AN92584

Designing for Low Power and Estimating Battery Life for BLE Applications

Authors: Uday Agarwal, Kunal Patel, Vikram, Prathap Reddy, Santosh S

Associated Project: Yes

Associated Part Family: CY8C4XX7-BL, CY8C4XX8-BL, CYBL1XX6X, CYBL1XX7X

Software Version: PSoC® Creator™ 3.3

Related Application Notes: AN91267, AN94020

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/AN92584

More code examples? We heard you.

To access an ever-growing list of hundreds of PSoC code examples, please visit our code examples web page. You can also explore the PSoC 4 video library here.

AN92584 teaches you how to design low-power applications with PSoC 4/PRoC™ BLE devices. It also guides you on how to compute the current consumption and battery life for a BLE application and provides tips and tricks to minimize the current consumption to increase battery life.

Contents

1 Introduction....1
2 Design and Implementation for Low-Power BLE....2
2.1 System Clocks....2
2.2 System Power Modes....3
2.3 BLE Subsystem Power Modes....4
2.4 Recommendations for Low Power....5
2.5 Low-Power Implementation....5
3 Example Projects....16
3.1 Example Project 1: Low-Power Modes in Advertisement....16
3.2 Example Project 2: Low-Power Modes in Connection....17
3.3 Average Current Measurement....17
3.4 Power Calculator....22
4 BLE Applications....23
4.1 BLE System Power....23
4.2 Battery Life....24
4.3 Techniques for Increasing Battery Life....26
4.4 Example Application: Heart-Rate Monitor....28
4.5 Example Application: Remote Control....30
4.6 Implementing System Design Recommendations....33
5 Summary....34
6 References....34
A Appendix A: Advertising State Current Profile....35
A.1 Current Profile....36
B Appendix B: Connection State Current Profile....39
B.1 Current Profile....40

1 Introduction

Bluetooth Low Energy (BLE) devices such as heart-rate monitors are typically battery operated. A long battery life is a key requirement for such devices. This application note shows how to implement a low-power solution and estimate the battery life for the device using Cypress's BLE solutions.

Before you read this document, you should have a basic knowledge of BLE and have read the Getting Started with PSoC 4 BLE or Getting Started with PRoC BLE application notes.

This application note is divided into two sections. The first section guides you in implementing low-power BLE solutions using PSoC 4/PRoC BLE devices. It first provides an overview of the clocks and low-power modes available in these devices. It then explains how to manage the clocks and the power modes in your application to reduce current consumption. The implementation is illustrated in the example projects provided with this application note. A procedure to measure the average current in the CY8CKIT-042-BLE Bluetooth Low Energy (BLE) Pioneer Kit is also discussed. A power calculator tool provided with this application note makes it easy for you to estimate the current consumption for a given configuration of the device.

The second section provides a system-level view of a BLE device and shows how to estimate its battery life. Tips and tricks to reduce the average current consumption and improve the battery life are discussed and illustrated with the help of two example use-cases: a heart-rate monitor and a remote control. The current consumption in the non-BLE parts of the system and the idle time between BLE operations adds significantly to the average current consumption.

2 Design and Implementation for Low-Power BLE

It is important that you first understand the different clocks and power modes available in PSoC 4/PRoC BLE devices that should be managed to conserve power.

2.1 System Clocks

The PSoC 4/PRoC BLE clock system includes the following clock sources:

Two internal clock sources:

- 3-MHz to 48-MHz internal main oscillator (IMO) ±2 percent across all frequencies with trim
- 32-kHz internal low-speed oscillator (ILO) ±60 percent with trim

Three external clock sources:

- External clock (EXTCLK) generated using a signal from an I/O pin
- 24-MHz external crystal oscillator (ECO)
- 32-kHz watch crystal oscillator (WCO)

The five clock sources and the clocks derived from these sources are shown in Figure 1.

