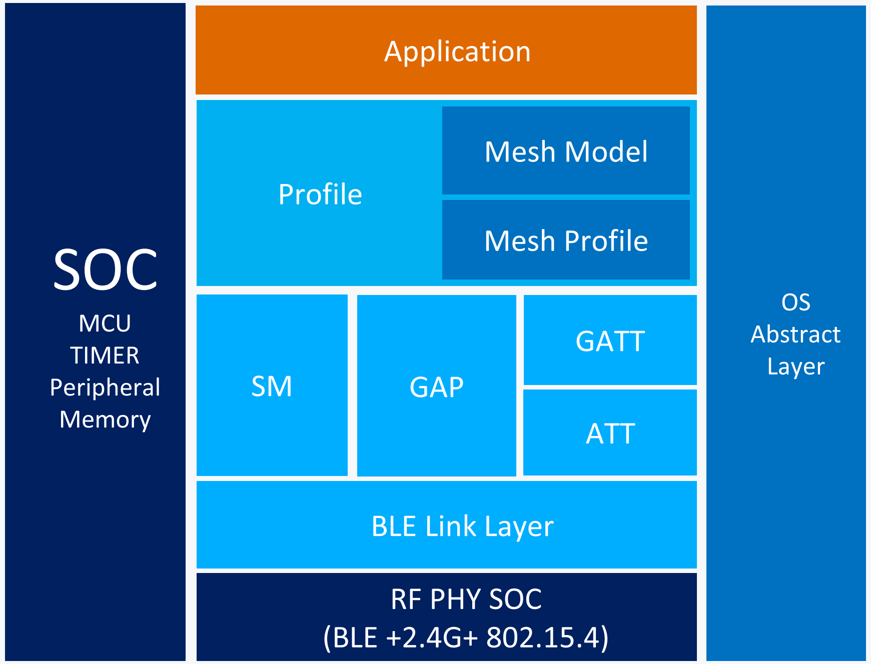
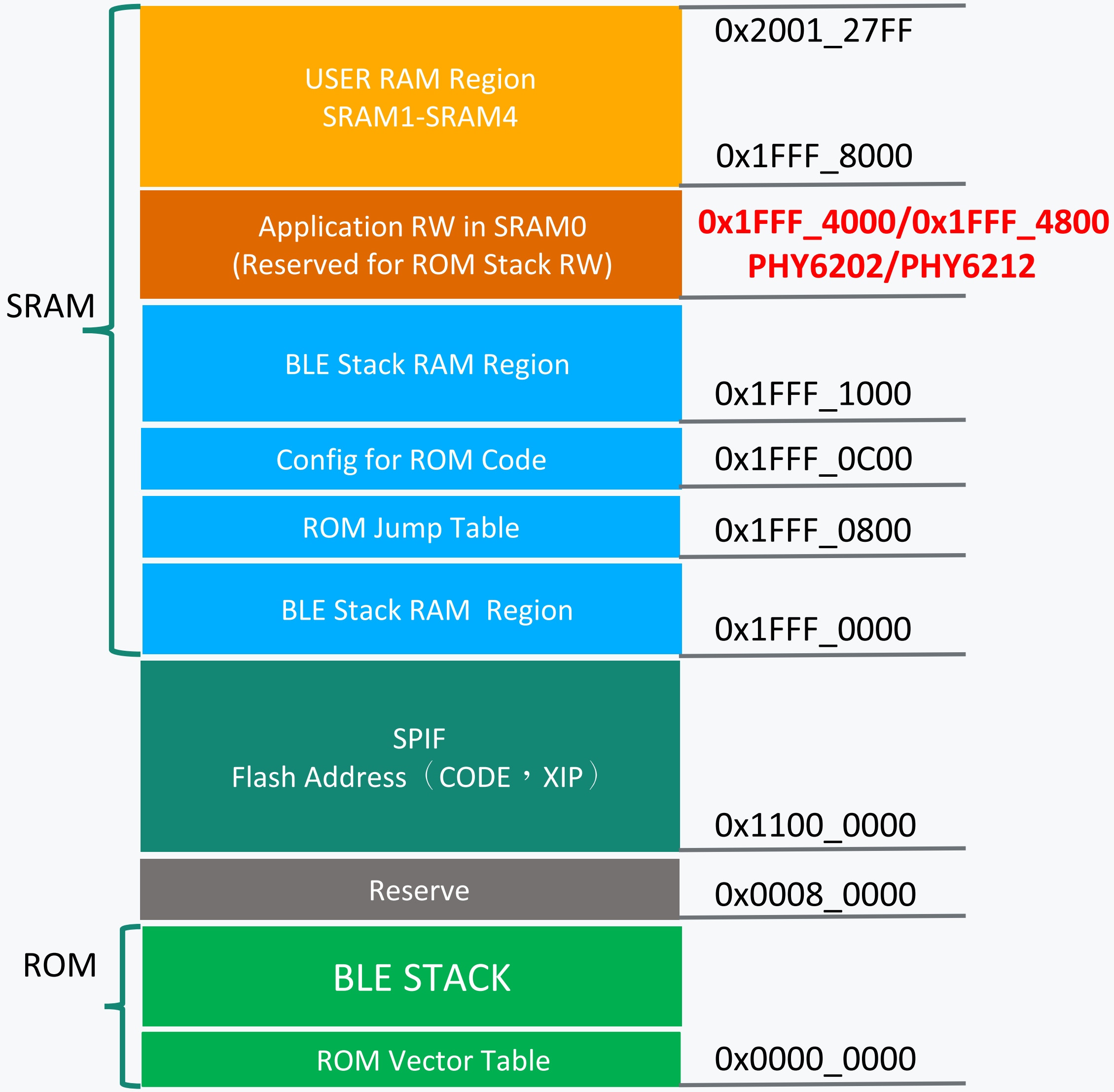
PRBMD0x Software Architecture summary

