition will continue to work until manually stopped.
Bool	is_differential_mode	Whether it is a differential mode, if it is a differential mode, the P terminal and N terminal need to be configured in pairs.
Bool	is_high_resolution	Whether to use attenuation, if the value is TRUE, attenuation is used, and the range is 0V~1V, otherwise the range is 0V~4V.
Bool	is_auto_mode	This function is temporarily not supported.

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 use, otherwise the result cannot be predicted or an error is returned.

- 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 the ADC acquisition channel.

- Parameter

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

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.3.11.3.3 int hal_adc_start(void)

Start acquiring.

- Parameter

  None.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.3.11.3.4 int hal_adc_stop(void)

Stop acquiring.

- Parameter

  None.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<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	Sampled raw data.
uint8_t	size	Number 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	Voltage value of 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 the two sets of SPI by time.
SPI transmission mode, you can choose polling mode or interrupt mode.
If the amount of data is small, the transmission is completed at one time, and it is convenient and simpler to directly use the polling method.
If the amount of data is large, the transmission is completed several times, either in polling mode or interrupt mode. Polling method will transmit each frame of data in turn, until the data transmission is completed, the program will continue to execute. Interrupt mode is executed downwards after the first frame is sent. Every time the data transfer is completed, it asks for data from the application and receives the data at the same time. Data sending buffer and receiving buffer are configured before transmission, and 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 enable is not allowed until the data transmission is completed, and the SPI informs the application that the transmission is complete through the callback function.
When sending data in interrupt mode, memory needs to be allocated 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.
For specific use, please refer to the SPI flash example.

3.3.12.1 Enumeration & Macro

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 operation 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 transfer completion callback function, need to be configured when transferring in interrupt mode.

3.3.12.1.5 spi_Type_t

SPI transfer completion callback function parameter.

TRANSMIT_COMPLETED	Transmission complete

3.3.12.2 Data structure

3.3.12.2.1 spi_Cfg_t

SPI configuration parameters.

GPIO_Pin_e	sclk_pin	Configure the clk pin
GPIO_Pin_e	ssn_pin	Configure the cs pin
GPIO_Pin_e	MOSI	Configure the mosi pin
GPIO_Pin_e	MISO	Configure the miso pin
uint32_t	baudrate	Set the spi baud rate
SPI_TMOD_e	spi_tmod	Configure spi operation mode
SPI_SCMOD_e	spi_scmod	Configure spi operation mode
bool	int_mode	Transmission using interrupt mode/non-interrupt mode
bool	force_cs	Use manual setting/IP to pull cs pin
spi_Hdl_t	evt_handler	When using interrupt mode 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 they can be used, otherwise the result cannot be predicted or an error is returned.

- 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 and 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	Succeeded.
Other values	Reference<error.h>

3.3.12.3.3 int hal_spi_bus_deinit(hal_spi_t* spi_ptr)

Disable the SPI module.

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<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	Succeeded.
Other values	Reference<error.h>

3.3.12.3.5 int hal_spi_set_force_cs(hal_spi_t* spi_ptr,bool en)

Set the 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 state False—IP sets cs status

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<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 sending data using interrupt mode, a buffer needs to be allocated for it.

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle.
uint8_t*	tx_buf	Send buffer pointer
uint16_t	len	Send buffer length

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<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 transmission.

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle.
uint8_t*	tx_buf	Send buffer pointer
uint8_t*	rx_buf	Receive buffer pointer
uint16_t	len	Length of the buffer, the length of the sending buffer and the receiving buffer must be equal

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.3.12.3.8 bool hal_spi_get_transmit_bus_state(hal_spi_t* spi_ptr)

Check the SPI status, whether there is data being transmitted in interrupt mode.

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle

- Return value

true	SPI idle
false	SPI is busy, there is data that has not yet been transferred

3.3.12.3.9 int hal_spi_module_select(hal_spi_t* spi_ptr)

SPI module selection, it will be called only when both groups of SPI are used. Two groups of SPI peripheral application circuits are independent, and the software is time-sharing multiplexed.