Figure 1. PSoC 4/PRoC BLE System Clocks

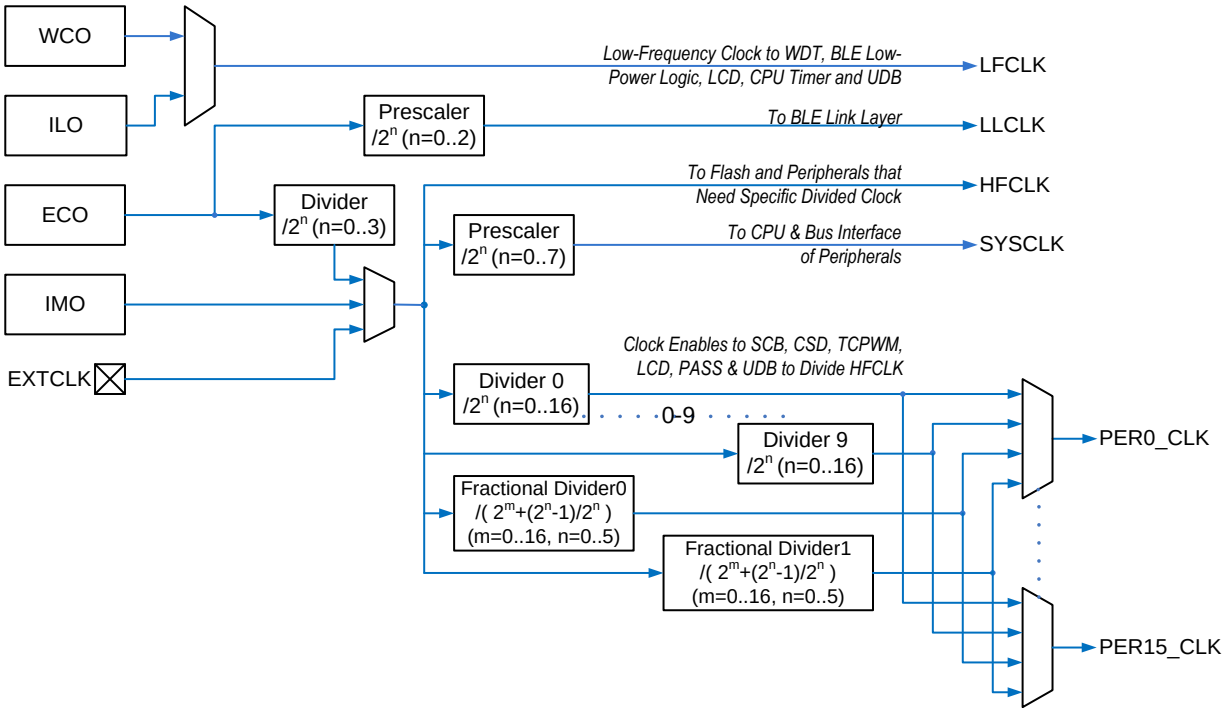


Table 1 summarizes the clock sources, their use in the system, and specific system constraints on them. The table also lists the APIs that are used to turn ON or OFF the clock sources.

Table 1. System Clocks

Clock Source	Frequency	System Requirements	System Constraints	APIs
IMO	3 MHz -48 MHz	Used to run the CPU and all other peripherals	Either the IMO or ECO is required to run the CPU. The CPU must use the IMO while performing flash write operations.	ON – CySysClkImoStart() OFF – CySysClkImoStop()
ECO	24 MHz	Used by BLE subsystem (BLESS) for packet transmissions and reception. Can also be used to run the CPU	Either the IMO or ECO is required to run the CPU.	ON – CySysClkEcoStart() OFF – CySysClkEcoStop()
WCO	32 kHz	Used by BLESS to maintain link timing. Can also be used to run the watchdog timer	Either the WCO or ILO is required to run the watchdog timer.	ON – CySysClkWcoStart() OFF – CySysClkWcoStop()
ILO	32 kHz	Can be used to run the watchdog timer if WCO is not available	The ILO cannot be used for BLESS link timing.	ON – CySysClkIloStart() OFF – CySysClkIloStop()

2.2 System Power Modes

PSoC 4/PRoC BLE devices support five system power modes. Table 2 summarizes these modes, their currents, active components, wakeup sources, and the APIs available to put the system into one of the low-power modes.

Table 2. System Power Modes

System Power Mode	Current Consumption	Code Execution	RAM Available	Digital Peripherals Available	Analog Peripherals Available	Clock Sources Available	Wakeup Sources	Wakeup Time	Wakeup State	API
Active	850 µA + 260 µA per MHz	Yes	ON	All	All	All	–	–	–	–
Sleep	850 µA + 60 µA per MHz	No	Retention	All	All	All	Any interrupt source	0	Active	CySysPmSleep()
Deep-Sleep	1.3 µA	No	Retention	WDT[1], LCD[2], SCB(I²C/SPI only), BLESS[3]	LP Comparator, CTBm[11], POR[4], BOD[5]	WCO[6], ILO[7]	LP Comparator, GPIO[8], CTBm, BLESS[3], WDT, SCB[9]	25 µs	Active	CySysPmDeepSleep()
Hibernate	150 nA	No	Retention	No	LP Comparator, POR, BOD	None	LP Comparator, GPIO	2 ms	Chip Reset	CySysPmHibernate()
Stop	60 nA	No	OFF	No	No	None	WAKEUP, XRES[10] pins	2 ms	Chip Reset	CySysPmStop()

[1] Watchdog timer
[2] Liquid crystal display
[3] BLE subsystem
[4] Power-on reset
[5] Brownout detect
[6] 32-kHz watch crystal oscillator
[7] 32-kHz internal low-speed oscillator
[8] General-purpose input/output
[9] Serial communication block
[10] External reset
[11] Continuous time block mini

2.3 BLE Subsystem Power Modes

In addition to the system power modes, PSoC 4/PRoC BLE devices support three BLESS power modes. Table 3 summarizes them, their active components, and their mapping to the internal states of the BLESS while being in these power modes.

- The BLESS power modes directly map to the input parameters of the `CyBle_EnterLPM`() API that is used to put the BLESS into the specific power mode.
- The BLESS internal states directly map to the states returned from the `Get_Blessstate()` API.
- The BLESS DEEPSLEEP and SLEEP modes are entered under application control. The exit may be initiated in one of two ways:
  - The application calls the `CyBle_EnterLPM()` function with the input parameter as ACTIVE.
  - The BLESS internally triggers the exit at a time determined by the BLE stack within the BLE Component.
- Upon exit from the DEEPSLEEP or SLEEP mode, the BLESS enters the ACTIVE mode.
- The CYBLE_BLESS_STATE_ECO_ON and CYBLE_BLESS_STATE_ECO_STABLE states are entered for a short duration (up to a 1-ms period) while the BLESS is transitioning from the DEEPSLEEP mode to the ACTIVE mode.