PRBMD0x Software Architecture



- **SOC**
- MCU :ARM M0 , HCLK16MHz/48MHz
- TIMER:
  - Total 8 24Bit 4MHz Timers， 4 CP Timers for BLE Stack and OS, 4 AP timers for application
  - 32KHz RTC Timer XTAL/RC
- Memory:
  - Total 138KB SRAM， 16KB is reserved for BLE STACK and OS, 122KB for Application.
- **BLE Stack IN ROM**
- All BLE Stack in ROM(LinkLayer +HOST)
- All BLE Stack could be pathed
- **OS**
- NO RTOS using infinite event loop instead

MEMORY MAP



SRAM

USER RAM Region
SRAM1-SRAM4

0x2001_27FF

0x1FFF_8000

Application RW in SRAM0
(Reserved for ROM Stack RW)

0x1FFF_4000/0x1FFF_4800
PHY6202/PHY6212

BLE Stack RAM Region

0x1FFF_1000

Config for ROM Code

0x1FFF_0C00

ROM Jump Table

0x1FFF_0800

BLE Stack RAM Region

0x1FFF_0000

SPIF
Flash Address（CODE，XIP）

0x1100_0000

Reserve

0x0008_0000

ROM

BLE STACK

ROM Vector Table

0x0000_0000

- ROM(128KB)
- FOR ROM BootLoader OS and BLE STACK

- Flash (512KB)
- Support XIP via SPIF

- SRAM(138KB)
- 5 Physical SRAM, all programmable retention in sleep mode
- 0x1FFF_0000-0x1FFF_4000 16KB SRAM in SRAM0 is reserved for BLE STACK and OS RW Regain

	Size	Address Range
SRAM0	32K	1FFF_0000~1FFF_7FFF
SRAM1	32K	1FFF_8000~1FFF_FFFF
SRAM2	64k	2000_0000~2000_FFFF
SRAM3	8K	2001_0000~2001_1FFF
SRAM4	2K	2001_2000~2001_27FF

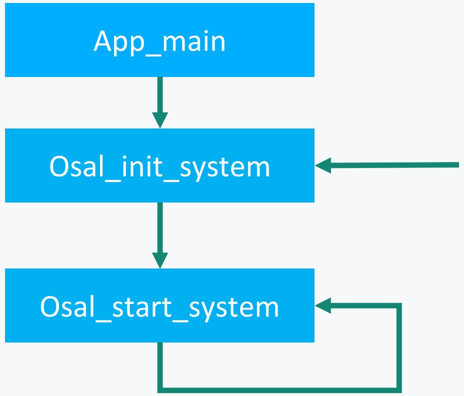
FLASH MAP

Flash Address Offset is 0x1100_0000

Support OTA										
	512K ver (Dual bank)(Has FCT)		512K ver (Single bank)(Has FCT)		512K ver (Dual bank)(No FCT)		512K ver (Single bank)(No FCT)		512K+XIP (Single bank+XIP)(No FCT)	
Reserved By PhyPlus	00000~01fff	8k	00000~01fff	8k	00000~01fff	8k	00000~01fff	8k	00000~01fff	8k
1st Boot info	02000~03fff	8k	02000~03fff	8k	02000~03fff	8k	02000~03fff	8k	02000~03fff	8k
FCDS	04000~04fff	4k	04000~04fff	4k	04000~04fff	4k	04000~04fff	4k	04000~04fff	4k
UCDS	05000~08fff	16k	05000~08fff	16k	05000~08fff	16k	05000~08fff	16k	05000~08fff	16k
2nd Boot info	09000~09fff	4k	09000~09fff	4k	09000~09fff	4k	09000~09fff	4k	09000~09fff	4k
OTA bootloader	0a000~11fff	32k	0a000~11fff	32k	0a000~11fff	32k	0a000~11fff	32k	0a000~11fff	32k
FCT App	12000~2ffff	120k	12000~2ffff	120k		0k		0k		0k
App Bank0	30000~4ffff	128k	30000~4ffff	128k	12000~31fff	128k	12000~31fff	128k	12000~31fff	128k
App Bank1	50000~6ffff	128k		0k	32000~51fff	128k		0k		0k
NVM(or XIP)	70000~7ffff	64k	50000~7ffff	192k	52000~7ffff	184k	32000~7ffff	312k	32000~7ffff(XIP)	312k