- Parameter

Type	Parameter name	Description
hal_spi_t*	channel	SPI channel handle

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.3.12.3.10 void hal_spi_send_byte(hal_spi_t* spi_ptr,uint8_t data)

SPI sends a data, and the interface is reserved to meet forward compatibility.

- Parameter

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

- 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 to meet 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 controlling UART synchronous or asynchronous mode.
For Rx, this module only provides asynchronous mode interface. When Rx data is ready, it will throw UART_EVT_TYPE_RX_DATA event or UART_EVT_TYPE_RX_DATA_TO (timeout: when the last data is less than the FIFO Threshold) through the callback function registered during initialization.
For Tx, this module provides both synchronous and asynchronous interfaces. If cfg.use_tx_buf is set to FALSE during initialization and Tx is synchronous, the function hal_uart_send_buff (uint8_t *buff,uint16_t len) can be called to send data synchronously, and the function is blocking.
If cfg.use_tx_buf is set to TRUE during initialization and cfg.use_fifo is TRUE, then Tx is asynchronous. After the initialization is completed, you need to call hal_uart_set_tx_buf(uint8_t* buf, uint16_t size) to configure the Tx buffer, and then you can use hal_uart_send_buff (uint8_t *buff,uint16_t len) Send data. At this time, the function is non-blocking. After the transmission is completed, the UART_EVT_TYPE_TX_COMPLETED event will be thrown through the callback function, and the application will respond to the event as needed.

3.3.13.1 Enumeration & Macro

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 is ready.
UART_EVT_TYPE_RX_DATA_TO	UART Rx data is ready and receiving timeout.
UART_EVT_TYPE_TX_COMPLETED	UART Tx transmission completed.

3.3.13.2 Data structure

3.3.13.2.1 uart_Cfg_t

UART configuration parameters.

GPIO_Pin_e	tx_pin	Set Tx pin
GPIO_Pin_e	rx_pin	Set Rx pin
uint32_t	Baudrate	Set the transmission baud rate
bool	use_fifo	Use FIFO? Recommended.
bool	use_tx_buf	Whether to use Tx cache, if cache is used, Tx is asynchronous mode
uart_Hdl_t	evt_handler	Set the callback function.
GPIO_Pin_e	ts_pin	Temporarily not supported.
GPIO_Pin_e	cts_pin	Temporarily not supported.
bool	hw_fwctrl	Temporarily not supported.
bool	parity	Temporarily not supported.

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 can be used after configuration, otherwise the result cannot be predicted or an error is returned.

- Parameter

Type	Parameter name	Description
uart_Cfg_t	cfg	Serial port configuration information

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.3.13.3.2 int hal_uart_set_tx_buf(uint8_t* buf, uint16_t size)

Set Tx buffer, 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 buffer pointer.
uint16_t	size	Cache size.

- Return value

PPlus_SUCCESS	Cache configuration succeeded.
Other values	Reference<error.h>

3.3.13.3.3 int hal_uart_get_tx_ready(void)

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

- Parameter

  None.

- Return value

PPlus_SUCCESS	Tx transmission completed.
PPlus_ERR_BUSY	Tx is 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 sending is completed or the sending timeout returns, if it is in asynchronous mode, the function is non-blocking, after the data transmission is completed, the UART_EVT_TYPE_TX_COMPLETED event is thrown through the callback function.

- Parameter

Type	Parameter name	Description
uint8_t*	buf	Data to be sent.
uint16_t	len	Size.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.3.14 Key Scan

TBD.

3.3.15 Flash

3.3.15.1 Enumeration & Macro

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 written to flash

- Return value

PPlus_SUCCESS	Write-in is successful
Other values	Reference<error.h>

3.3.15.3.2 uint32_t flash_read_ucds(uint32_t addr)

Used for users to read the 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)

- Return value

0xffffffff	Flash value is 0xffffffff or the address is wrong
Other values	Flash address 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)

Used to erase the sector of the specified address by the user, the user address range is 0x0000~0x3ffc.

