USER'S GUIDE

Apollo4 Family Bluetooth Low Energy®

Ultra-Low Power Apollo4 Blue SoC Family

A-SOCA4B-UGGA02EN v1.0

Legal Information and Disclaimers

AMBIQ MICRO INTENDS FOR THE CONTENT CONTAINED IN THE DOCUMENT TO BE ACCURATE AND RELIABLE. THIS CONTENT MAY, HOWEVER, CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS OR OTHER MISTAKES. AMBIQ MICRO MAY MAKE CORRECTIONS OR OTHER CHANGES TO THIS CONTENT AT ANY TIME. AMBIQ MICRO AND ITS SUPPLIERS RESERVE THE RIGHT TO MAKE CORRECTIONS, MODIFICATIONS, ENHANCEMENTS, IMPROVEMENTS AND OTHER CHANGES TO ITS PRODUCTS, PROGRAMS AND SERVICES AT ANY TIME OR TO DISCONTINUE ANY PRODUCTS, PROGRAMS, OR SERVICES WITHOUT NOTICE.

THE CONTENT IN THIS DOCUMENT IS PROVIDED "AS IS". AMBIQ MICRO AND ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS CONTENT FOR ANY PURPOSE AND DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS CONTENT, INCLUDING BUT NOT LIMITED TO, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHT.

AMBIQ MICRO DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF AMBIQ MICRO COVERING OR RELATING TO THIS CONTENT OR ANY COMBINATION, MACHINE, OR PROCESS TO WHICH THIS CONTENT RELATE OR WITH WHICH THIS CONTENT MAY BE USED.

USE OF THE INFORMATION IN THIS DOCUMENT MAY REQUIRE A LICENSE FROM A THIRD PARTY UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF THAT THIRD PARTY, OR A LICENSE FROM AMBIQ MICRO UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF AMBIQ MICRO.

INFORMATION IN THIS DOCUMENT IS PROVIDED SOLELY TO ENABLE SYSTEM AND SOFTWARE IMPLEMENTERS TO USE AMBIQ MICRO PRODUCTS. THERE ARE NO EXPRESS OR IMPLIED COPYRIGHT LICENSES GRANTED HEREUNDER TO DESIGN OR FABRICATE ANY INTEGRATED CIRCUITS OR INTEGRATED CIRCUITS BASED ON THE INFORMATION IN THIS DOCUMENT. AMBIQ MICRO RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN. AMBIQ MICRO MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES AMBIQ MICRO ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. "TYPICAL" PARAMETERS WHICH MAY BE PROVIDED IN AMBIQ MICRO DATA SHEETS AND/OR SPECIFICATIONS CAN AND DO VARY IN DIFFERENT APPLICATIONS AND ACTUAL PERFORMANCE MAY VARY OVER TIME. ALL OPERATING PARAMETERS, INCLUDING "TYPICALS" MUST BE VALIDATED FOR EACH CUSTOMER APPLICATION BY CUSTOMER'S TECHNICAL EXPERTS. AMBIQ MICRO DOES NOT CONVEY ANY LICENSE UNDER NEITHER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS. AMBIQ MICRO PRODUCTS ARE NOT DESIGNED, INTENDED, OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE AMBIQ MICRO PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. SHOULD BUYER PURCHASE OR USE AMBIQ MICRO PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD AMBIQ MICRO AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT AMBIQ MICRO WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE PART.

Revision History

Revision	Date	Description
1.0	December 9, 2022	Initial release

Reference Documents

These reference documents can be accessed on the Ambiq Website and/or Content Portal.

Document ID	Description
DS-A4BP-*	Apollo4 Blue Plus SoC Datasheet
DS-A4B-*	Apollo4 Blue SoC Datasheet
n/a	Apollo Blue/Apollo4 Blue Plus SoC 32 MHz Crystal Calibration

* Indicates to use the latest version of document

Table of Contents

List of Tables

List of Figures

SECTION

1

Introduction

The Apollo4 Blue and Apollo4 Blue Plus system on chips (SoCs) have an on-chip Bluetooth Low Energy® controller which provides low-power Bluetooth Low Energy 5.1 connectivity. The purpose of this document is to help the reader understand the Bluetooth Low Energy's operation and the provided functions of the Bluetooth Low Energy module in the AmbiqSuite SDK.

> **NOTE:** Reference to "Apollo4 Blue" applies to either the Apollo4 Blue, Apollo4 Blue Plus SoC or any future "Blue" versions of the Apollo4 family unless stated otherwise.

SECTION

2

Overview

The Bluetooth Low Energy controller includes an Arm® Cortex®-M0, Bluetooth Low Energy baseband, modem and a 2.4 GHz transceiver. Communication with, and control of, the Bluetooth Low Energy controller are implemented through a high-speed SPI interface. Dedicated data movement hardware enables efficient interface for HCI packet transfers.

The Bluetooth Low Energy controller can operate at 32 MHz and it includes a PLL to generate the necessary clocking for the Bluetooth Low Energy subsystem. The reference clock for the PLL can be sourced from either a dedicated external crystal or a single-ended clock input from the Apollo4 SoC. Power regulation is supported internally via a buck DCDC regulator and supporting LDO regulators needed to generate all internal voltages for the radio and digital subsystems.

The following figures show the Bluetooth Low Energy controller subsystems and the Buck-enabled configuration. Figure 2-1 is a high-level block diagram of the Bluetooth Low Energy Controller subsystem, and Figure 2-2 provides a schematic of the BLE Buck circuit required to provide power to the subsystem. Values for the external components can be found in the Electrical Characteristics section of the applicable datasheet.

Figure 2-1: Apollo4 Bluetooth Low Energy Controller Core and Radio Subsystems

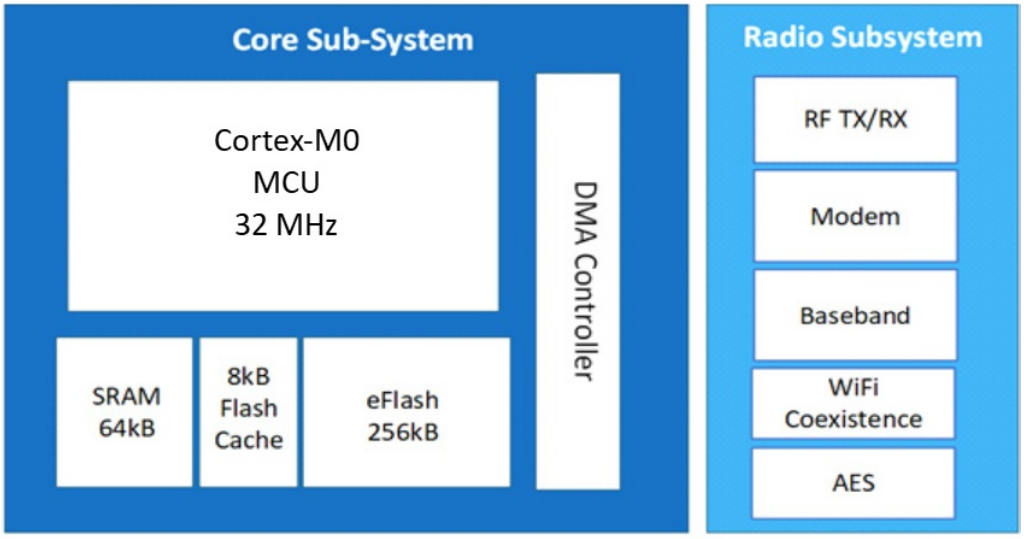
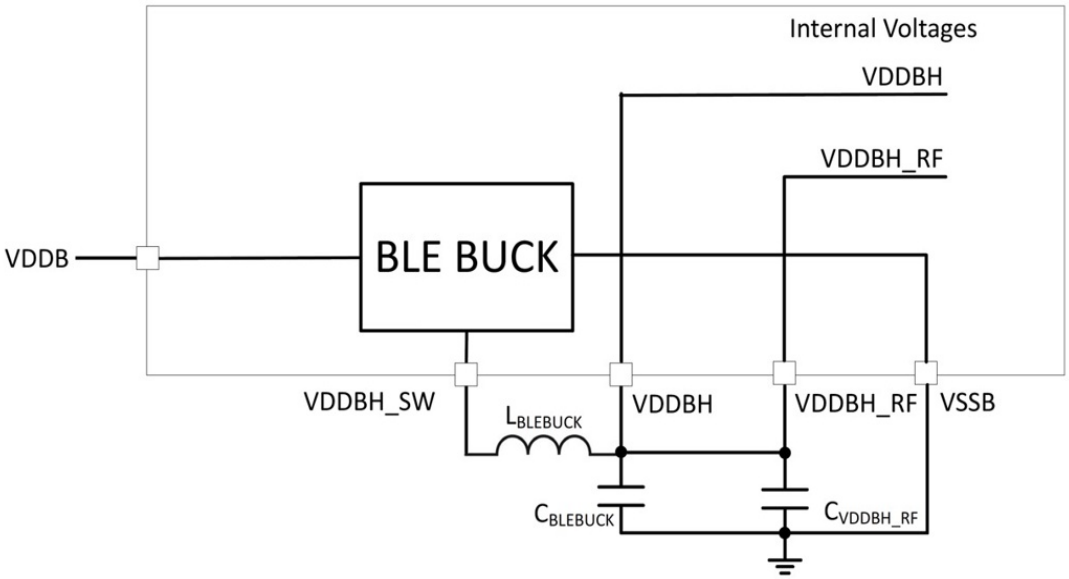