Flash Mapping NO OTA			
Function	Address	Size	Description
Reserved	00000~01fff	8k	Reserved For PhyPlus Only
1st Boot info	02000~03fff	8k	Boot Configuration
FCDS	04000~04fff	4k	Factory Configuration Data Storage
UCDS	05000~08fff	16k	User Configuration Data Storage
App Bank	09000~4ffff	284k	Application Code
FCT App(or XIP)	50000~6ffff	128k	Option
NVM	70000~7ffff	64k	Option

OS ABSTRACT LAYER(OSAL)



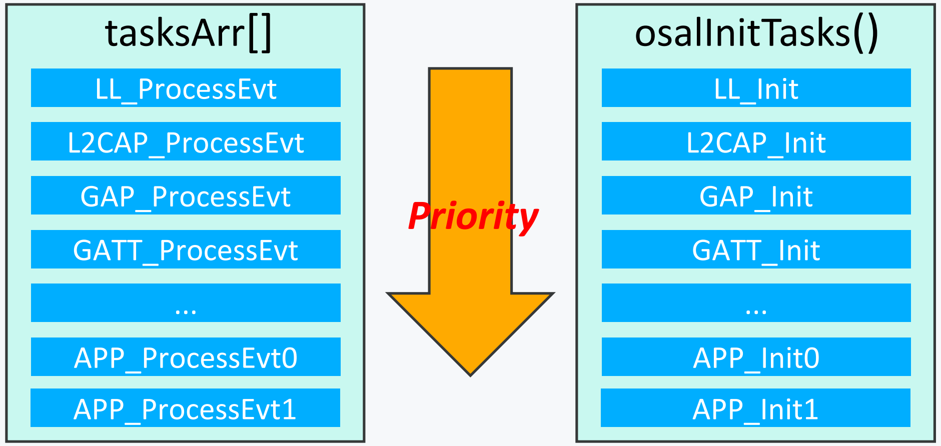
App_main

Osal_init_system

Osal_start_system

Invoke **osalInitTasks**() to initial OSAL tasks

Infinite event process loop, the event process function of each OSAL tasks is stored in array **tasksArr**[]



tasksArr[]

- LL_ProcessEvt
- L2CAP_ProcessEvt
- GAP_ProcessEvt
- GATT_ProcessEvt
- …
- APP_ProcessEvt0
- APP_ProcessEvt1

Priority

osalInitTasks()

- LL_Init
- L2CAP_Init
- GAP_Init
- GATT_Init
- …
- APP_Init0
- APP_Init1

- No RTOS，using infinite event loop instead
- Task process routine could be broken by IRQ
- Task Priority: task in lower index has higher process priority
- Interaction of tasks : could be direct function call, event and message
- Osal task

  the ID of the task is set by osalInitTasks(), start from 0
- Osal Event

  - 16bits event BITMAP for each task

- Event Loop

OSAL EVENT MANAGEMENT

- Osal timer Set Event
- Start timer

☒single ：uint8 osal_start_timerEx(uint8 taskID, uint16 event_id, uint32 timeout_value)

☒Periodic ：uint8 osal_start_reload_timer(uint8 taskID, uint16 event_id, uint32 timeout_value)

timer tick is mini-second. When Timer expires，event BITMAP of the task is set

- Stop Timer

☒uint8 osal_stop_timerEx(uint8 task_id, uint16 event_id)