- Parameter

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

- Return value

PPlus_SUCCESS	Write-in is successful
Other values	Reference<error.h>

3.4 Sleep management

PHY62XX series chips provide two external sleep modes: sleep mode and Power Off mode
For sleep mode, the SDK centrally manages sleep and wakeup, and enables or disables sleep through pre-compiled macros. If the application code requires a certain process to disable sleep, please refer to the chapter for configuration.
For Power Off mode, SDK provides API for entering Power Off mode, and it can be configured through parameters.

3.4.1 Enable and disable sleep

It can be configured in the MDK item → Option dialog box → C/C++ page →→ Preprocessor Symbol → Define: →→→

CFG_SLEEP_MODE=PWR_MODE_SLEEP	Enable hibernation
CFG_SLEEP_MODE=PWR_MODE_NO_SLEEP	No hibernation

3.4.1.1 How does Application control sleep

Application code can obtain sleep control by registering the PwrMgr module. It shall be noted that the modules below MOD_VOC are reserved for actual hardware modules, MOD_USR0 is reserved for rfPhy, and the modules that can be used by the application are MOD_USR1 and above. Please refer to the module ID for module definition:

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

//Application prohibits sleep
hal_pwrmgr_lock(MOD_USR1);
//Application enable sleep
hal_pwrmgr_lock (MOD_USR1);
```

3.4.2 Power Off mode

If you need to enter Power Off mode, you can call the function hal_pwrmgr_poweroff (pwroff_cfg_t cfg) to enter sleep, and the parameter cfg is used to configure the wakeup pin. In Power off mode, the system is in 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)

Module initialization.

- Parameter

  None.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.4.3.1.2 bool hal_pwrmgr_is_lock(MODULE_e mod)

Query whether a certain module (hardware or application module) is forbidden to sleep.

- Parameter

Type	Parameter name	Description
MODULE_e*	mod	Module ID.

- Return value

TRUE	This module prohibits sleep.
FALSE	This module allows hibernation

3.4.3.1.3 int hal_pwrmgr_lock(MODULE_e mod)

Module is prohibited from sleeping.

- Parameter

Type	Parameter name	Description
MODULE_e*	mod	Module ID.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.4.3.1.4 int hal_pwrmgr_unlock(MODULE_e mod)

Module allows sleep.

- Parameter

Type	Parameter name	Description
MODULE_e*	mod	Module ID.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<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 modules, registering sleep callback functions, and registering wakeup callback functions. Module is a mandatory option, and the sleep and wakeup callback functions are optional parameters. If you do not need to set them, set the parameters to NULL.

- Parameter

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

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.4.3.1.6 int hal_pwrmgr_unregister(MODULE_e mod)

Module logs off from the sleep management system.

- Parameter

Type	Parameter name	Description
MODULE_e*	mod	Module ID.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.4.3.1.7 int hal_pwrmgr_RAM_retention(uint32_t sram)

Configure the RAM that can be retained during sleep. Bits 0~4 are valid and correspond to RAM0~RAM4 respectively. If the corresponding Bit is 1, this RAM can retain data during sleep. If it is 0, the data will be lost during sleep.

- Parameter

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

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.5 Libraries

3.5.1 File system

A lightweight file system based on internal flash, the file system provides a set of synchronization APIs, supports 16bit file ID, file search, read, write, delete, file system query, and garbage file recovery.

3.5.1.1 Enumeration & Macro

3.5.1.1.1 FS_SETTING

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

FS_ITEM_LEN_16BYTE	Length of each record is 16 bytes
FS_ITEM_LEN_32BYTE	Length of each record is 32 bytes
FS_ITEM_LEN_64BYTE	Length of each record is 64 bytes

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 configuration parameters. This function needs to be set during system initialization. For details, please refer to the routine.

- Parameter

Type	Parameter name	Description
uint32_t	fs_start_address	FS start address. 4096-byte alignment is required, and space cannot conflict with other uses.
uint8_t	sector_num	Number of FS sectors, the effective value is 3~78. Example: Assign FS 4 sectors, starting address is 0x11005000 hal_fs_init(0x11005000,4)

- Return value

PPlus_SUCCESS	Initialization succeeded.
Others	Reference<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 file.

- Parameter

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

- Return value

PPlus_SUCCESS	Initialization succeeded.
Others	Reference<error.h>

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

Write FS file.