Figure 2-2: Buck-enabled Configuration



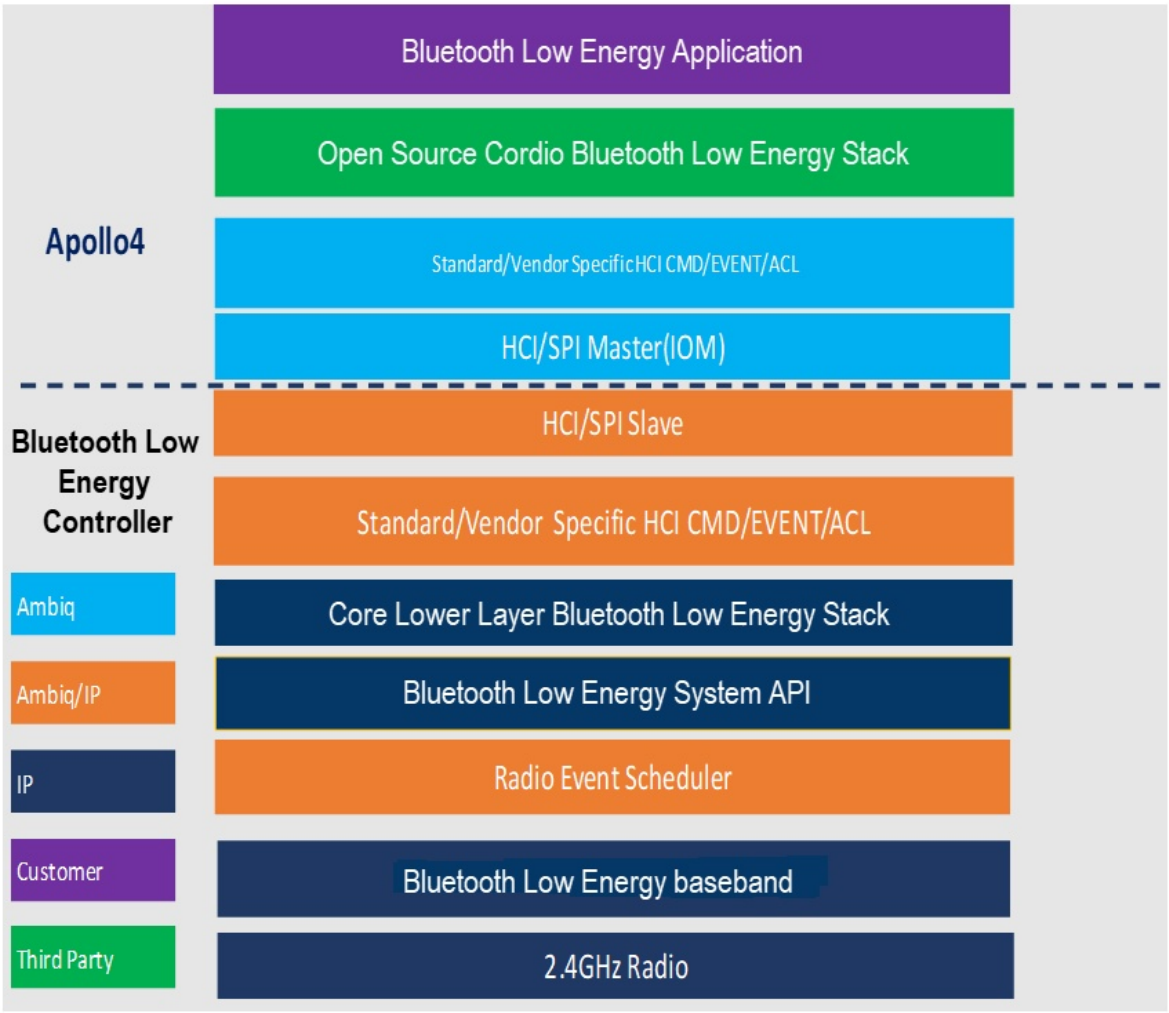
SECTION

3

Bluetooth Low Energy Software Diagram

Figure 3-1 shows the software stack for the Bluetooth Low Energy module.

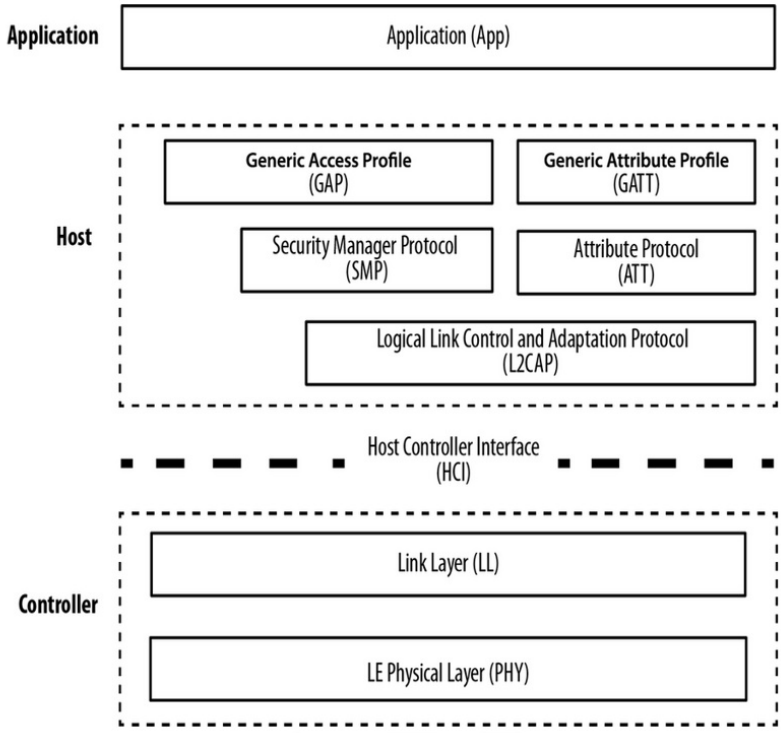
Figure 3-1: Software Components in Bluetooth Low Energy Module



3.1 LE Host Stack

The LE host stack is as shown in the below figure.

Figure 3-2: LE Host Stack



3.2 Bluetooth Low Energy in AmbiqSuite

The Bluetooth Low Energy module configuration and initialization in the AmbiqSuite SDK are as described below.

Configuration:

The Bluetooth Low Energy initialization must be run in one task (RadioTask).

Initialization:

The RadioTask initialization is as follows.

- Boot the radio with HciDrvRadioBoot(1).
  1. Initialize the SPI module, enable crystals such as XTAL32M and XTAL32K for the Bluetooth Low Energy Controller.
  2. Set the default Bluetooth Low Energy TX output power.
  3. Register Bluetooth Low Energy Controller IRQ pin ISR and ClkReq pin ISR.
  4. If it's a cold boot, will use Apollo4's device ID to form Bluetooth address.

- Initialize the main ExactLE stack.
  1. Set up timers for the WSF (Wireless Software Foundation) scheduler with **WsfOsInit()**, and **WsfTimerInit()**.
  2. Initialize a buffer pool for WSF dynamic memory needs.
  3. Initialize the WSF security service with calling **SecInit()**, **SecAesInit()**, **SecCmacInit()**, **SecEccInit()**.
- Set up a callback function for the various layers of the ExactLE stack with **WsfOsSetNextHandler()**, and **xxxHandlerInit()**, such as WSF event handler for HCI(Host Controller Interface), DM(Device Manager), L2C (Logical Link Control) slave, ATT(Attribute Protocol), SMP(Security Manager Protocol), app framework, application and HCI-related events.
- Start the application profile, e.g., xxxStart().
  1. Register for stack callbacks.
  2. Register for app framework callbacks.
  3. Initialize attribute server database.
  4. Set service changed CCCD index.
  5. Set running speed and cadence features.
  6. Reset the device.
- Enter an infinite loop to dispatch WSF OS message.

Detailed initialization of Cordio stack modules is as follows:

- **scheduler_timer_init()** enables a platform-specific (Apollo4 Blue MCU in this case) timer that the WSF timer module uses.
- **WsfTimerInit()** performs WSF timer module initialization.
- **WsfBufInit()** initializes the buffer pool used for messages, data buffer, etc. This should be carefully tuned to meet product-specific requirement.
- Security algorithm module initialization is done by calling the following functions:
  - **SecInit(), SecAesInit(), SecCmacInit(), SecEccInit()**
- Device Manager initialization is done through the following functions:
  - **DmDevVsInit(0) DmAdvInit(), DmScanInit(), DmConnInit(), DmConnMasterInit(), DmConnSlaveInit(), DmSecInit(), DmSecLescInit(), DmPrivInit(), DmHandlerInit()**
- L2CAP module's initialization is done through the following functions:
  - **L2cSlaveHandlerInit(); L2cInit(), L2cSlaveInit(), L2cMasterInit()**
- ATT server and client are initialized through the following functions:
  - **AttHandlerInit(), AttsInit(), AttsIndInit(), AttcInit()**

- Security Manager protocol's initialization is done through the following functions:
  - **SmpHandlerInit(), SmpiInit(), SmpiScInit(), SmprInit(), SmprScInit()**
- The application layer registers certain callbacks for Device Manager, Connection Manager, and ATT-related sub-modules through the following functions: (Note: Look at example for reference.)
  - **DmRegister(watchDmCback), DmConnRegister(DM_CLIENT_ID_APP, watchDmCback), AttRegister(watchAttCback); AttConnRegister(AppServerConnCback), AttsCccRegister();**
- **WsfOsSetNextHandler()** is called with handler function to get a handler ID which can be used by other modules to send a message to its corresponding handler through the WSF scheduler.
- **HciSetMaxRxAclLen()** is called to set the maximum size of an ACL packet that can be reassembled at the HCI layer. The current ACL packet length is 251 bytes which may increase.

Data Length Extension Support:

Data Length Extension, enabled by default for high-speed traffic, requires that a large buffer ((280) in below code snippet) is enabled for receiving/transmitting DLE packet. The **Radio_task.c** (in AmbiSuite SDK any example project source) can be modified to support Data Length Extension (DLE) as in the below code snippet.

```
/***** Important note: the size of g_pui32BufMem should
accommodate both overhead of the internal buffer management
structure, wsfBufPool_t (up to 16 bytes for each pool), and pool
description (e.g. g_psPoolDescriptors below). *****/

// Memory for the buffer pool

static uint32_t g_pui32BufMem [(WSF_BUF_POOLS*16+ 16*8 + 32*4 +
64*6 + 280*8) / sizeof(uint32_t)];

// Default pool descriptor.

static wsfBufPoolDesc_t g_psPoolDescriptors [WSF_BUF_POOLS] =

{

    {16,  8 },

    {32,  4 },

    {64,  6 },

    {280,  8 }

);
```

3.3 Configuring the ATT Maximum Transmission Unit (MTU)

ATT Maximum Transmission Unit (MTU) is the maximum length of an ATT packet. MTU determines the maximum amount of data that can be handled by the transmitter and receiver and held in their buffers.

The ATT PDU defined default values in AmbiqSuite SDK as below:

```
#define ATT_DEFAULT_MTU     23        /*!< \brief Default value of ATT_MTU */
#define ATT_MAX_MTU         517       /*!< \brief Maximum value of ATT_MTU */
```

The MTU size is able to be configured through the ATT layer as shown in the following structure of the application layer using one of the below methods.