Table 3. BLESS Power Modes

BLESS Power Mode	BLESS Internal States	BLESS Operation	BLESS Radio Clock	BLESS Link Timing Clock Source	ECO	Internal State Entry Control	Allowed System Power Modes
DEEPSLEEP	CYBLE_BLESS_STATE_DEEPSLEEP	Idle. Maintain link.	OFF	WCO	OFF	CPU	Deep-Sleep Sleep Active
	CYBLE_BLESS_STATE_ECO_ON	ECO startup and amplitude stabilization	OFF	WCO	ON	BLESS	Deep-Sleep Sleep Active
	CYBLE_BLESS_STATE_ECO_STABLE	ECO frequency stabilization and BLE timing synchronization	OFF	WCO	ON	BLESS	Sleep Active
SLEEP	CYBLE_BLESS_STATE_SLEEP	Idle. Maintain link.	OFF	ECO	ON	CPU	Sleep Active
ACTIVE	CYBLE_BLESS_STATE_ACTIVE	Idle. Maintain link.	OFF	ECO	ON	CPU	Sleep Active
	CYBLE_BLESS_STATE_ACTIVE	Transmit	ON (Transmitter)	ECO	ON	BLESS	Sleep Active
	CYBLE_BLESS_STATE_ACTIVE	Receive	ON (Receiver)	ECO	ON	BLESS	Sleep Active
	CYBLE_BLESS_STATE_EVENT_CLOSE	Enter Deep-Sleep mode.	OFF	ECO	ON	BLESS	Active

The WCO clock source must be ON to use the DEEPSLEEP mode to maintain the BLE link timing during the DEEPSLEEP mode. The WCO may be turned OFF if BLE is not used or the BLESS DEEPSLEEP power mode is not used.

The system can be in the Sleep or Active power modes during the BLESS SLEEP and ACTIVE power modes. It can be put into the Deep-Sleep mode only when the BLESS is in the DEEPSLEEP mode and the BLESS link layer and modem logic are not using the ECO clock. To ensure that invalid combinations of the system and BLESS modes are not entered, the application should check the BLESS internal state first and then put the system into the appropriate low-power mode.

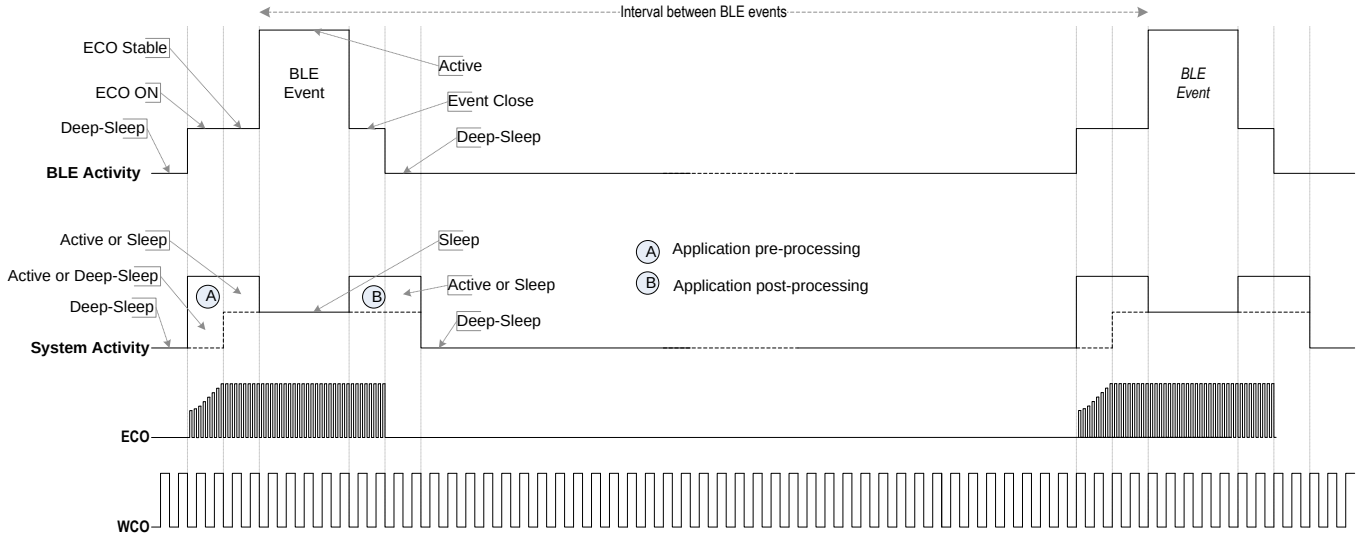
2.4 Recommendations for Low Power

The user application should manage the system clocks, system power modes, and BLESS power modes to implement a low-power design in PSoC 4/PRoC BLE devices.