- Parameter

Type	Parameter name	Description
uint16_t	id	Write file id.
uint8_t*	buf	Incoming buffer start address.
uint16_t	len	Incoming buffer start length

- Return value

PPlus_SUCCESS	Initialization succeeded.
Other values	Reference<error.h>

3.5.1.3.4 uint32_t hal_fs_get_free_size(void)

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

- Parameter

  None.

- Return value

  FS can store file space in bytes.

3.5.1.3.5 int hal_fs_get_garbage_size(uint32_t* garbage_file_num)

Size of the data area of the FS deleted file, in bytes.

- Parameter

Type	Parameter name	Description
uint32_t*	garbage_file_num	Number of files deleted.

- Return value

  FS Space occupied by deleted files, in bytes.

3.5.1.3.6 int hal_fs_item_del (uint16_t id)

Delete a file.

- Parameter

Type	Parameter name	Description
uint16_t	id	Delete file id.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<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 erase multiple sectors, which takes a relatively long time.
It is recommended to execute when the CPU is idle and there are more garbage. Execution time is related to the main frequency and the size of the FS.

- Parameter

  None.

- Return value

PPlus_SUCCESS	Initialization succeeded.
Other values	Reference<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. Execution time is related to the main frequency and FS size.

- Parameter

Type	Parameter name	Description
uint32_t	fs_start_address	FS start address. 4096-byte alignment is required, and space cannot conflict with other uses.
uint8_t	sector_num	Number of FS sectors, the effective value is 3~78. Example: Assign FS 4 sectors, starting address is 0x11005000 hal_fs_init(0x11005000,4)

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.5.1.3.9 bool hal_fs_initialized(void)

Query the initialization status of FS.

- Parameter

  None.

- Return value

true	It has been initialized and can be used.
false	It is not initialized and cannot be used.

3.5.2 Date and time

To achieve the perpetual calendar function, datetime_t is the time accurate to the second. As a service running in the background, the library is driven by osal_timer to realize the automatic update of the time, and provides APIs for time acquisition and time setting.

3.5.2.1 Enumeration & Macro

3.5.2.1.1 USE_SYS_TICK

TRUE	Use osal_sys_tick as the clock reference count, in the case of using RC as the 32K clock source, osal_sys_tick as the calibrated tick value, with higher accuracy
FALSE	Use the RTC counter as the reference count of 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	Date (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)

Calendar application synchronization function requires synchronization once a minute (or less than one minute). No precise call is 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 that needs to be set, accurate to the second

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

3.5.2.3.4 int app_datetime(datetime_t* pdtm)

Get the current system time.

- Parameter

Type	Parameter name	Description
datetime_t*	pdtm	Output parameter, if it returns successfully, pdtm will be written into the latest system time.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

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

Compare two times and get the difference in seconds.

- Parameter

Type	Parameter name	Description
const datetime_t*	pdtm_base	Time being compared.
const datetime_t*	pdtm_cmp)	Time of price comparison, the formula is pdtm_cmp – pdtm_base

- Return value

int	Time difference, if the time being compared is newer, the return value is negative

3.5.3 Dot matrix font

PHY62XX Font library provides multi-language, configurable font library solutions.
The font scheme includes two parts:

- PHY62XX Font library + API
  - Font library driver library and API function, providing font library registration, UTF-8 analysis and font library bitmap interface
  - Font library bitmap interface supports variable width text.
- PHY62XX multilingual font library
  - Multilingual fonts can be generated according to customer needs. In the Unicode basic multilingual plane (BMP 0x0000~0xffff), any language combination can be configured as required, and the text resolution and scanning method can be configured according to customer needs.
  - For Chinese, it can support the 2009 version of the 3500 common word generation program.
  - Font file provides binary format file (.bin) and upgrade file (.res).

3.5.3.1 APIs

3.5.3.1.1 void* ui_font_load(uint32_t flash_addr)

Load the font library, load the font library that has been burned in the internal flash, after loading, a font library handle will be obtained, and the bitmap of Unicode characters can be obtained by calling the function ui_font_unicode() through the handle.

- Parameter

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

- Return value