Method 1:

Refer to **\third_party\cordio\ble-host\include\att_api.h** files in AmbiqSuite SDK

```
*! \brief ATT run-time configurable parameters */
typedef struct
{
  wsfTimerTicks_t   discIdleTimeout;  /*!< \brief ATT server service

discovery connection idle timeout in seconds */
  uint16_t          mtu;              /*!< \brief desired ATT MTU */
  uint8_t           transTimeout;     /*!< \brief transcation timeout in
                                      seconds */
  uint8_t           numPrepWrites;    /*!< \brief number of queued prepare
                                      writes supported by server */
} attCfg_t;
```

Example code with attCfg_t structure to increase the MTU size:

```
diff --git a/third_party/cordio/ble-profiles/sources/apps/fit/fit_main.c
           b/third_party/cordio/ble-profiles/sources/apps/fit/fit_main.c
--- a/third_party/cordio/ble-profiles/sources/apps/fit/fit_main.c
+++ b/third_party/cordio/ble-profiles/sources/apps/fit/fit_main.c
@@ -122,6 +122,14 @@ static const appUpdateCfg_t fitUpdateCfg = 5
/*! Number of update attempts before giving up */
 };
+static const attCfg_t fitAttCfg =
+{
+  15,                           /* ATT server service discovery connection
                                 idle timeout in seconds */
+  <Target MTU>,                 /* desired ATT MTU, max value is
                                 ATT_MAX_MTU */
+  ATT_MAX_TRANS_TIMEOUT,        /* transcation timeout in seconds */
+  4                             /* number of queued prepare writes
                                 supported by server*/
+};
 /*! heart rate measurement configuration */
 static const hrpsCfg_t fitHrpsCfg =
```

```
 {
@@ -741,6 +749,7 @@ void FitHandlerInit(wsfHandlerId_t handlerId)
   pAppSlaveCfg = (appSlaveCfg_t *) &fitSlaveCfg;
   pAppSecCfg = (appSecCfg_t *) &fitSecCfg;
   pAppUpdateCfg = (appUpdateCfg_t *) &fitUpdateCfg;
+  pAttCfg = (attCfg_t *) &fitAttCfg;
  /* Initialize application framework */
   AppSlaveInit();
```

Method 2:

Change the MTU size to desired value using below function.

```
/*************************************************************************/
/*!
 *  \brief  For internal use only.
 *  \param  connId    DM connection ID.
 *  \param  mtu       Attribute protocol MTU.
 *  \return None.
/*************************************************************************/
void AttcMtuReq(dmConnId_t connId, uint16_t mtu);
```

Application Framework Initialization

The Cordio stack provides a helper module to simplify management of connections of slave and master roles and service discovery of the remote device's service and profile.

Application Layer Configuration

Various data structures are provided to configure project-related logic and behaviors. See below examples in **AmbiqSuite_R4.x.x\third_party\cordio\ble-profiles\sources\apps\watch\watch_main.c**.

```
pAppMasterCfg = (appMasterCfg_t *) &watchMasterCfg;
pAppSlaveCfg = (appSlaveCfg_t *) &watchSlaveCfg;
pAppAdvCfg = (appAdvCfg_t *) &watchAdvCfg;
pAppSecCfg = (appSecCfg_t *) &watchSecCfg;
pAppUpdateCfg = (appUpdateCfg_t *) &watchUpdateCfg;
pAppDiscCfg = (appDiscCfg_t *) &watchDiscCfg;
pAppCfg = (appCfg_t *) &watchAppCfg;
```

SECTION

4

Clocking

4.1 Clock Sources

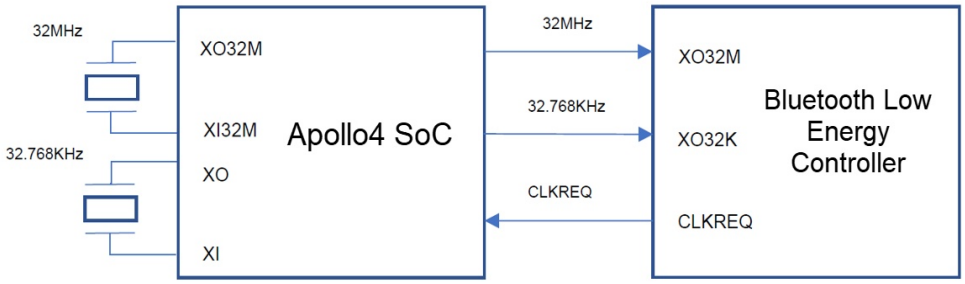
4.1.1 Configure CLKREQ GPIO

The Apollo4 Blue Bluetooth Low Energy Controller supports crystal clock and single-ended clock sources. On the Apollo4 Blue, the clocks are sourced from the MCU to the Bluetooth Low Energy controller. In this configuration, the 32 MHz clock is driven on the XO32M input as a single-ended signal. As well as the 32 kHz clock is driven on the XO32K input as a single-ended signal. This section describes how the XTAL32MHz clock is sourced from Apollo4 to the Blue Bluetooth Low Energy controller.

The Apollo4 Blue has inputs for both 32 MHz and 32 kHz crystals. The 32 MHz crystal is connected to the XO32M / XI32M pins, while the 32 kHz crystal is connected to the XO/XI pins. The clocking configuration must be set in the **MCUCTRL_XTALHSCTRL** register accordingly.

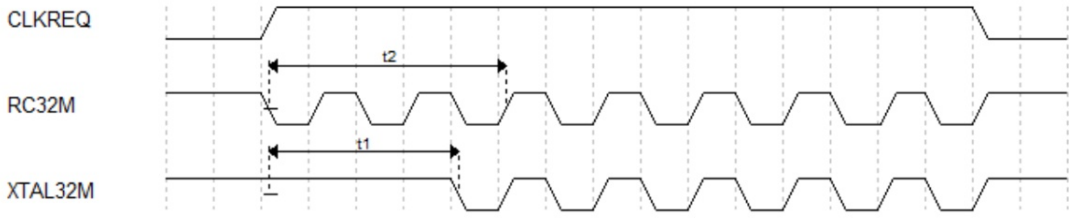
The CLKREQ (GPIO) signal is used to assert clock requests to the SoC. This allows the SoC to power down the 32 MHz crystal to save power. The 32 kHz is always on after the Bluetooth Low Energy controller in the Apollo4 Blue is initialized and turned on.

Figure 4-1: Configuring CLKREQ GPIO



The handshake of XTAL32MHz is as described below.

Figure 4-2: Wake Event Sequence



On a "wake" event the sequence is as follows:

1. The Bluetooth Low Energy controller utilizes an internal 32 MHz RC oscillator signal (RC32M) to assert CLKREQ.
2. The Apollo4 SoC initiates XTAL32M startup after receiving the CLKREQ interrupt.
3. XTAL32MHz is stable after 't1' delay time and available to output to XO32M of the BLE controller.
4. The Bluetooth Low Energy controller switches to use XTAL32M after asserting CLKREQ for 't2' delay.

Note:

> **NOTE:** 't1' delay must be shorter than 't2' delay, meaning that the XTAL32M must become stable before the Bluetooth Low Energy controller switches to use XTAL32M from RC32M.

On "sleep" event:

1. The Bluetooth Low Energy controller switches to use the low-frequency clock and de-asserts CLKREQ.
2. The SoC gates XTAL32M and optionally powers down XTAL32M.

> **NOTES:**
>
> - The XTAL32MHz is powered down by default when CLKREQ de-assertion is detected. XTAL32MHz should be kept on if other modules of Apollo4 SoC are using it and powered down when not used.
> - To determine whether the 32 MHz crystal needs to be trimmed and how to perform the trim, refer to the knowledge base article *Apollo4 Blue / Apollo4 Blue Plus SoC 32 MHz Crystal Calibration*.

4.2 Clock Configuration

4.2.1 Configure CLKREQ GPIO

Initialize the GPIO configuration of CLKREQ pin and enable it in **am_devices_cooper_pins_enable()**.

> **NOTE:** Different Apollo4 series devices use different GPIOs for CLKREQ.

```
am_hal_gpio_pincfg_t g_AM_DEVICES_COOPER_CLKREQ =
{
   .GP.cfg_b.uFuncSel             = AM_HAL_PIN_40_GPIO,
   .GP.cfg_b.eGPInput             = AM_HAL_GPIO_PIN_INPUT_ENABLE,
   .GP.cfg_b.eGPRdZero            = AM_HAL_GPIO_PIN_RDZERO_READPIN,
   .GP.cfg_b.eIntDir              = AM_HAL_GPIO_PIN_INTDIR_LO2HI,
   .GP.cfg_b.eGPOutCfg            = AM_HAL_GPIO_PIN_OUTCFG_DISABLE,
   .GP.cfg_b.eDriveStrength       = AM_HAL_GPIO_PIN_DRIVESTRENGTH_12MA,
   .GP.cfg_b.uSlewRate            = 0,
   .GP.cfg_b.ePullup              = AM_HAL_GPIO_PIN_PULLUP_NONE,
   .GP.cfg_b.uNCE                 = 0,
   .GP.cfg_b.eCEpol               = AM_HAL_GPIO_PIN_CEPOL_ACTIVELOW,
   .GP.cfg_b.uRsvd_0              = 0,
   .GP.cfg_b.ePowerSw             = AM_HAL_GPIO_PIN_POWERSW_NONE,
   .GP.cfg_b.eForceInputEn        = AM_HAL_GPIO_PIN_FORCEEN_NONE,
   .GP.cfg_b.eForceOutputEn       = AM_HAL_GPIO_PIN_FORCEEN_NONE,
   .GP.cfg_b.uRsvd_1              = 0,
};

   void am_devices_cooper_pins_enable(void)
   {
       am_hal_gpio_pinconfig(AM_DEVICES_COOPER_CLKREQ_PIN,
                       g_AM_DEVICES_COOPER_CLKREQ);
…
   }
```

4.2.2 4.2.2 Initialize CLKREQ Interrupt Service

Initialize the CLKREQ interrupt and corresponding service handler in **HciDrvRadioBoot()**.

```
uint32_t HciDrvRadioBoot(bool bColdBoot)
{
     …
     uint32_t IntNum = AM_DEVICES_COOPER_CLKREQ_PIN;
     am_hal_gpio_interrupt_register(AM_HAL_GPIO_INT_CHANNEL_0, IntNum,
                                    ClkReqIntService, NULL);
     am_hal_gpio_interrupt_control(AM_HAL_GPIO_INT_CHANNEL_0,
                         AM_HAL_GPIO_INT_CTRL_INDV_ENABLE, (void *)&IntNum);
     …
}

static void ClkReqIntService(void *pArg)
{
     if(am_devices_cooper_clkreq_read(g_IomDevHdl))
     {
        // Power up the 32MHz Crystal
        am_hal_mcuctrl_control(AM_HAL_MCUCTRL_CONTROL_EXTCLK32M_KICK_START,
                               0);
     }
     else
     {
        am_hal_mcuctrl_control(AM_HAL_MCUCTRL_CONTROL_EXTCLK32M_DISABLE, 0);
     }
     am_hal_gpio_intdir_toggle(AM_DEVICES_COOPER_CLKREQ_PIN);
}
```