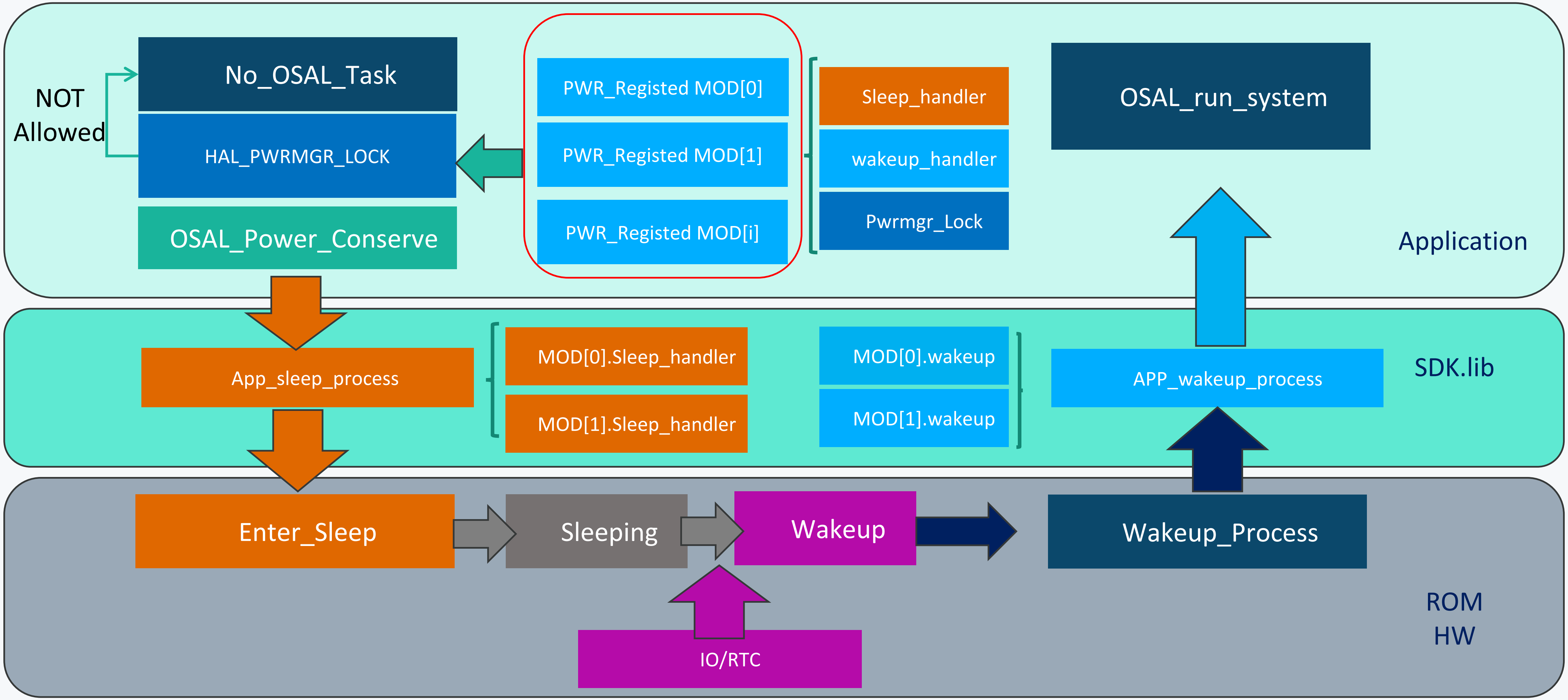
- Osal Message

☒uint8 osal_msg_send(uint8 destination_task, uint8 *msg_ptr)