The following are the recommendations for implementing low-power designs (see Figure 2):

- The BLESS should be put into the DEEPSLEEP mode between BLE events. The system should be put into the Deep-Sleep mode during this time if there is no pending application processing.
- The system should be put into the Sleep mode when the BLESS is in the ACTIVE or SLEEP modes if no application processing is required. The system can also be put into the Sleep mode when the BLESS is in the CYBLE_BLESS_STATE_ECO_STABLE state.
- If the system is in the Sleep mode while the BLESS is in the ACTIVE or SLEEP modes, the ECO clock source should be used for the CPU clock. This allows the IMO to be turned OFF to reduce current consumption.
- If the ECO is used in the system Sleep mode, the ECO clock should be divided down from 24 MHz to 3 MHz. The 3-MHz frequency is enough to keep the system in the Sleep mode and switch back to the IMO clock when the system exits the Sleep mode.
- The WCO must be selected with a low ppm variation. You should further tune it to reduce the ppm variation. A lower ppm reduces the listening window time in the BLE Peripheral role, thus reducing the current consumed. Refer to AN95089 – PSoC 4/PRoC BLE Crystal Oscillator Selection and Tuning Techniques for details on WCO crystal tuning.
- If your application does not use the ILO, the ILO should be stopped to reduce current consumption.

Figure 2. Low-Power Implementation Recommendations



2.5 Low-Power Implementation

This section discusses how to implement the low-power-mode recommendations of section 2.4 in your BLE project. It involves two steps:

1. Configure your project in PSoC Creator™.
2. Implement the recommendations in your application code.

2.5.1 PSoC Creator Configuration

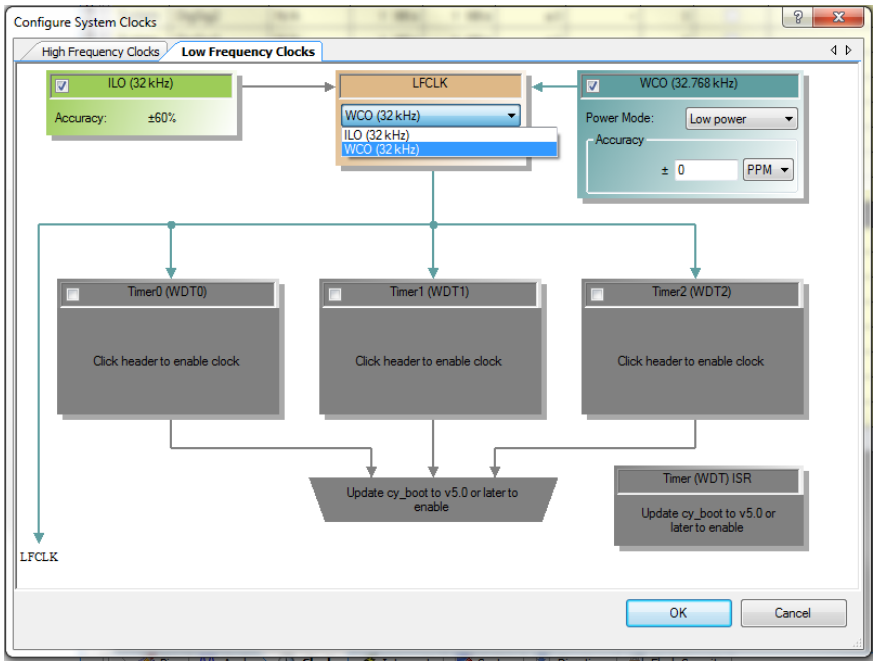
The following configurations should be implemented in your project to ensure a low-power operation.

2.5.1.1 Low-Frequency Clock (LFCLK) Selection

The LFCLK in the system must be set to use the WCO to operate the BLESS in the DEEPSLEEP mode.

1. Access this setting in the **Clocks** tab of the *.cydwr* file in your project.
2. Click **Edit Clock** to open the **Configure System Clocks** window, as shown in Figure 3.
3. Go to the **Low Frequency Clocks** tab.
4. Confirm that the WCO tab is selected and the WCO is selected as LFCLK.
5. Close the window.