4.2.3 Initialize XTAL32MHz Startup

am_hal_mcuctrl_control() is used to enable and disable the 32MHz crystal. Modify the trim codes for CAP1/CAP2 by setting the **MCUCTRL_XTALHSTRIMS_XTALHSCAPTRIM** and **MCUCTRL_XTALHSTRIMS_XTALHSCAP2TRIM** fields of the **MCUCTRL_XTALHSTRIMS** register based on the specific XTAL32M model on your board in the case of **AM_HAL_MCUCTRL_CONTROL_EXTCLK32M_KICK_START, AM_HAL_MCUCTRL_CONTROL_EXTCLK32M_DISABLE** and **AM_HAL_MCUCTRL_CONTROL_EXTCLK32M_NORMAL**.

```
uint32_t am_hal_mcuctrl_control(am_hal_mcuctrl_control_e eControl, void
*pArgs)
{
      volatile uint32_t ui32Reg;
      switch ( eControl )
      {
          …
          case AM_HAL_MCUCTRL_CONTROL_EXTCLK32M_KICK_START:
          // Set the specific trim code for CAP1/CAP2, it impacts frequency
accuracy and should be retrimmed
           ui32Reg = _VAL2FLD(MCUCTRL_XTALHSTRIMS_XTALHSCAP2TRIM, 44)      |
                     _VAL2FLD(MCUCTRL_XTALHSTRIMS_XTALHSCAPTRIM, 4)        |
         // Set the transconductance of crystal to maximum, it accelerates
the startup sequence
                     _VAL2FLD(MCUCTRL_XTALHSTRIMS_XTALHSDRIVETRIM, 3)      |
         // Choose the power of clock driver to be the cleanest one
                     _VAL2FLD(MCUCTRL_XTALHSTRIMS_XTALHSDRIVERSTRENGTH, 0) |
         // Tune the bias generator
                     _VAL2FLD(MCUCTRL_XTALHSTRIMS_XTALHSIBIASCOMP2TRIM, 3) |
                     _VAL2FLD(MCUCTRL_XTALHSTRIMS_XTALHSIBIASCOMPTRIM, 15) |
         // Set the bias of crystal to maximum
                     _VAL2FLD(MCUCTRL_XTALHSTRIMS_XTALHSIBIASTRIM, 127)    |
                     _VAL2FLD(MCUCTRL_XTALHSTRIMS_XTALHSRSTRIM, 0)         |
                     _VAL2FLD(MCUCTRL_XTALHSTRIMS_XTALHSSPARE, 0);
              MCUCTRL->XTALHSTRIMS = ui32Reg;
        …
            break;
    …
  }
}
```

am_devices_cooper_crystal_trim_set() also may be used to set the CAP1/CAP2 to test the 32 MHz crystal frequency on your board to find a suitable values for good accuracy.

4.2.4 Wakeup Time Configuration

For the "t1" delay mentioned in *Section 3.1 LE Host Stack on page 11*, it means that the XTAL32MHz needs some time to startup and become available to provide the clock to the Bluetooth Low Energy Controller to work. Generally, it needs at least 750 µs.

For the "t2" delay mentioned in *Section 3.1 LE Host Stack on page 11* it means that the Bluetooth Low Energy Controller waits for one fixed setting time after asserting CLEREQ then switches to use the XTAL32MHz from RC 32M. If "t1" is longer than "t2", the Bluetooth Low Energy Controller will enter an unknown state You can set the "t2" by modifying the E**XT_WAKEUP_TIME_VALUE and OSC_WAKEUP_-TIME_VALUE** in **am_devices_cooper.h**.

> **NOTE: EXT_WAKEUP_TIME_VALUE** determines the time before switching to XTAL32M from RC32M when the Bluetooth Low Energy Controller is awakened by an external signal, while **OSC_WAKEUP_TIME_VALUE** determines the time before switching to XTAL32M from RC32M when the Bluetooth Low Energy Controller is awakened by its internal timer. The Bluetooth Low Energy Controller may not know the wakeup source in the next wakeup instant so it will choose the greater of these two parameters to determine the time which is called 't2' delay in the context. These two parameters are always set to be the same value and that value is written to Bluetooth Low Energy Controller NVDS field. The default value is 1000 µs.

```
#ifndef EXT_WAKEUP_TIME_VALUE
#define EXT_WAKEUP_TIME_VALUE           1000 // microsecond
#endif
#ifndef OSC_WAKEUP_TIME_VALUE
#define OSC_WAKEUP_TIME_VALUE           1000 // microsecond
#endif
```

The Apollo4 SoC needs to execute the **ClkReqIntService()** function within "t2-t1" time after receiving the CLKREQ interrupt. For complex systems, there may be other GPIO interrupts in the same GPIO group with CLKREQ which may block the executing of **ClkReqIntService()**. The higher "t2" delay needs to be set to make the Bluetooth Low Energy Controller wait for more time.

> **NOTE:** The higher wakeup time makes the Bluetooth Low Energy Controller wake up earlier when waiting for the XTAL32M and may result in higher power consumption.

SECTION

5

32 MHz Crystal Calibration

See knowledgebase article *Apollo4 Blue / Apollo4 Blue Plus SoC 32 MHz Crystal Calibration* in the Ambiq Support Center for information about how to determine if trimming of the 32 MHz crystal is needed and instructions for trimming the crystal frequency.

SECTION

6

Vendor-Specific HCI Commands for BLE Controller

The overall vendor-specific HCI commands supported are shown in Table 6-1.

Table 6-1: Overall Vendor-Specific HCI Command

HCI Command	OGF	OCF	Opcode
Set Transmit Power Level	0x3F	0x0070	0xFC70
Set Log Bit Map	0x3F	0x0073	0xFC73
Set Bluetooth Address	0x3F	0x0074	0xFC74
Update Bluetooth Low Energy Firmware	0x3F	0x0075	0xFC75
Read Register Value	0x3F	0x0039	0xFC39
Write Register Value	0x3F	0x003A	0xFC3A
Get Device ID	0x3F	0x0076	0xFC76
Set NVDS Parameter	0x3F	0x0077	0xFC77
Set Link Layer Feature	0x3F	0x0078	0xFC78

6.1 HciVscSetRfPowerLevelEx

This command is used to configure the radio transmit power level during normal mode or test mode.

Table 6-2: HciVscSetRfPowerLevelEx Parameter Description

Parameter	Size (in bytes)	Description	Opcode
Transmit Power level	1	Transmit output power levels -20dBm/-15dBm/-10dBm/-5dBm/0dBm/4dBm/6dBm, using value 0x00/0x01/0x02/0x03/0x04/0x05/0x06 as the parameter value.	0xFC70
Return Parameters			
Status	1	Standard BT error code	

Example: Set transmit output power level to 4dBm:
Command: 01 70 FC 01 **05**
Event: 04 0E 04 05 70 FC 00

> **NOTE:** The **05** means the output power level is 4dBm as in the description section.

Set Bluetooth Low Energy Transmit Power function:

The following API / code snippet used to set the TX power through the main application with the immediate call of the **HCIDrvRadioBoot** function.

```
\boards\apollo4b_blue_evb\examples\ble\ble_freertos_fit\src\radio_task.c
void
RadioTask(void *pvParameters)
{
#if WSF_TRACE_ENABLED == TRUE
    //
    // Enable ITM
    //
    am_util_debug_printf("Starting wicentric trace:\n\n");
#endif
    // Boot the radio.
    HciDrvRadioBoot(1);

  + // Set the default BLE TX Output power.
  +  am_util_ble_tx_power_set(g_IomDevHdl, TX_POWER_LEVEL_DEFAULT);

    // Initialize the main ExactLE stack.
    //
    exactle_stack_init();
    }
```

The Tx power level in dBm is defined as below in the AmbiqSuite SDK in this file:

```
 \third_party\cordio\ble- host\sources\hci\ambiq\cooper\hci_drv_cooper.h.
//*****************************************************************************
// AMBIQ vendor specific events
//*****************************************************************************
// Tx power level in dBm.
typedef enum
{
    TX_POWER_LEVEL_MINUS_20P0_dBm,
    TX_POWER_LEVEL_MINUS_15P0_dBm,
    TX_POWER_LEVEL_MINUS_10P0_dBm,
    TX_POWER_LEVEL_MINUS_5P0_dBm,
    TX_POWER_LEVEL_0P0_dBm,
    TX_POWER_LEVEL_PLUS_3P0_dBm,
    TX_POWER_LEVEL_PLUS_4P0_dBm,
    TX_POWER_LEVEL_PLUS_6P0_dBm,
    TX_POWER_LEVEL_INVALID,
}txPowerLevel_t;

// Set the default BLE TX Output power to +0dBm.
#define TX_POWER_LEVEL_DEFAULT TX_POWER_LEVEL_0P0_dBm
```

6.2 HciVscSetTraceBitMap

This command is used to enable the specified logging bitmap in Bluetooth Low Energy controller to output corresponding logging information to HOST.

Table 6-3: HciVscSetTraceBitMap Parameter Description

Parameter	Size (in bytes)	Description	Opcode
Logging bitmap	4	Bitmap configuration for interested traces from controller to host.	0xFC73
Return Parameters			
Status	1	Standard BT error code	

6.3 HciVscUpdateFw

This command is used to indicate Cooper which type of firmware to update.

Table 6-4: HciVscUpdateFw Parameter Description

Parameter	Size (in bytes)	Description	Opcode
Logging bitmap	4	Bitmap configuration for interested traces from controller to host.	0xFC73
Return Parameters			
Status	1	Standard BT error code	

Example:

Command:01 75 FC 04 0D 5A B7 38
Event:04 0E 04 05 75 FC 00

6.4 HciVscReadReg

This command is used to read the value of a specified register from Cooper.

Table 6-5: HciVscReadReg Parameter Description

Parameter	Size (in bytes)	Description	Opcode
Register address	4	4 bytes of register address	0xFC39
Return Parameters			
Status	1	Standard BT error code	
Register address	4	Register address in little endian format	
Register value	4	Register value in little endian format	

Example: Read the value of register address 0x45C00018
Command: 01 39 FC **04 18 00 C0 45**
Event: 04 0E 0C 05 39 FC **00 18 00 C0 45 40 3B 09 C8**

NOTES:

- For the command and event, the data applies little endian format.
- For the event, the **00** status means success, the 4 bytes **18 00 c0 45** is the register address, and the 4 bytes **40 3B 09 C8** is the value.