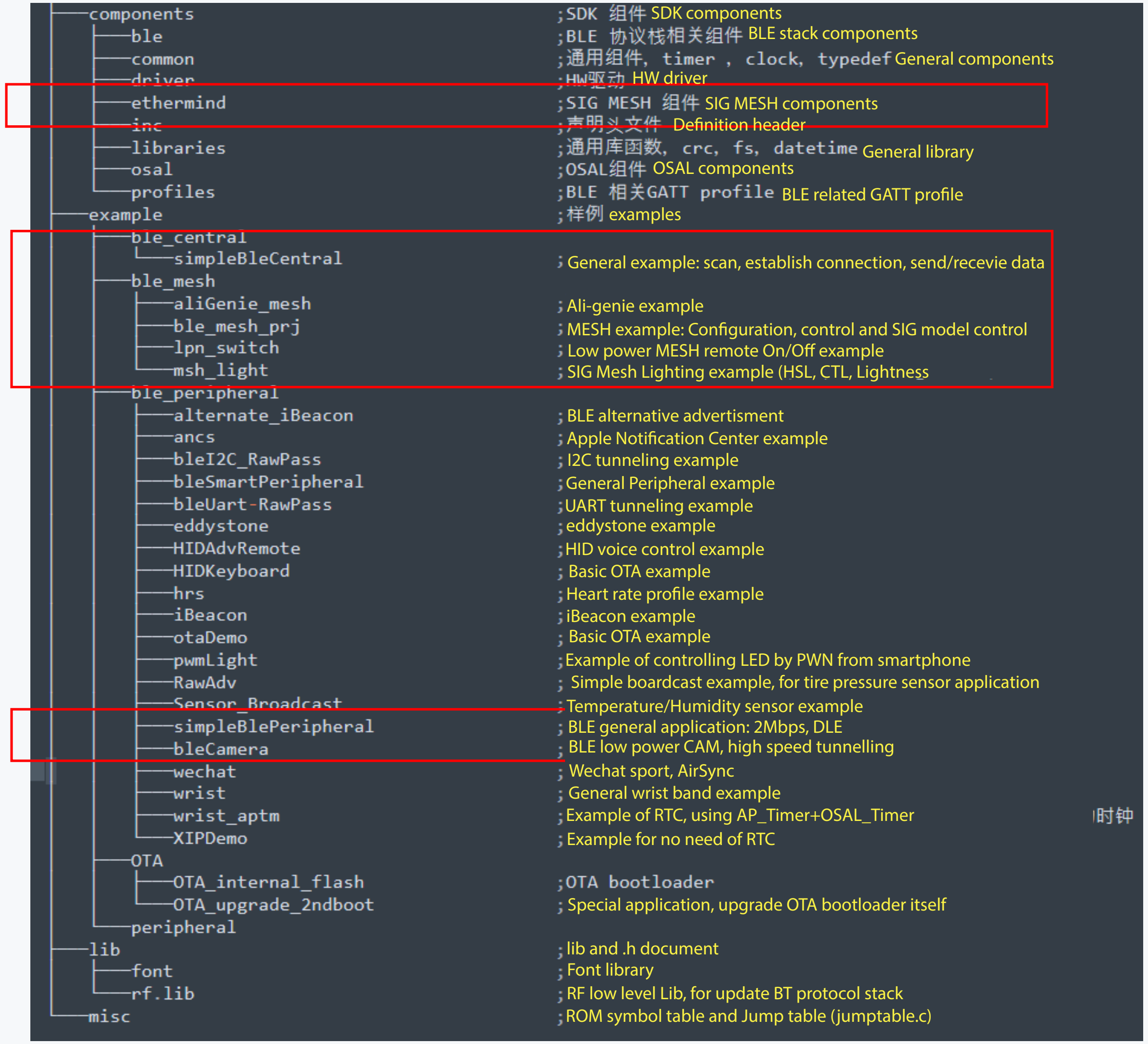
- Osal Set Event

☒uint8 osal_set_event(uint8 task_id, uint16 event_flag)

POWER MANAGEMENT



SDK HIERARCHY -- PHY62XX_SDK_2.X.X



```
├──components                        ;SDK 组件 SDK components
│   ├──ble                           ;BLE 协议栈相关组件 BLE stack components
│   ├──common                        ;通用组件, timer , clock, typedef General components
│   ├──driver                        ;HW驱动 HW driver
│   ├──ethermind                     ;SIG MESH 组件 SIG MESH components
│   ├──inc                           ;声明头文件  Definition header
│   ├──libraries                     ;通用库函数, crc, fs, datetime General library
│   ├──osal                          ;OSAL组件 OSAL components
│   └──profiles                      ;BLE 相关GATT profile BLE related GATT profile
├──example                           ;样例 examples
│   ├──ble_central
│   │   └──simpleBleCentral          ; General example: scan, establish connection, send/recevie data
│   ├──ble_mesh
│   │   ├──aliGenie_mesh             ; Ali-genie example
│   │   ├──ble_mesh_prj              ; MESH example: Configuration, control and SIG model control
│   │   ├──lpn_switch                ; Low power MESH remote On/Off example
│   │   └──msh_light                 ; SIG Mesh Lighting example (HSL, CTL, Lightness
│   ├──ble_peripheral
│   │   ├──alternate_iBeacon         ; BLE alternative advertisment
│   │   ├──ancs                      ; Apple Notification Center example
│   │   ├──bleI2C_RawPass            ; I2C tunneling example
│   │   ├──bleSmartPeripheral        ;General Peripheral example
│   │   ├──bleUart-RawPass           ;UART tunneling example
│   │   ├──eddystone                 ;eddystone example
│   │   ├──HIDAdvRemote              ;HID voice control example
│   │   ├──HIDKeyboard               ; Basic OTA example
│   │   ├──hrs                       ;Heart rate profile example
│   │   ├──iBeacon                   ;iBeacon example
│   │   ├──otaDemo                   ; Basic OTA example
│   │   ├──pwmLight                  ;Example of controlling LED by PWN from smartphone
│   │   ├──RawAdv                    ; Simple boardcast example, for tire pressure sensor application
│   │   ├──Sensor_Broadcast          ;Temperature/Humidity sensor example
│   │   ├──simpleBlePeripheral       ; BLE general application: 2Mbps, DLE
│   │   ├──bleCamera                 ; BLE low power CAM, high speed tunnelling
│   │   ├──wechat                    ; Wechat sport, AirSync
│   │   ├──wrist                     ; General wrist band example
│   │   ├──wrist_aptm                ;Example of RTC, using AP_Timer+OSAL_Timer          时钟
│   │   └──XIPDemo                   ; Example for no need of RTC
│   ├──OTA
│   │   ├──OTA_internal_flash        ;OTA bootloader
│   │   └──OTA_upgrade_2ndboot       ; Special application, upgrade OTA bootloader itself
│   └──peripheral
├──lib                               ; lib and .h document
│   ├──font                          ; Font library
│   └──rf.lib                        ; RF low level Lib, for update BT protocol stack
└──misc                              ;ROM symbol table and Jump table (jumptable.c)
```

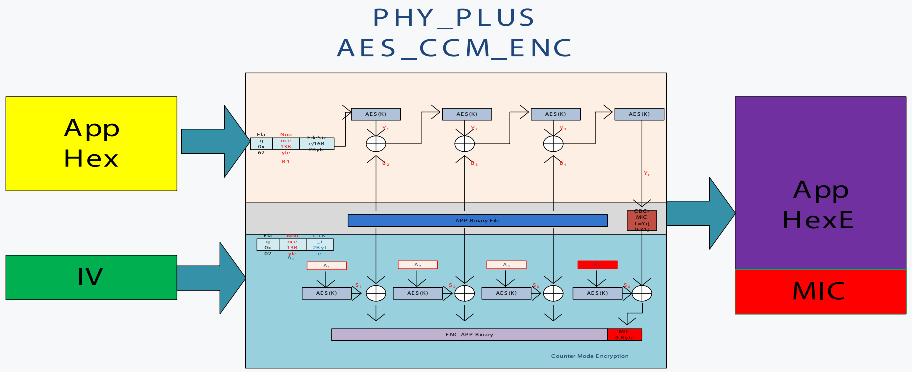
- This SDK is for PRBMD00. PRBMD02 uses SDK3.x and can take this SDK as reference.