Figure 3. LFCLK Setting

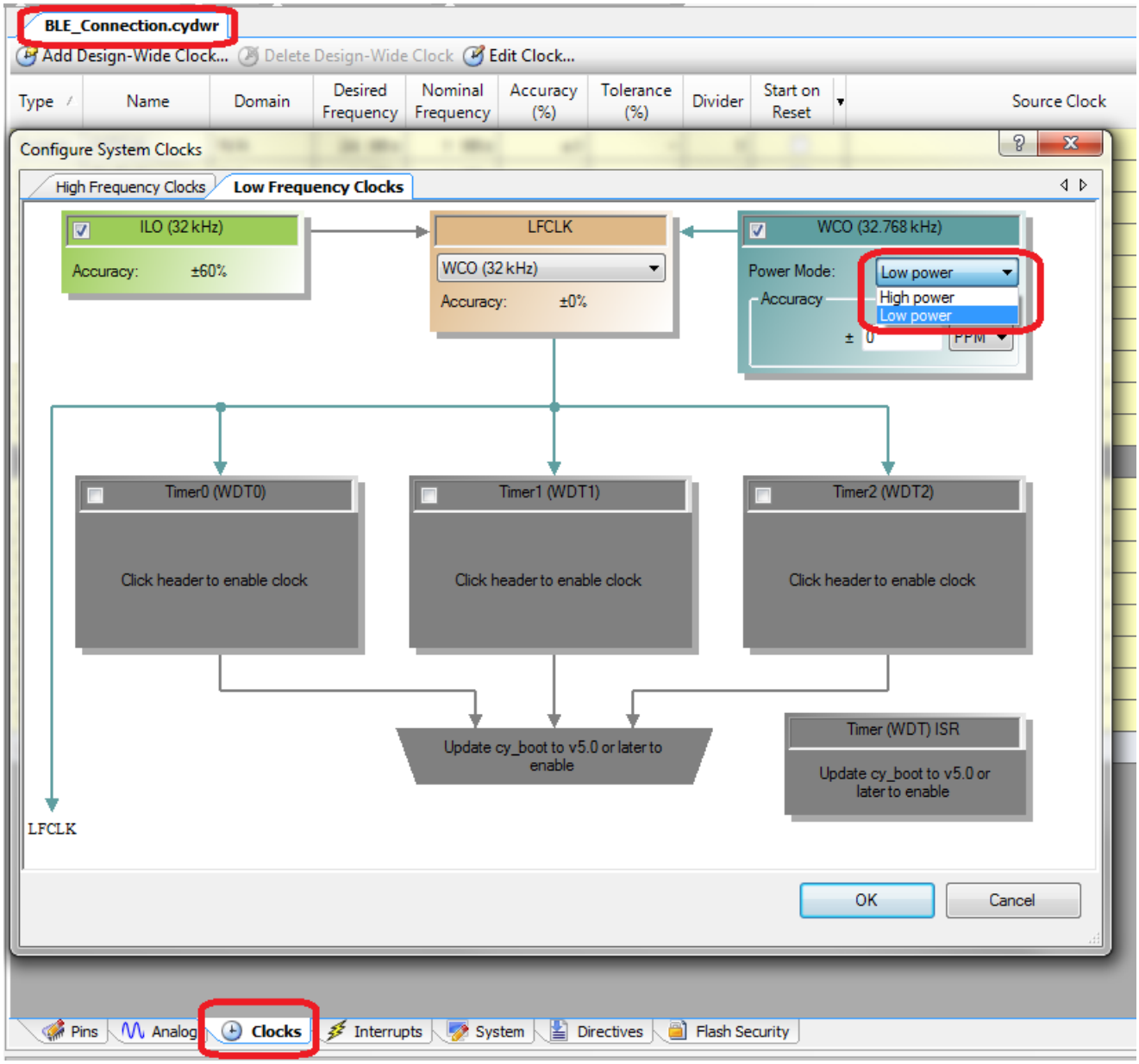


2.5.1.2 WCO Power Mode

The WCO supports two modes of operation: low power and high power. The high-power mode is required for a fast start up of the oscillator block upon power up of the chip. After the oscillation has stabilized, the WCO must be set to operate in the low-power mode to reduce current consumption.

1. Access this setting under the **Clocks** tab of the *.cydwr* file.
2. Click **Edit Clock** to open the **Configure System Clocks** window, as shown in Figure 4.
3. Go to the **Low Frequency Clocks** tab.
4. Select the **Low Power** option.
5. Close the window.

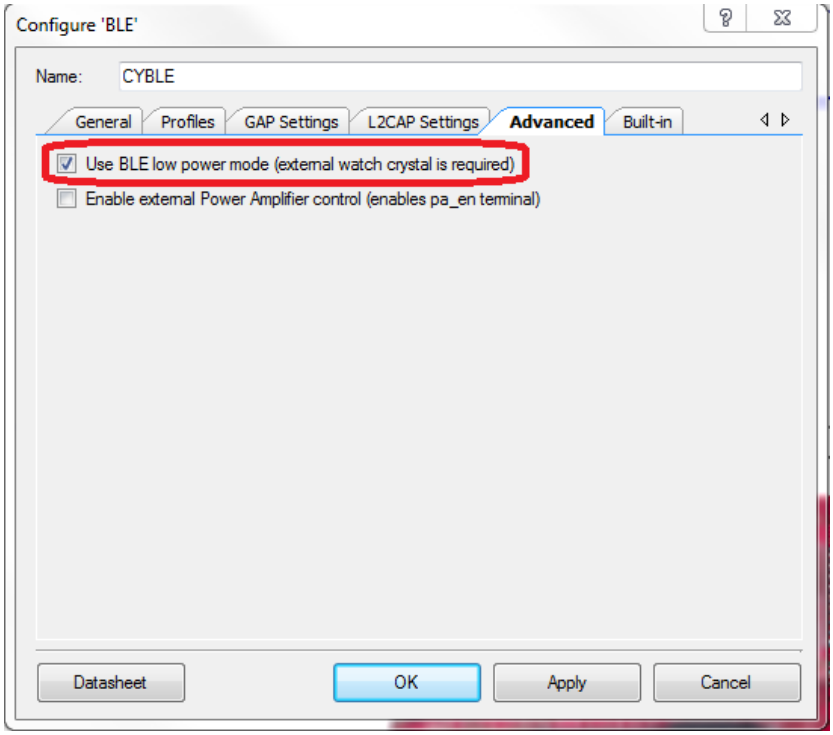
Figure 4. WCO Low-Power Mode Settings



2.5.1.3 BLE Component Deep-Sleep Mode

The **Use BLE low power mode** button in the BLE Component customization window must be selected to use the BLESS DEEPSLEEP mode. This setting is available in the **Advanced** tab of the BLE Component configuration, as shown in Figure 5. If this setting is selected, the WCO must be configured as the LFCLK source in the Design-Wide Resources (DWR) Clock Editor, as shown in Figure 3. If the WCO is not configured for LFCLK and this button is selected, then an error is reported when you build the project.