6.5 HciVscWriteReg

This command is used to write a value to a specified register.

Table 6-6: HciVscWriteReg Parameter Description

Parameter	Size (in bytes)	Description	Opcode
Register address	4	4 bytes of register address	0xFC3A
Value to set	4	4 bytes of value to set	
Return Parameters			
Status	1	Standard BT error code	
Register address	4	Register address in little endian format	

Example: Write value 0xC8093B45 to register 0x45C00018
Command: 01 3A FC 08 **18 00 c0 45 40 3B 09 c8**
Event: 04 0E 08 05 3A FC **00 18 00 C0 45**

NOTES:

- For the command and event, the data applies little endian format.
- For the command, the 4 bytes **18 00 c0 45** is the register address, and the 4 bytes **40 3B 09 C8** is the value.
- For the event, the **00** status means success, the 4 bytes **18 00 c0 45** is the register address.

6.6 HciVscGetDeviceId

This command is used to get Bluetooth Low Energy chip ID.

Table 6-7: HciVscGetDeviceId Parameter Description

Parameter	Size (in bytes)	Description	Opcode
NULL	0		0xFC76
Return Parameters			
Status	1	Standard BT error code	
Register address	4	Chip ID in little endian format	

Example:
Command: 01 76 FC 00
Event: 04 0E 0C 05 76 FC 00 **04 17 61 64 71 F4 30 50**

6.7 HciVscUpdateNvdsParam

This command is used to change NVDS parameter according to actual implementation.

Table 6-8: HciVscUpdateNvdsParam Parameter Description

Parameter	Size (in bytes)	Description	Opcode
NVDS parameter	240	240 is the maximum length of NVDS parameters.	0xFC77
Return Parameters			
Status	1	Standard BT error code	

NOTE: Refer to **nvds_data** array in **hci_drv_cooper.c,** you can add the parameters to be changed.

6.8 HciVscUpdateLinklayerFeature

This command is used to change Link Layer supported features according to actual implementation.

Table 6-9: HciVscUpdateLinklayerFeature Parameter Description

Parameter	Size (in bytes)	Description	Opcode
Link Layer features	8	The default length for link layer features	0xFC78
Return Parameters			
Status	1	Standard BT error code	

SECTION

7

Bluetooth Low Energy MAC Address

The Bluetooth address (or) Bluetooth MAC address is a 48-bit value that uniquely identifies a Bluetooth device. In the Bluetooth specification, it is referred to as **BD_ADDR**.

In the HCI driver initialization, the Bluetooth device address is created with Apollo4's device ID and it is programmed into the Bluetooth Low Energy controller through the HCI VS command during startup of the Cordio stack. See the example code below:

```
/*****************************************************************************/
Refer hci_drv_cooper.c and radio_task.c files in AmbiqSuite SDK
/*****************************************************************************/
HciDrvRadioBoot(bool bColdBoot)
{
    // When it's bColdBoot, it will use Apollo's Device ID to form Bluetooth address.
    if (bColdBoot)
    {
        am_hal_mcuctrl_device_t sDevice;
        am_hal_mcuctrl_info_get(AM_HAL_MCUCTRL_INFO_DEVICEID, &sDevice);
        // Bluetooth address formed by ChipID1 (32 bits) and ChipID0 (8-23 bits).
        memcpy(g_BLEMacAddress, &sDevice.ui32ChipID1, sizeof(sDevice.ui32ChipID1));
        // ui32ChipID0 bit 8-31 is test time during chip manufacturing
        g_BLEMacAddress[4] = (sDevice.ui32ChipID0 >> 8) & 0xFF;
        g_BLEMacAddress[5] = (sDevice.ui32ChipID0 >> 16) & 0xFF;
    }
    return AM_DEVICES_COOPER_STATUS_SUCCESS;
   }
```

Public Address:

A public address is a global fixed address which must be purchased from IEEE. Ambiq does not ship Apollo4 family SoCs with public addresses preprogrammed.

The following API is used to set the custom BD_ADDR. In AmbiqSuite SDK R4.x releases, uncomment the following code snippets. Using the project **ble_freertos_fit** as an example, and modify the address as shown below to apply.

```
\boards\apollo4b_blue_evb\examples\ble\ble_freertos_fit\src\radio_task.c

void RadioTask(void *pvParameters)
{
exactle_stack_init();
   // uncomment the following to set custom Bluetooth address here
-    // {
-    //     uint8_t bd_addr[6] = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66};
-    //     HciVscSetCustom_BDAddr(&bd_addr);
-    // }
+    {
+        uint8_t bd_addr[6] = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66};
+        HciVscSetCustom_BDAddr(&bd_addr[0]);
       }
```

Random Static Address:

To use a random static address, apply the following code snippet to the application in development. Use the project **ble_freertos_fit** as an example. The macro **DM_RAND_ADDR_SET** is called to make sure the address to be set follows the format defined by the Bluetooth specification.

Make sure the address is configured before any air activity executions, advertising, scanning, etc.

```
\third_party\cordio\ble-profiles\sources\apps\fit\fit_main.c
static void fitSetup(fitMsg_t *pMsg)
{
    AppAdvSetData(APP_ADV_DATA_CONNECTABLE, sizeof(fitAdvDataDisc), (uint8_t *) fitAdv-
DataDisc);
    AppAdvSetData(APP_SCAN_DATA_CONNECTABLE, sizeof(fitScanDataDisc), (uint8_t *) fitS-
canDataDisc);

+  uint8_t ui8Addr[BDA_ADDR_LEN] = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66};
+
+  DM_RAND_ADDR_SET(ui8Addr, DM_RAND_ADDR_STATIC);
+  DmDevSetRandAddr(ui8Addr);
+  DmAdvSetAddrType(DM_ADDR_RANDOM);

   /* start advertising; automatically set connectable/discoverable mode and bondable
mode */
   AppAdvStart(APP_MODE_AUTO_INIT);
}
```

SECTION

8

Enabling the BLE Resolvable Private Address Resolution

A Resolvable Private Address (RPA) is an address that's generated using a random number and the secret Identity Resolving Key (IRK). It is used to prevent malicious third-parties from tracking a Bluetooth device and allowing one or more trusted parties from identifying the Bluetooth device of interest.

A Resolvable Random Private address is "resolvable" using a key shared with a trusted device. This key is referred to as the IRK (Identity Resolving Key). This IRK is shared between two devices at the time of pairing and stored in the device's internal memory during bonding.

Along with IRK, the devices share a fixed address called the Identity Address. A Resolvable Private address contains following fields (little-endian format):

Hash (24 bits)	**Prand (22 bits)**	**1**	**0**

- The "prand" is a 24-bit number that has 22 random bits and 0 and 1 are fixed in the most significant bits (MSB)
- The lower 24-bits represent a hash value which is generated using the "prand" and the IRK

The AmbiqSuite **ble_freertos_fit** example enables the RPA feature demo with the following code added into **fit_main.c** source file (**\third_party\cordio\ble-profiles\sources\apps\fit\fit_main.c**).

The code defined in the macro **PRIVACY_RPA_FEATURE_ENABLE** as shown below.

```
\third_party\cordio\ble-profiles\sources\apps\fit\fit_main.c
/**************************************************************************************
   Macros
**************************************************************************************/
-
+#define PRIVACY_RPA_FEATURE_ENABLE
 /*! WSF message event starting value */
 #define FIT_MSG_START               0xA0

@@ -65,6 +65,9 @@ enum
   FIT_HR_TIMER_IND = FIT_MSG_START,    /*! Heart rate measurement timer expired */
   FIT_BATT_TIMER_IND,                  /*! Battery measurement timer expired */
   FIT_RUNNING_TIMER_IND               /*! Running speed and cadence measurement timer
expired */
+#ifdef PRIVACY_RPA_FEATURE_ENABLE
+  ,FIT_RPA_ADDR_READ_TIMER_IND
+#endif
 };

/**************************************************************************************
@@ -105,7 +108,11 @@ static const appSecCfg_t fitSecCfg =
 #endif
     DM_AUTH_SC_FLAG,                     /*! Authentication and bonding flags */
   0,                                     /*! Initiator key distribution flags */
-  DM_KEY_DIST_LTK,                       /*! Responder key distribution flags */
+#ifdef PRIVACY_RPA_FEATURE_ENABLE
+  DM_KEY_DIST_LTK|DM_KEY_DIST_IRK,
+#else
+    DM_KEY_DIST_LTK,                 /*! Responder key distribution flags */
+#endif
   FALSE,                              /*! TRUE if Out-of-band pairing data is present */
   FALSE                               /*! TRUE to initiate security upon connection */
 };
@@ -149,7 +156,17 @@ static const smpCfg_t fitSmpCfg =
   64000,                                    /*! Time msec before attemptExp decreases */
   2                                         /*! Repeated attempts multiplier exponent */
 };
+#ifdef PRIVACY_RPA_FEATURE_ENABLE
+extern uint8_t g_BLEMacAddress[6];
+#define RPA_TIMEOUT_SEC             (10)  // RPA timeout in second unit

+/*! local IRK */
+static uint8_t localIrk[] =
+{
+  0xA6, 0xD9, 0xFF, 0x70, 0xD6, 0x1E, 0xF0, 0xA4, 0x46, 0x5F, 0x8D, 0x68, 0x19, 0xF3,
0xB4, 0x96
+};
+wsfTimer_t     rpaAddrReadTimer;
+#endif
 /**************************************************************************************
   Advertising Data
 **********/
```

```
@@ -603,6 +620,9 @@ static void fitProcMsg(fitMsg_t *pMsg)
       AttsCalculateDbHash();
       DmSecGenerateEccKeyReq();
       fitSetup(pMsg);
+#ifdef PRIVACY_RPA_FEATURE_ENABLE
+      DmPrivSetResolvablePrivateAddrTimeout(RPA_TIMEOUT_SEC);
+#endif
       uiEvent = APP_UI_RESET_CMPL;
       break;

@@ -616,6 +636,9 @@ static void fitProcMsg(fitMsg_t *pMsg)

      case DM_ADV_START_IND:
      uiEvent = APP_UI_ADV_START;
+#ifdef PRIVACY_RPA_FEATURE_ENABLE
+ HciLeReadLocalResolvableAddr(appSlaveCb.peerAddrType[DM_ADV_HANDLE_DEFAULT],
                               appSlaveCb.peerAddr[DM_ADV_HANDLE_DEFAULT]);
+#endif
       break;

     case DM_ADV_STOP_IND:
@@ -642,7 +665,42 @@ static void fitProcMsg(fitMsg_t *pMsg)
       DmSecGenerateEccKeyReq();
       uiEvent = APP_UI_SEC_PAIR_CMPL;
       break;
+#ifdef PRIVACY_RPA_FEATURE_ENABLE
+   case FIT_RPA_ADDR_READ_TIMER_IND:
+   HciLeReadLocalResolvableAddr(appSlaveCb.peerAddrType[DM_ADV_HANDLE_DEFAULT],
                                 appSlaveCb.peerAddr[DM_ADV_HANDLE_DEFAULT]);
+   break;
+
+    case DM_PRIV_READ_LOCAL_RES_ADDR_IND:
+    {
+      hciLeReadLocalResAddrCmdCmplEvt_t *evt = (hciLeReadLocalResAddrCmdCmplEvt_t*)
                                               pMsg;

+      WsfTrace("current local RPA:%s", Bda2Str(evt->localRpa));
+      WsfTimerStartSec(&rpaAddrReadTimer, RPA_TIMEOUT_SEC);
+    }
+    break;
+
+    case DM_PRIV_ADD_DEV_TO_RES_LIST_IND:
+    {
+        dmSecKey_t *pPeerKey;
+        appDbHdl_t dbHdl;
+
+        /* get device database record handle */
+        dbHdl = AppDbGetHdl((dmConnId_t) pMsg->hdr.param);
+
+        /* if database record handle valid */
+        if (dbHdl != APP_DB_HDL_NONE)
+        {
+            pPeerKey =  AppDbGetKey(dbHdl, DM_KEY_IRK, NULL);
+
```

```
+            if(pPeerKey != NULL)
+            {
+                /* set advertising peer address */
+                AppSetAdvPeerAddr(pPeerKey->irk.addrType, pPeerKey->irk.bdAddr);
+            }
+        }
+    }
+      break;
+#endif
     case DM_SEC_PAIR_FAIL_IND:
       DmSecGenerateEccKeyReq();
       uiEvent = APP_UI_SEC_PAIR_FAIL;
@@ -751,6 +809,15 @@ void FitHandlerInit(wsfHandlerId_t handlerId)
   /* initialize heart rate profile sensor */
   HrpsInit(handlerId, (hrpsCfg_t *) &fitHrpsCfg);
   HrpsSetFlags(fitHrmFlags);
+#ifdef PRIVACY_RPA_FEATURE_ENABLE
+  // update Local IRK to be unique value
+  memcpy(localIrk, g_BLEMacAddress, BDA_ADDR_LEN);
+  /* Set IRK for the local device */
+  DmSecSetLocalIrk(localIrk);
+
+  rpaAddrReadTimer.msg.event = FIT_RPA_ADDR_READ_TIMER_IND;
+  rpaAddrReadTimer.handlerId = fitHandlerId;
+#endif

   /* initialize battery service server */
   BasInit(handlerId, (basCfg_t *) &fitBasCfg);
```