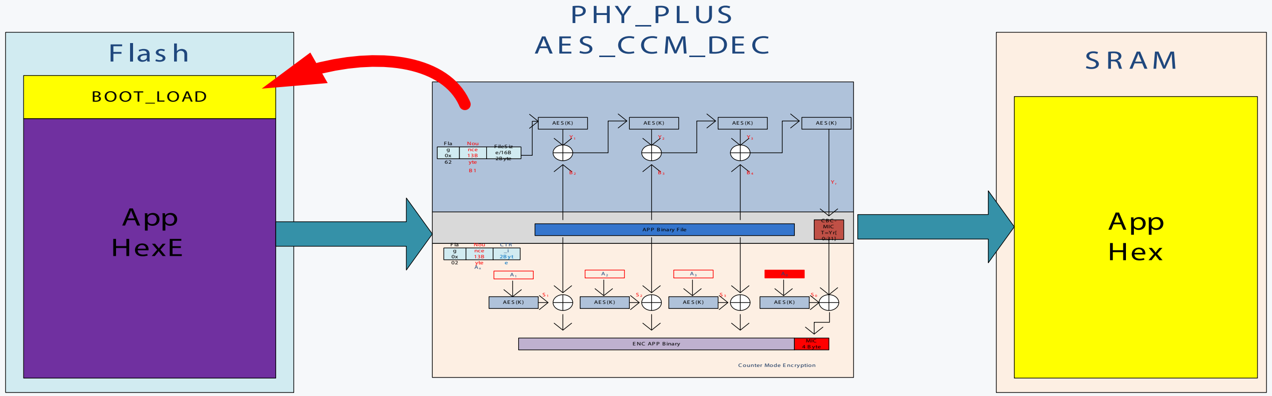
- Application of BLE Master
    - provide high speed OTA mode, upgrade 512KB in 3 sec.

- SIG MESH software module and application:
    - Ali-genie configuration
    - SIG MESH generatl examples (Network configuratiuon, Node configuration, Flow control)
    - SIG MESH low power remote On/Off application
    - SIG MESH light control application, including SIG MESH standard models (HSL, CTL, Lightness, Scene)

- Lower power tunneling application
    - SimpleBlePeripheral example (2Mbps, DLE demo
    - Low power Bluetoot CAM example

SECURITY CODE (IV)





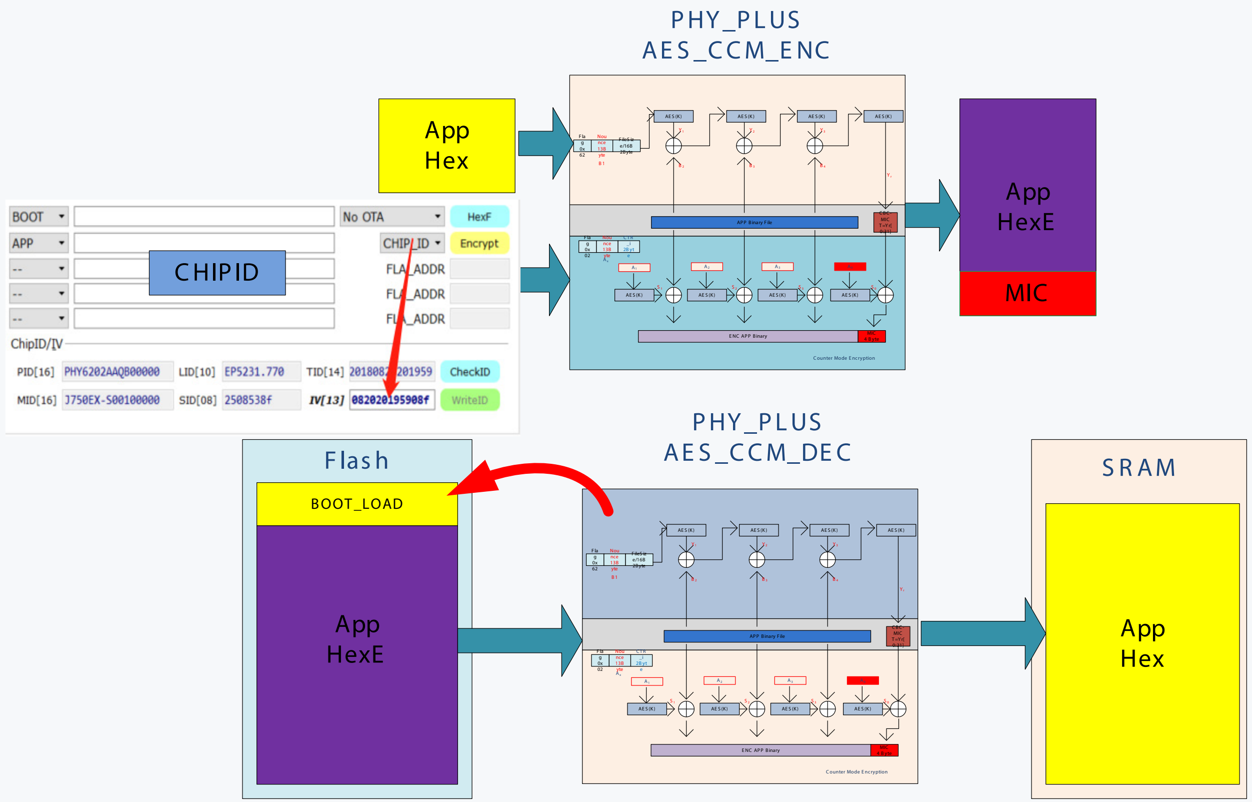
- Encrypt the app using PhyPlus's custom AES_CCM algorithm. Output encrypted AppHexE and MIC for authentication