Figure 5. Deep Sleep Setting for BLE Component



2.5.1.4 Stop Mode

If you are using the Stop mode in your application, make sure that the WAKEUP pin is configured correctly to exit the Stop mode successfully:

1. Configure P2.2 as the WAKEUP pin.
2. Set the WAKEUP pin active HIGH or active LOW as required by your application.

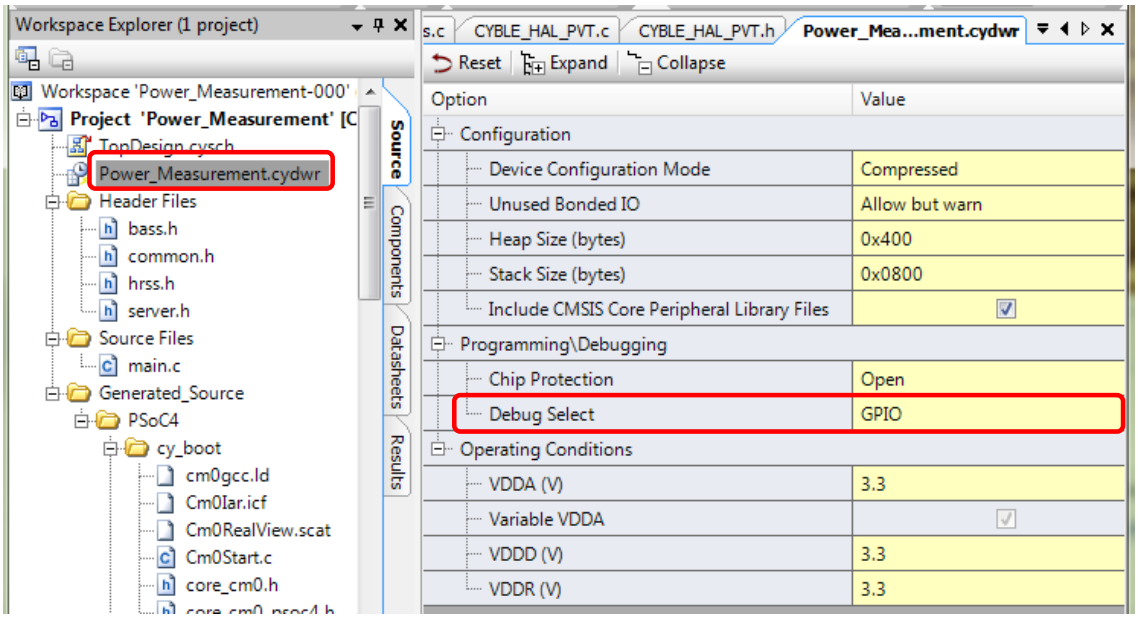
Setting the WAKEUP pin to the configured level will cause the device to wake up from the Stop mode. The WAKEUP pin is active LOW by default.

2.5.1.5 SWD Pin Configuration

SWD pins are used for runtime firmware debug during development stages. Configuring the SWD pins for debug increases the current consumption. Therefore, in the production release, SWD pins should be switched to the GPIO mode. They will still be available for programming the device upon chip reset.

1. Go to the **System** tab in the *.cydwr* file.
2. Click **Debug Select** under **Programming\Debugging,** as shown in Figure 6.
3. Change the value to **GPIO**.

Figure 6. Debug Pin Setting



2.5.2 Implementing Low-Power Operations in Application Code

The implementation is divided into three functional blocks in your application:

- System initialization and main loop
- BLE stack event handler
- Power management functions

2.5.2.1 System Initialization and Main Loop

After a power on or reset, the system initializes various components by calling the respective start functions. The following steps are performed at initialization for a low-power operation. Refer to the code snippet with comments (comments are labeled with “C<number>”) for the reference implementation.