Workflow of RPA:

When the slave device (Apollo4 Blue) is paired with the peer device, the IRK will be exchanged and sent to the controller,. Then on the next time advertising will use RPA.

1. Pairing Apollo4 SDK fit example with phone.

> **NOTE:** As mentioned include above code snippet changes in AmbiqSuite SDK **third_party\cordio\ble-profiles\sources\apps\fit\fit_main.c**.

2. Disconnect the connection, and then the Apollo4 fit example will advertise using RPA.

Monitor SWO log of the Apollo4 FIT Example:

When the DUT starts advertising for the very first time, the log says **current local RPA:000000000000**.

```
FreeRTOS Fit Example
Running setup tasks...

RadioTask: setup

Starting wicentric trace:

BLE Controller Info:
  FW Ver:      1.21.0.0
  Chip ID0:    0x92492492
  Chip ID1:    0x41116025

No new image to upgrade
BLE Controller FW Auth Passed, Continue with FW
BLE Controller Init Done

FitHandlerInit
Calculating database hash
Fit got evt 101
Hci config trace ack, bitmap:0x00080000
[00:00:00:811.562] Custom data: id 0x0, 0x00008005
32K clock= 32773 Hz
Fit got evt 32
>>> Reset complete <<<
dmAdvActConfig: state: 0
dmAdvActSetData: state: 0
dmAdvActSetData: state: 0
dmAdvActStart: state: 0
HCI_LE_ADV_ENABLE_CMD_CMPL_CBACK_EVT: state: 3
dmDevPassEvtToDevPriv: event: 20, param: 33, advHandle: 0
Fit got evt 33
>>> Advertising started <<<
Fit got evt 62
current local RPA:000000000000
Database hash calculation complete
Fit got evt 17
```

After successfully pairing the devices, disconnect the DUT with Apollo4/Fit. The RPA will keep updating based on every 10 seconds due to **RPA_TIMEOUT_SEC** being set to 10 sec in the demo.

```
Fit got evt 40
>>> Connection closed <<<
dmAdvActConfig: state: 0
dmAdvActSetData: state: 0
dmAdvActSetData: state: 0
dmAdvActStart: state: 0
HCI_LE_ADV_ENABLE_CMD_CMPL_CBACK_EVT: state: 3
dmDevPassEvtToDevPriv: event: 20, param: 33, advHandle: 0
Fit got evt 33
>>> Advertising started <<<
Fit got evt 62
current local RPA:7F16F6646347
Fit got evt 163
Fit got evt 62
current local RPA:7F16F6646347
Fit got evt 163
Fit got evt 62
current local RPA:7056BE43F089
Fit got evt 163
Fit got evt 62
current local RPA:78D30B604904
Fit got evt 163
Fit got evt 62
current local RPA:45957DE8B150
Fit got evt 163
Fit got evt 62
current local RPA:6D22E6E40078
Fit got evt 163
Fit got evt 62
current local RPA:4029EA851082
Fit got evt 163
Fit got evt 62
current local RPA:44EB0DCE055E
Fit got evt 163
Fit got evt 62
current local RPA:7ECE1B14E77C
Fit got evt 163
Fit got evt 62
current local RPA:45C8BB37E947
Fit got evt 163
Fit got evt 62
current local RPA:57871FB8FB00
Fit got evt 163
Fit got evt 62
current local RPA:77D38B1EE751
Fit got evt 163
```

SECTION

9

Different Types of Advertising

This section describes how to apply different legacy advertising types in applications developed using the AmbiqSuite SDK.

Four types of legacy advertising are defined in the Bluetooth core specification. With different application requirements, different types of advertisement might be applied. The following table maps these four types with the settings defined in the Bluetooth Low Energy host stack solution in AmbiqSuite as well as in the scannable and connectable properties. This section focuses on how to apply it when developing applications with AmbiqSuite. The other features, advertising filter policy and privacy, which can change how a slave device responds to scan or connection requests, are assumed to be configured "not in use" or "disabled" and thus are not in effect.

Table 9-1: Types of Advertising

Advertising Type	Setting[1] in AmbiqSuite	Scannable	Connectable
ADV_IND	DM_ADV_CONN_UNDIRECT	x	x
ADV_DIRECT_IND	DM_ADV_CONN_DIRECT DM_ADV_CONN_DIRECT_LO_DUTY		x[2]
ADV_SCAN_IND	DM_ADV_SCAN_UNDIRECT	x	
ADV_NONCONN_IND	DM_ADV_NONCONN_UNDIRECT		

[1] Settings are defined in **<AmbiqSuite_root>/third_party/cordio/ble-host/include/dm_api.h**.

[2] Initiators other than the correctly addressed initiator shall not respond.

The following code changes can be applied to any Bluetooth Low Energy projects which need the corresponding advertising types.

ADV_IND:

This is a connectable and scannable undirected advertising type which allows a scanner or initiator to respond with either a scan request (**SCAN_REQ**) or connect request (**CONNECT_IND**).

This is the commonly used advertising type and is also the default advertising type used in most of Bluetooth Low Energy application examples in AmbiqSuite. In some applications, it might require to switch between advertising types. To switch back from another advertising type, call the following APIs in order when there's no ongoing advertising.

```
AppSetAdvType(DM_ADV_CONN_UNDIRECT);
AppAdvStart(APP_MODE_AUTO_INIT);
```

The initiator then sends a scan request (**SCAN_REQ**) and the corresponding response (**SCAN_RSP**) is sent back by advertiser.

ADV_SCAN_IND:

This is a non-connectable and scannable undirected advertising type. Device advertising with **ADV_SCAN_IND** will only respond to scan requests. The following code modification to enable **ADV_SCAN_IND** is quite straightforward. Call the following APIs in order when there is no ongoing advertising. Ambiq Suite SDK **ble_freertos_fit** example code taken as a reference.

```
static void fitSetup(fitMsg_t *pMsg)
{
  /* set advertising and scan response data for discoverable mode */
  AppAdvSetData(APP_ADV_DATA_DISCOVERABLE, sizeof(fitAdvDataDisc), (uint8_t *) fitAdv-
DataDisc);
  AppAdvSetData(APP_SCAN_DATA_DISCOVERABLE,sizeof(fitScanDataDisc),(uint8_t*) fitScan-
DataDisc);
+   AppSetAdvType(DM_ADV_SCAN_UNDIRECT);
+   AppAdvStart(APP_MODE_DISCOVERABLE);
-  /* start advertising; automatically set connectable/discoverable mode and bondable
mode */
-    AppAdvStart(APP_MODE_AUTO_INIT);
    }
```

ADV_NONCONN_IND:

This is a non-connectable and non-scannable undirected advertising type and is usually referred to as "beacon" mode. Device advertising with **ADV_NONCONN_IND** will not respond to any scan requests nor to connect requests. Since no scan requests will be responded to by device advertising with **ADV_NONCONN_IND**.