  Returns the void* type font library handle. If the value is NULL, it means loading 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	Input parameters, the input UTF-8 string ends with '\0'.
uint16_t *	unicode	Output parameter, a Unicode character obtained by converting UTF-8.

- Return value

  Returns int type.

0	String parsing is complete, and the output parameter is invalid data.
1~n	A Unicode character is converted, 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)

Obtain bitmap data based on Unicode characters.

- Parameter

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

- Return value

  Returns int type.

0	Operation succeeded.
Other values	Error code.

3.5.3.1.4 const char* ui_font_version(void)

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

- Parameter

  None.

- Return value

  const char* type string.

4 BLE

4.1 GAP

Universal access profile (GAP):
The GAP layer in the Ble protocol stack is responsible for handling device access modes, including device discovery, connection establishment, connection 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——a device that cannot be connected and is always broadcasting;
- Observer——a device that can scan for broadcast device, but cannot initiate a connection;
- Peripheral slave——a broadcast device that can be connected and can be used as a slave in a single link layer connection.
- Central host——can scan for 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 the host receives the broadcast data, it sends a scan request ScanRequest to the slave, and then the slave responds with specific data to the host, which is called a scan response ScanResponse. After the host receives the scan response, it knows that this is an external device that can establish a connection. This is the entire process of device discovery. At this time, the host can initiate a connection establishment request to the slave. Connection request includes the following parameters.

- Connection interval——frequency modulation mechanism is used in the connection of two BLE devices. Two devices use a specific channel to send and receive data, and then use a new channel after a period of time. (Link layer handles channel switching). Two devices send and receive data after the channel switch is called a connection event. Even if there is no application data sending and receiving, the two devices will still exchange link layer data to maintain the connection. Connection interval is two Time interval between two connection events, the connection interval is 1.25ms as the unit, and the value of the connection interval is 6 (7.5ms) ~ 3200 (4s).
- Slave delay——the setting of this parameter allows 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 consumption.
- Management timeout——this is the maximum allowable interval between two successful connection events. If this time is exceeded (the unit of this value is 10ms) without a successful connection event, the device is considered to have lost the connection and returns to not connected Status, management timeout 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)].
- Security 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 for the encrypted connection is formed. In typical applications,

suppose the concentrator is requested to provide a key to complete the pairing work. Key is a fixed value, such as 000000, or a data can be randomly generated and provided 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 pair of peripherals and the host will be connected and disconnected from time to time. There is a feature in ble's security mechanism that allows the establishment of long-term security key information between the two devices. This feature is called binding. Allows two devices to quickly complete encryption authentication when they are connected, without the need to perform the complete pairing process each time they connect.

4.2 GATT

GATT (Generic Attribute Profile): Generic Attribute Profile, which is constructed on the basis of the Attribute Protocol (ATT), and provides some general operations and frameworks for the transmission and storage of data in the Attribute Protocol.

4.2.1 How to implement custom services

This chapter describes how to implement custom services through the SDK.
We introduce how to implement a service through the BLE-Uart example, which contains three types of characteristic values: Write, Notify, and Indicate.

4.2.1.1 Create an attribute table

Attribute table is a static array of type gattAttribute_t, including:

- Master services.
- Attribute of characteristic value 1 (GATT_PROP_WRITE_NO_RSP| GATT_PROP_WRITE).
- Value of characteristic value 1
- Attribute of characteristic value 2 (GATT_PROP_NOTIFY| GATT_PROP_INDICATE).
- Client Characteristic Configuration Descriptor with characteristic value 2.
- Value of characteristic value 2.

For details, please refer to the SDK routine code.