- [C1] Stop the ILO to reduce current consumption.
- [C2] Configure the divider values for the ECO so that a 3-MHz ECO divided clock can be provided to the CPU in the Sleep mode. The clock to the CPU is switched from the IMO to the ECO before entering the Sleep mode. The ECO-derived clock is available when the CPU wakes up from the Sleep mode.
- [C3] In the main while loop, call the function `CyBle_EnterLPM()` () to put the BLESS into the DEEPSLEEP mode soon after the BLE event processing is completed. Note that this function will internally check the BLESS conditions if it can enter the DEEPSLEEP mode. Therefore, no check is required before calling this function.
- [C4, C5] If the system is active, run your application as illustrated by calling the `run_application()` function [C4]. Once the application completes all its tasks, check if it is ready to enter a low-power mode. The application can enter a low-power mode if all the non-BLE components used in the application are idle. The application should enter the Sleep mode if some of the components need the high-frequency clock (ECO or IMO) to wake up the system. The application should enter the Deep-Sleep mode if the clock to all the components can be shut down. This is done by setting a flag (called `applicationPower` in the code) to SLEEP or DEEPSLEEP. The flag is used by the `ManageApplicationPower()` function to put the components into the Sleep or Deep-Sleep modes. Note that the definition of the Sleep and Deep-Sleep modes are application-specific. Refer to the function definition of `run_application()` for a reference implementation [C5].
- [C6] Call the `ManageApplicationPower()` function to manage the power mode transitions for the application-specific, non-BLE components.
- [C7] Call the `ManageSystemPower()` function. The function checks both the application power mode and the BLESS power mode to put the entire system into the Sleep or Deep-Sleep mode.

```
int main()
{
    /* Variable declarations */
    CYBLE_LP_MODE_T lpMode;
    CYBLE_BLESS_STATE_T blessState;
    uint8 interruptStatus;

    /* Enable global interrupts */
    CyGlobalIntEnable;

    /* C1. Stop the ILO to reduce current consumption */
    CySysClkIloStop();

    /* C2. Configure the divider values for the ECO, so that a 3-MHz ECO divided
    clock can be provided to the CPU in Sleep mode */
    CySysClkWriteEcoDiv(CY_SYS_CLK_ECO_DIV8);

    /* Start the BLE Component and register the generic event handler */
    apiResult = CyBle_Start(AppCallBack);

    /* Wait for BLE Component to initialize */
    while (CyBle_GetState() == CYBLE_STATE_INITIALIZING)
    {
        CyBle_ProcessEvents();
    }
    /*Application-specific Component and other initialization code below */
      applicationPower = ACTIVE;

    /* main while loop of the application */
    while(1)
    {
        /*  Process all pending BLE events in the stack */
        CyBle_ProcessEvents();

        /* C3. Call the function that manages the BLESS power modes */
        CyBle_EnterLPM(CYBLE_BLESS_DEEPSLEEP);
       /*C4. Run your application specific code here */
       if(applicationPower == ACTIVE)
       {
            RunApplication();
       }

       /*C6. Manage Application power mode */
       ManageApplicationPower();

       /*C7. Manage System power mode */
       ManageSystemPower();
   }
}
/*******************************************************************************
* C5. Function Name: RunApplication()
********************************************************************************
*
* Summary:
*   This function is a template to run Application-specific code.
*
* Parameters:
* none
*
*******************************************************************************/
```

```
inline void RunApplication()
{
    /*********************************************************************
    *  Place your application code here
    *********************************************************************/

    /* if you are done with everything and ready to go to sleep,
    then set it up to go to sleep. Update the code inside if() specific
    to your application*/
    if(0)
    {
        applicationPower = SLEEP;
    }

    /* if you are done with everything and ready to go to deepsleep,
    then set it up to go to deepsleep. Update the code inside if() specific
    to your application*/
    if (1)
    {
        applicationPower = DEEPSLEEP;
    }
}
```

2.5.2.2 BLE Stack Event Handler

The BLE stack event handler function handles the events generated by the BLE stack. Employ the following low-power techniques in the event-handling section soon after the BLE stack is initialized (an EVT_STACK_ON event is received):

- [C8] Set the device sleep-clock accuracy (SCA) based on the tuned ppm of the WCO. The SCA is defined by the Bluetooth specifications in seven subranges in the total range of 0 ppm to 500 ppm.

The code snippet for these is shown below:

```
void AppCallBack(uint32 event, void* eventParam)
{
    CYBLE_BLESS_CLK_CFG_PARAMS_T clockConfig;

    switch(event)
    {
      /* Handle stack events */
      case CYBLE_EVT_STACK_ON:

          /* C8. Get the configured clock parameters for BLE subsystem */
          CyBle_GetBleClockCfgParam(&clockConfig);

          /* C8. Set the device sleep-clock accuracy (SCA) based on the tuned ppm
          of the WCO */
          clockConfig.bleLlSca = CYBLE_LL_SCA_000_TO_020_PPM;

          /* C8. Set the clock parameter of BLESS with updated values */
          CyBle_SetBleClockCfgParam(&clockConfig);

          /* Put the device into discoverable mode so that a Central device can
          connect to it. */
          apiResult = CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);

          /* Application-specific event handling here */
          break;

      /* Other application-specific event handling here */
      case CYBLE_EVT_GAP_DEVICE_CONNECTED:

    }
}
```

2.5.2.3 Power Management Functions

The power management functions control the application and system power mode transitions.

Application Power Management

The function `ManageApplicationPower()` manages the power state transitions for all the components used by the application, except the BLE Component. Five application power states are defined. You should customize this function for your application.

The ACTIVE power state indicates that the application is active. There is no specific action required when the application is active.

The WAKEUP_SLEEP and WAKEUP_DEEPSLEEP states indicate that the system is waking up from the Sleep or Deep-sleep mode, respectively. The application should also wake up the components from their Sleep or Deep-Sleep modes.