The following code modification to enable **ADV_NONCONN_IND** is quite straightforward. Call the following APIs in order when there is no ongoing advertising.Ambiq Suite SDK **ble_freertos_fit** example code taken as reference.

```
static void fitSetup(fitMsg_t *pMsg)
{
  /* set advertising and scan response data for discoverable mode */
  AppAdvSetData(APP_ADV_DATA_DISCOVERABLE,sizeof(fitAdvDataDisc),
                                                (uint8_t *)fitAdvDataDisc);
  AppAdvSetData(APP_SCAN_DATA_DISCOVERABLE,sizeof(fitScanDataDisc),
                                                (uint8_t*)fitScanDataDisc);
 +  AppSetAdvType(DM_ADV_NONCONN_UNDIRECT);
 +  AppAdvStart(APP_MODE_DISCOVERABLE);
 -  /* start advertising; automatically set connectable/discoverable mode and bondable
mode */
 -  AppAdvStart(APP_MODE_AUTO_INIT);

}
```

SECTION

10

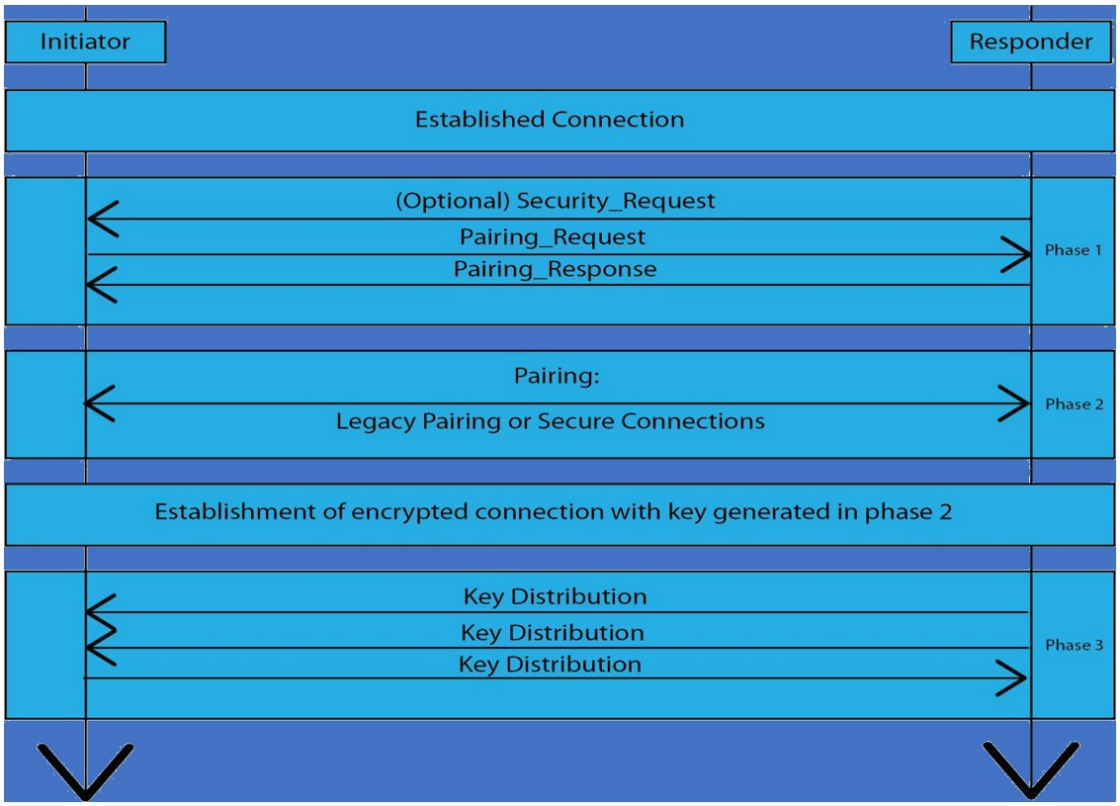
Saving and Managing Peer Credentials

The security manager defines the protocols and algorithms for generating and exchanging keys between two devices. It involves following security features:

- **Pairing**: the process of creating shared secret keys between two devices.
- **Bonding**: the process of creating and storing shared secret keys on each side (central and peripheral) for use in subsequent connections between the devices.
- **Authentication**: the process of verifying that the two devices share the same secret keys.
- **Encryption**: the process of encrypting the data exchanged between the devices. Encryption in BLE uses the 128-bit AES Encryption standard, which is a symmetric-key algorithm (meaning that the same key is used to encrypt and decrypt the data on both sides).
- **Message Integrity**: the process of signing the data, and verifying the signature at the other end. This goes beyond the simple integrity check of a calculated CRC

The security works in Bluetooth Low Energy with two important concepts: pairing and bonding, check the below sequence diagram showing the security process.

Figure 10-1: Security Process



Pairing is the combination of Phases 1 and 2. Bonding is represented by Phase 3 of the process. One important thing to note is that Phase 2 is the only phase that differs between LE Legacy Connections and LE Secure Connections.

Pairing and Bonding:

Phase 1:

In this phase, the slave may request the start of the pairing process. The master initiates the pairing process by sending a pairing request message to the slave, which then responds with a pairing response message.

The pairing request and pairing response messages represent an exchange of the features supported by each device, as well as the security requirements for each device. Each of these messages include the following:

- Input Output (IO) capabilities: display support, keyboard support, yes/no input support.
- Out-Of-Band (OOB) method support.
- Authentication requirements: includes MITM protection requirement, bonding requirement, secure connections support.
- Maximum encryption key size that the device supports.
- The different security keys each device is requesting to use

Phase 2:

Phase2 differs based on which method is used: LE secure connections or LE legacy connections.

Legacy Connections:

In legacy connections, there are two keys used: the temporary key (TK) and the short term key (STK). The TK is used along with other values exchanged between the two devices to generate the STK.

Secure Connections:

In secure connections, the pairing method does not involve exchanging keys over the air between the two devices. Rather, the devices utilize the ECDH protocol (WIKIpedia page here) to generate a public/private key pair. The devices then exchange the public keys only, and from that a shared secret key called the long term key (LTK) is generated.

Phase 3:

Phase 3 represents the bonding process. This is an optional phase that is utilized to avoid the need to re-pair on every connection to enable a secure communication channel. The result of bonding is that each device stores a set of keys that can be used in each subsequent connection and allows the devices to skip the pairing phase. These keys are exchanged between the two devices over a link that's encrypted using the keys resulting from phase two.

The following APIs represent the pairing and bonding of slave device

```
/*! \brief Data type for SMP_MSG_API_PAIR_REQ and SMP_MSG_API_PAIR_RSP */
typedef struct
{
  wsfMsgHdr_t         hdr;        /*!< \brief Message header */
  uint8_t             oob;        /*!< \brief Out-of-band data present flag */
  uint8_t             auth;       /*!< \brief authentication flags */
  uint8_t             iKeyDist;   /*!< \brief Initiator key distribution flags */
  uint8_t             rKeyDist;   /*!< \brief Responder key distribution flags */
} smpDmPair_t;

/*************************************************************************************/
/*!
 *  \brief  This function is called by a slave device to proceed with pairing after a
 *          DM_SEC_PAIR_IND event is received.
 *
 *  \param  connId    DM connection ID.
 *  \param  oob       Out-of-band pairing data present or not present.
 *  \param  auth      Authentication and bonding flags.
 *  \param  iKeyDist  Initiator key distribution flags.
```

```
 *  \param  rKeyDist  Responder key distribution flags.
 *
 *  \return None.
 */
/*************************************************************************************/
void DmSecPairRsp(dmConnId_t connId, bool_t oob, uint8_t auth, uint8_t iKeyDist,
uint8_t rKeyDist);

/**@}*/

/** \name App Security and Bonding Functions
 * Security and Bonding functions for configuration and interaction with \ref STACK_SMP
 * Pairing procedures.
 */
/**@{*/

/*************************************************************************************/
/*!
 *  \brief  Set the bondable mode of the device.
 *
 *  \param  bondable  TRUE to set device to bondable, FALSE to set to non-bondable.
 *
 *  \return None.
 */
/*************************************************************************************/
void AppSetBondable(bool_t bondable);

/*************************************************************************************/
/*!
 *  \brief  Initiate a request for security as a slave device.This function will send a
 *          message to the master peer device requesting security. The master device
 *          should either initiate encryption or pairing.
 *
 *  \param  connId    Connection identifier.
 *
 *  \return None.
 */
/*************************************************************************************/
void AppSlaveSecurityReq(dmConnId_t connId);

/*************************************************************************************/
/*!
 *  \brief  Clear all bonding information on a slave device and make it bondable.
 *
 *  \return None.
 *
```

```
 *  \Note   This API should not be used when Advertising (other than periodic
 *          advertising) is enabled. Otherwise, clearing the resolving list in the
 *          Controller may fail.
 */
/*************************************************************************************/
void AppSlaveClearAllBondingInfo(void);

If no connection as a slave, do the following procedure to pair and bond.

/* start or restart advertising */
 AppAdvStart(APP_MODE_AUTO_INIT);

/* enter discoverable and bondable mode mode */
 AppSetBondable(TRUE);
 AppAdvStart(APP_MODE_DISCOVERABLE);

/* clear all bonding info */
AppSlaveClearAllBondingInfo();
/* restart advertising */
AppAdvStart(APP_MODE_AUTO_INIT);
```

SECTION

11 Adding the Customized Service (CUSTS)

A Customized Service is created as a template service which consists of 4 sample characteristics, Write Only Sample Characteristic, Read Only Sample Characteristic, Notification Sample Characteristic and Indication Sample Characteristic. Find the service implementation from:

```
<AmbiqSuite_root>/ambiq_ble/services/svc_cust.h
<AmbiqSuite_root>/ambiq_ble/services/svc_cust.c
```

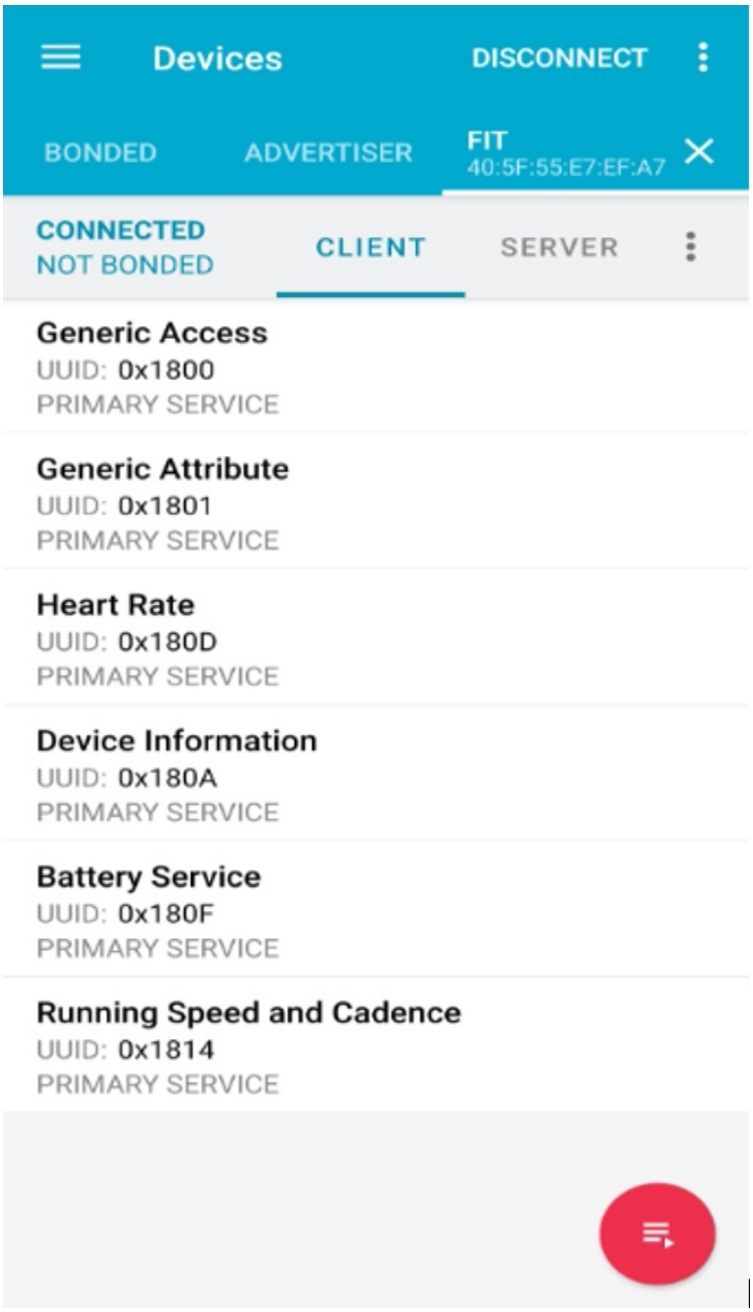
At the beginning of **svc_cust.h**, a macro define **INCLUDE_USER_DESCR** is defined which includes Characteristic User Description. This is optional.