```
static gattAttribute_t bleuart_ProfileAttrTbl[] =
{
//Master service
  {
    { ATT_BT_UUID_SIZE, primaryServiceUUID }, /* type */
    GATT_PERMIT_READ,                              /* permissions */
    0,                                             /* handle */
    (uint8 *)&bleuart_Service                /* pValue */
  },
//Characteristic value 1: Attribute (write, write-no response required)
    {
      { 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
      {
          { 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 property sheet configuration, we need to add services and register read and write callback functions. 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 callback of link status to respond to the change of connection status.
- Registration of the service, the registration of the service is realized by passing the attribute table and the GATT service callback function to the protocol stack.

- Save the callback function pointer of the application layer, which is used to send to the application layer.

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

Callback function, which responds to the write (Write or Write no response) operation on the Central side, contains two parts, the write operation for characteristic value 1; the write operation for the client characteristic value configuration descriptor of characteristic value 2. Please refer to the example code for specific implementation.

4.2.1.2.3 static void bleuart_HandleConnStatusCB (uint16 connHandle, uint8 changeType)

Callback function, in response to the change of link status.

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 calling service interface

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

4.3 OTA

OTA means Over the Air, which refers to the function of wirelessly upgrading firmware through BLE. The OTA of PHY62XX provides a complete and reliable over-the-air firmware upgrade program. Upgrade includes application firmware upgrade, resource file upgrade, and OTA bootloader upgrade.

- Application firmware upgrade
  Upgrade application firmware, support non-encrypted upgrade and encrypted upgrade.
- Resource file upgrade
  When upgrading resource files, the storage area of resource files is a non-program area and a system protected area. For the NVM area, the user needs to protect it independently. 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, usually 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 become invalid. Therefore, after the OTA bootloader is upgraded, the application firmware must be upgraded.

4.3.1 OTA mode

Application firmware upgrade process needs to be performed in OTA mode.
There are three ways to enter OTA mode:

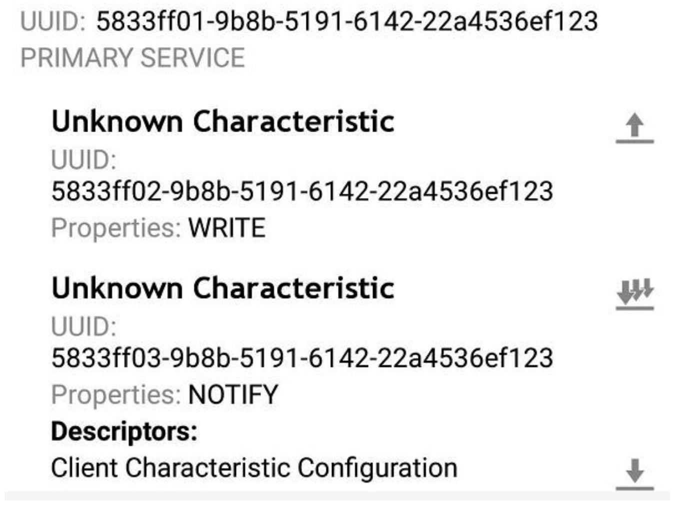
- Application firmware is not detected. If the application firmware is damaged or only the OTA Bootloader firmware is burned, the device will enter the OTA mode by default after reset or power-on.
- In the application mode, the system can enter the OTA mode through the OTA App Service instruction, and the application firmware supporting OTA requires the OTA App Service to be loaded. Through the Service, the host sends a control command to the characteristic value of the service to make the application firmware jump to OTA mode
- 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

Resource file upgrade process needs to be carried out in OTA Resource mode.
Application mode can enter the OTA Resource mode through the instruction of OTA App Service.

4.3.3 OTA Service

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.


UUID: 5833ff01-9b8b-5191-6142-22a4536ef123
PRIMARY SERVICE

Unknown Characteristic
UUID:
5833ff02-9b8b-5191-6142-22a4536ef123
Properties: WRITE

Unknown Characteristic
UUID:
5833ff03-9b8b-5191-6142-22a4536ef123
Properties: NOTIFY
Descriptors:
Client Characteristic Configuration

Figure 11: OTA Service loading interface

The OTA App Service code is located in SDK\components\profiles\ota_app.