- Decrypt AppHexE through the AES_CCM decryption algorithm in BOOT_LOAD. Then load the restored App program into the corresponding sram area to start the program.

- IV is the key used for encryption, which can be customized by the customer, with a length of 13Byte

- PhyPlus's custom AES_CCM algorithm uses PHY62XX's AES128 hardware acceleration

SECURITY CODE (CHIP_ID)



BOOT ▾ | No OTA ▾ | HexF

APP ▾ | CHIP_ID ▾ | Encrypt

-- ▾ | FLA_ADDR

-- ▾ | FLA_ADDR

-- ▾ | FLA_ADDR

ChipID/IV

PID[16] PHY6202AAQB00000 LID[10] EP5231.770 TID[14] 2018082[illegible]201959 CheckID

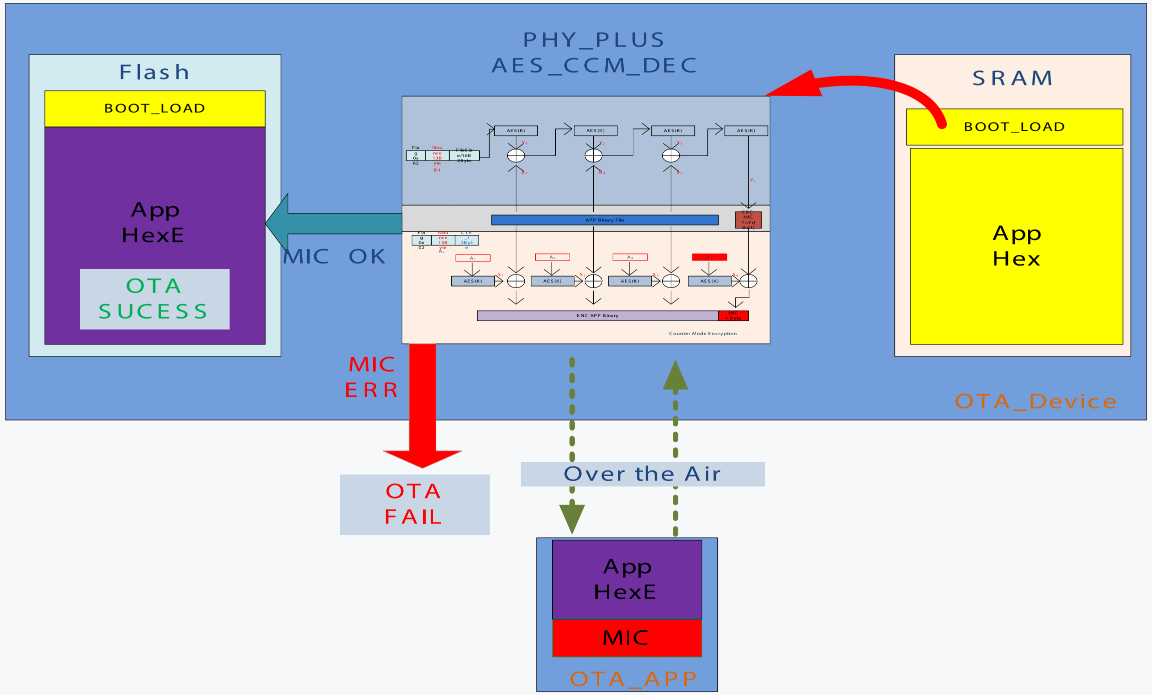
MID[16] J750EX-S00100000 SID[08] 2508538f IV[13] 082020195908f WriteID

- Each PHY62XX chip has a unique ChipID with a total of 64Bytes.

- If ChipID is used to encrypt the App, the IV will be automatically generated by ChipID. The encryption IV of each chip is different.

- For decrypted projects, BOOT_LOAD will first calculate the IV through ChipID, and then use the IV to decrypt

SECURITY OTA



The OTA upgrade process needs to pass the Message Identify Check.