Along with the above service implementation, the way to use the service is implemented in the following file. Through the APIs provided in this file, the application can manipulate the service easily.

```
<AmbiqSuite_root>/ambiq_ble/profiles/custss/custss_main.c
```

In order to help you add this customized service into the project take the **ble_freertos_fit** example project from AmbiqSuite SDK R4.x.x. The **ble_freertos_fit** example is already implemented with DIS and BAS Services. Connect to DUT (an Apollo4 Blue series board with the Fit Example loaded). The output is shown in Figure 11-1 on page 46.

Figure 11-1: Apollo4 Blue Series Boards with Fit Example Output



The below code snippet shows how to add customized service to the **ble_freertos_fit** example project. The Custom service code added under **TUTORIAL_ADDING_CUSTS** macro.

```
@File: AmbiqSuite_R4.x.x\third_party\cordio\ble-profiles\sources\apps\fit\fit_main.c
// Custom service
+ #ifndef TUTORIAL_ADDING_CUSTS
+   #include "ambiq_ble/services/svc_cust.h"
+  #include "ambiq_ble/profiles/custss/custss_api.h"
+  #endif
/*! WSF message event enumeration */
enum
{
    FIT_HR_TIMER_IND = FIT_MSG_START,       /*! Heart rate measurement timer expired */
    FIT_BATT_TIMER_IND,                     /*! Battery measurement timer expired */
    FIT_RUNNING_TIMER_IND,                  /*! Running speed and cadence measurement
                                                timer expired */
+  #ifndef TUTORIAL_ADDING_CUSTS
+    FIT_CUST_TIMER_IND
+  #endif
};
/*! enumeration of client characteristic configuration descriptors */
enum
{
    FIT_GATT_SC_CCC_IDX,    /*! GATT service, service changed characteristic */
    FIT_HRS_HRM_CCC_IDX,    /*! Heart rate service, heart rate monitor characteristic */
    FIT_BATT_LVL_CCC_IDX,   /*! Battery service, battery level characteristic */
    FIT_RSCS_SM_CCC_IDX,    /*! Running speed and cadence measurement characteristic */
+  #ifndef TUTORIAL_ADDING_CUSTS
+FIT_CUST_SS_CCC_IDX,
+  #endif
FIT_NUM_CCC_IDX
};

/*! client characteristic configuration descriptors settings, indexed by above enumera-
tion */
static const attsCccSet_t fitCccSet[FIT_NUM_CCC_IDX] =
{
  /* cccd handle            value range               security level */
    {GATT_SC_CH_CCC_HDL,    ATT_CLIENT_CFG_INDICATE,  DM_SEC_LEVEL_NONE},
                                                      /* FIT_GATT_SC_CCC_IDX */
    {HRS_HRM_CH_CCC_HDL,    ATT_CLIENT_CFG_NOTIFY,    DM_SEC_LEVEL_NONE},
                                                      /* FIT_HRS_HRM_CCC_IDX */
    {BATT_LVL_CH_CCC_HDL,   ATT_CLIENT_CFG_NOTIFY,    DM_SEC_LEVEL_NONE},
                                                      /* FIT_BATT_LVL_CCC_IDX */
    {RSCS_RSM_CH_CCC_HDL,   ATT_CLIENT_CFG_NOTIFY,    DM_SEC_LEVEL_NONE},
                                                      /* FIT_RSCS_SM_CCC_IDX */
+  #ifndef TUTORIAL_ADDING_CUSTS
+{CUSTS_HANDLE_SVC,   ATT_CLIENT_CFG_NOTIFY,    DM_SEC_LEVEL_NONE}
                                                      /* FIT_CUST_SS_CCC_IDX */
+   #endif
};
```

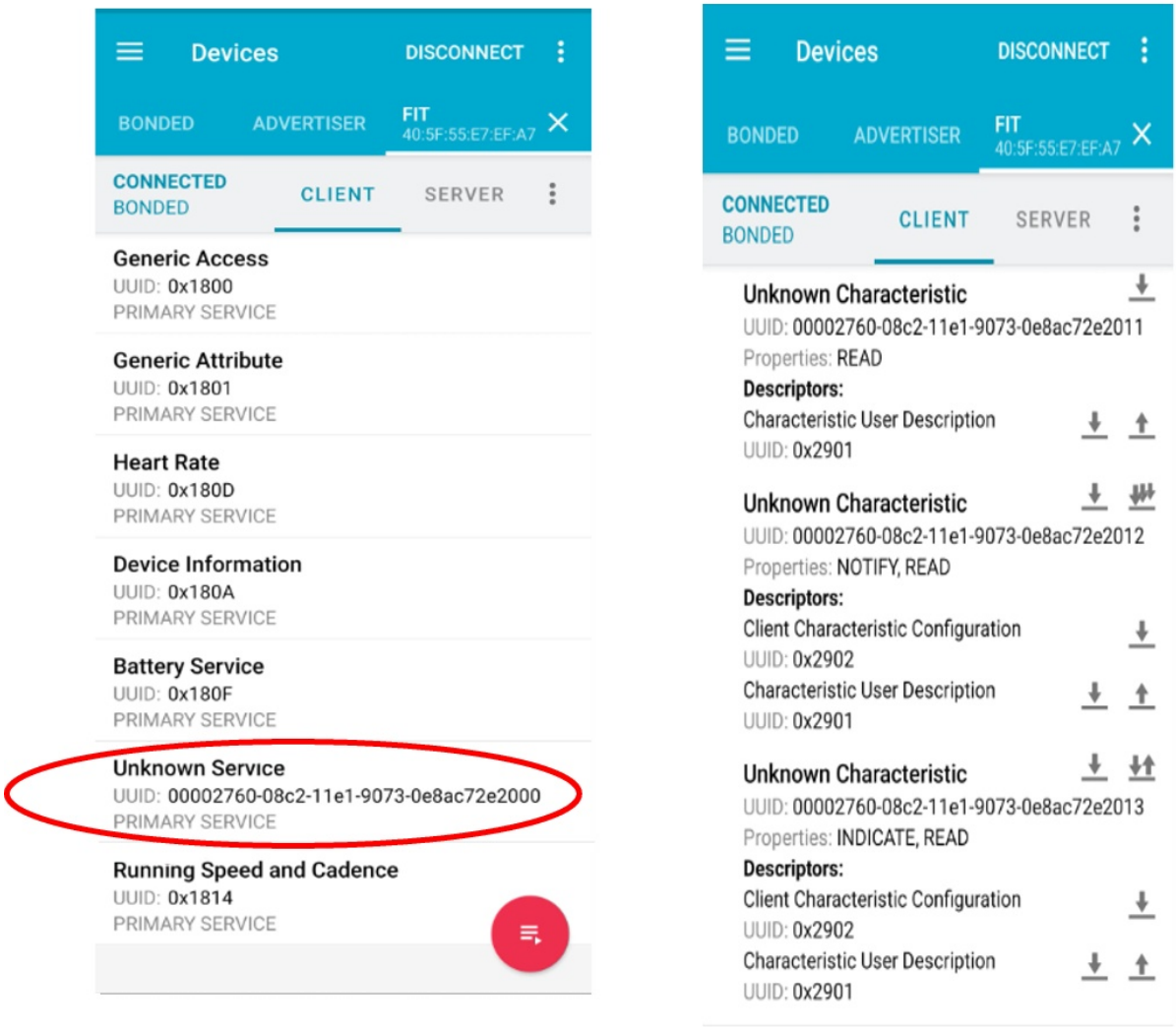
```
static void fitProcMsg(fitMsg_t *pMsg)
{
  uint8_t uiEvent = APP_UI_NONE;

  switch(pMsg->hdr.event)
   {
+       #ifndef TUTORIAL_ADDING_CUSTS
+          case FIT_CUST_TIMER_IND:
+                  CustssProcMsg(&pMsg->hdr);
+                 break;
+#endif
     }
}
void FitHandlerInit(wsfHandlerId_t handlerId)
{
    APP_TRACE_INFO0("FitHandlerInit");
   /* initialize battery service server */
    BasInit(handlerId, (basCfg_t *) &fitBasCfg);
     +  #ifndef TUTORIAL_ADDING_CUSTS
+   /* initialize Custom service server */
+CustssInit(handlerId);
+ #endif
}

/*************************************************************************************/
/*  \brief  Start the application.
 *  \return None.
 */
/*************************************************************************************/
void FitStart(void)
{
    SvcBattAddGroup();
    SvcRscsAddGroup();
+  #ifndef TUTORIAL_ADDING_CUSTS
+SvcCustAddGroup();
+  #endif
}
```

Compile and flash the fit project on the Apollo4 Blue evaluation board with the above-mentioned Custom Service changes. Connect to DUT. The output is as shown in Figure 11-2. The Custom Service is added as "Unknown Service".

Figure 11-2: Custom Service Added as "Unknown Service"



6500 River Place Boulevard, Building 7, Suite 200, Austin, TX 78730
www.ambiq.com
sales@ambiq.com
+1 (512) 879-2850

A-SOCA4B-UGGA02EN v1.0
December 2022