4.3.4 OTA Bootloader

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

4.3.4.1 OTA Dual Bank Has FCT

Open up two symmetrical 128K spaces on the internal flash for the application firmware, and open up a separate 120K space for FCT firmware (functional test firmware). The Flash address 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
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 does not provide FCT firmware separately. Flash address 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
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 testing firmware. 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 testing firmware. 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, the application firmware needs to be encrypted through a PC tool, and the key is embedded in the chip. From an application perspective, the upgrade process is no different from non-encrypted OTA. If the key is If it does not match, an error message will be reported during the upgrade process.

4.3.6 How to implement OTA

If the chip burns the 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

Application firmware needs to load the OTA App Service. Generally speaking, we will load the service during the initialization of the application task. Please refer to the sample code: SDK\example\ble_peripheral\otaDemo
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
    GATTServApp_RegisterForMsg(otaDemo_TaskID);
    DevInfo_AddService();
    otaDemo_AddService();
    ota_app_AddService();



    // Setup a delayed profile startup
    osal_set_event( otaDemo_TaskID, START_DEVICE_EVT );
}
```

4.3.7 Burn application firmware and OTA bootloader

Application firmware and OTA Bootloader can be burned through the PhyPlusKit tool or Phyplus programmer hardware. For specific usage, please refer to the PhyPlusKit user guide and the programmer's user documentation.

4.3.8 OTA summary

In general, three steps are required to implement OTA:

- Compile the OTA Bootloader to obtain ota.hex, and the example provides four modes of hex files (for 512K flash version of the hardware).
- Apply firmware to add OTA App Service: ota_app_AddService();
- Program the chip through PhyPlusKit or PlyPlus writer.

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 program

Provide the application examples shown in the following table in the SDK:
○1 Routine can be run on the development board PHY6200_32_V1.4 and above;
○2 Routine can be run on the development board PHY6200_48_Bracelet_V1.0 and above;
○3 Routine can be run on the development board Phyplus_BLE_Mesh_V1.0 and above.

ble_peripheral		
alternate_iBeacon	Sample alternate iBeacon.	○1○2○3
ancs	Apple Message Center (ANCS) routine	○1○2○3
bleI2C_RawPass	I2C transparent transmission routine	○1○2○3
bleSmartPeripheral	Comprehensive BLE Peripheral routine	○1○2○3
bleUart-RawPass	UART transparent transmission routine	○1○2○3
eddystone	eddystone routine	○1○2○3
HIDKeyboard	HID routine	○1○2○3
hrs	Heart rate profile demo routine	○1○2○3
iBeacon	iBeacon routine	○1○2○3
otaDemo	The most basic OTA demo example	○1○2○3
pwmLight	A routine for controlling PWM to adjust LED brightness through BLE commands	○2○3
RawAdv	Simple broadcast routines for applications such as wireless tire pressure monitoring	○1○2○3
Sensor_Broadcast		○1○2○3
wrist	Comprehensive samples for applications such as sports bracelets	○2
wrist_aptm	Based on a comprehensive sample, realize a clock with higher real-time performance through AP Timer+OSAL Timer	○2
XIPDemo	Some codes with low real-time requirements run directly on the internal flash routines	○2○3
OTA		
OTA_internal_flash	OTA bootloader	○1○2○3
OTA_upgrade_2ndboot	Special application, used to upgrade OTA bootloader itself	○1○2○3
peripheral		
adc	ADC driving application sample	○1○2○3
ap_timer	AP Timer driver application sample	○1○2○3

fs	Sample file system	o1o2o3
gpio	GPIO demo example	o1o2o3
kscan	4x4 key demo example	o2
lcd_ST7789VW	240x240 TFT color screen demo example	o1
pwm	PWM demo example	o1o2o3
qdec	QDEC demo example	o1o2o3
spiflash	External SPI demo example	o2
voice	Voice capture demo routine	o2
voice_sbc	SBC format voice capture coding demo routine	o2
watchdog	Watchdog demo routine	o1o2o3