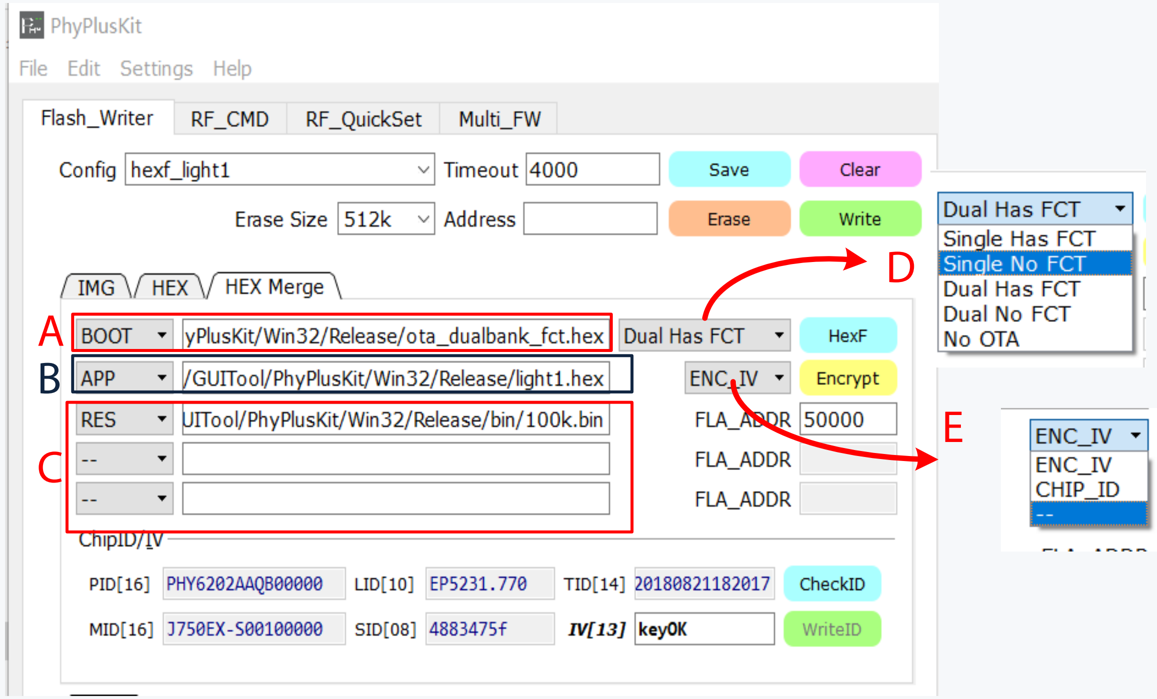
MIC is the 32bit Message Identify code output during the encryption process, which is used to authorize the program downloaded by OTA. Prevent downloading malicious programs.

Only the user's IV is consistent, the MIC test authorization can pass. The AppHexE program will only be written to the flash after the MIC detection is passed.

If the MIC authentication fails, the OTA process fails this time, and the app will not be updated.

If ChipID is used for encryption protection, each OTA_APP needs to obtain the ChipID first, and encrypt the app program with the IV generated by the ChipID.

GENERATE HEXE



Config hexf_light1 Timeout 4000

Erase Size 512k Address

BOOT yPlusKit/Win32/Release/ota_dualbank_fct.hex Dual Has FCT

APP /GUITool/PhyPlusKit/Win32/Release/light1.hex ENC_IV

RES UITool/PhyPlusKit/Win32/Release/bin/100k.bin FLA_ADDR 50000

-- FLA_ADDR

-- FLA_ADDR

PID[16] PHY6202AAQB00000 LID[10] EP5231.770 TID[14] 20180821182017

MID[16] J750EX-S00100000 SID[08] 4883475f IV[13] keyOK

See PhyPlusKit_User_Guide_V2.3 for more details

HEX Merge

A: For inputting the ota_boot.hex file, you need to select a different OTA_BOOT mode through the D selection box. A total of 5 modes are currently supported. Different modes of OTA_BOOT have different mapping of flash address. Refer to the table below for flash mapping

B: It is used to input the hex file of the APP program, and different encryption methods can be selected through the E selection box.

The E:[ENC_IV] mode is to encrypt and protect the app file using the Identify Vector input by IV. The E:[CHIP_ID] mode is to encrypt the app file with the unique chip's chip id. IV=function(chipid.TID,chipid.SID)E:[--] mode, no encryption mode is used.

C: is used to input resource file, currently supports binary format and hex format. You can enter the flash storage address of the resource file in FLA_ADDR.

The [HexF] button is used to generate a *.hexf file with one click, which is generated by combining multiple hex files of A, B, and C, and can be used for direct programming. The output path of the file is the app file directory. The [Encrypt] button is used to generate the encrypted file *.hexe of the app file, and the output path of the file is the app file directory.

ChipID/IV

The CheckID button is used to detect the factory ID of the current chip and needs to be connected to the UART. If the detection result is not [EMPTY], the WriteID button will be activated. For the chip without factory ID programmed, it can be programmed by the WriteID button. Need to fill in the corresponding PID, LID, MID, TID, SID IV: Identify Vector used for the input file, 13Byte. If the encryption method in the figure is selected as CHIP_ID, this part will automatically generate the IV from the CHIP ID.