The SLEEP and DEEP SLEEP states are entered when the application has completed all application processing and is requested at the end of it to enter the Sleep or Deep-Sleep modes. The non-BLE components used by the application are put into their respective Sleep or Deep-Sleep modes. Note that the definition of the Sleep and Deep-Sleep are application-specific.

```
void ManageApplicationPower()
{
    switch(applicationPower)
    {
        case ACTIVE: // don't need to do anything
        break;

        case WAKEUP_SLEEP: // do whatever wakeup needs to be done

            applicationPower = ACTIVE;
        break;

        case WAKEUP_DEEPSLEEP: // do whatever wakeup needs to be done.

            applicationPower = ACTIVE;
        break;



        case SLEEP:
        /*************************************************************************
         * Place code to place the application components to sleep here
        *************************************************************************/
        break;

        case DEEPSLEEP:
        /*************************************************************************
        *  Place code to place the application components to deepsleep here
        *************************************************************************/

        break;
    }
}
```

2.5.2.4 System Power Management

The application should call the `ManageSystemPower()` function to put the system into low-power modes. The function puts the system into the allowed low-power modes as follows:

- [C9] Get the current internal state of the BLESS by calling the `CyBle_GetBleSsState()` function.
- [C10, C11] If the BLESS is in the DEEPSLEEP mode and the non-BLE application components are also in the Deep-Sleep mode [C10], then put the system into the Deep-Sleep mode [C11]. The code execution halts here until the system wakes up from the Deep-Sleep mode due to an interrupt.
- [C12] If the BLESS does not enter the DEEPSLEEP mode, it is either because it is at the beginning or in the middle of an event. The system can be put into the Sleep mode in this period except in the `CYBLE_BLESS_STATE_EVENT_CLOSE` state.
- [C13, C14, C15, C16] There are two possibilities about the application power mode under the above condition. If the application is in the Deep-Sleep mode [C13], the system can use the ECO instead of the IMO as the clock source in the Sleep mode. First, the HFCLK source is switched to the ECO and the IMO is stopped [C14]. The system is then put into the Sleep mode [C15]. The code execution halts here until the system wakes up from the Sleep mode due to an interrupt. The IMO is restarted upon wakeup and the clock source for HFCLK is reverted to the IMO [C16].
- [C17, C18] If the application is in the Sleep mode and requires the IMO [C17], then the system is put into the Sleep mode [C18], without switching OFF the IMO.

  Note 1: It is important that the code to handle the low-power transitions is protected in a critical section and that interrupts are not allowed to change the thread of operation. In the code snippet, this critical section is bound by the `CyEnterCriticalSection()` function at the beginning and the `CyExitCriticalSection()` function at the end. If you do not put the code in this critical section, it may result in race conditions between the system and the BLESS in entering the BLESS low-power modes, causing the device to enter an unknown state from which it cannot recover.

  Note 2: When you put the BLESS into the DEEPSLEEP mode in the application using the `CyBle_EnterLPM()` function call, the ECO is configured to be OFF within the function. Therefore, the application need not do an explicit ECO OFF. However, if you are not using the BLESS, then you need to explicitly stop the ECO in the system if it is not required. This is because the system API call, `CySysPmDeepSleep()`, will **not** turn OFF the ECO. This can be done by calling the `CySysClkEcoStop()` API, described previously, just before putting the system into the Deep-Sleep mode.

The code snippet follows.

```
void ManageSystemPower()
{
    /* Variable declarations */
    CYBLE_BLESS_STATE_T blePower;
    uint8 interruptStatus ;

    /* Disable global interrupts to avoid any other tasks from interrupting this
section of
    code*/
    interruptStatus  = CyEnterCriticalSection();

    /* C9. Get current state of BLE sub system to check if it has successfully
entered deep
    sleep state */
    blePower = CyBle_GetBleSsState();

    /* C10. System can enter Deep-Sleep only when BLESS and rest of the application
are in
    DeepSleep or equivalent power modes */
    if((blePower == CYBLE_BLESS_STATE_DEEPSLEEP || blePower ==
CYBLE_BLESS_STATE_ECO_ON) &&
        applicationPower == DEEPSLEEP)
    {
        applicationPower = WAKEUP_DEEPSLEEP;
```

```
            /* C11. Put system into Deep-Sleep mode*/
            CySysPmDeepSleep();
        }
        /* C12. BLESS is not in Deep Sleep mode. Check if it can enter Sleep mode */
        else if((blePower != CYBLE_BLESS_STATE_EVENT_CLOSE))
        {
            /* C13. Application is in Deep Sleep. IMO is not required */
            if(applicationPower == DEEPSLEEP)
            {
                applicationPower = WAKEUP_DEEPSLEEP;

                /* C14. change HF clock source from IMO to ECO*/
                CySysClkWriteHfclkDirect(CY_SYS_CLK_HFCLK_ECO);
                /* C14. stop IMO for reducing power consumption */
                CySysClkImoStop();
                /*C15. put the CPU to sleep */
                CySysPmSleep();
                /* C16. starts execution after waking up, start IMO */
                CySysClkImoStart();
                /* C16. change HF clock source back to IMO */
                CySysClkWriteHfclkDirect(CY_SYS_CLK_HFCLK_IMO);
            }
            /*C17. Application components need IMO clock */
            else if(applicationPower == SLEEP )
            {
                /* C18. Put the system into Sleep mode*/
                applicationPower = WAKEUP_SLEEP;
                CySysPmSleep();
            }
        }

        /* Enable interrupts */
        CyExitCriticalSection(interruptStatus );

    }
```

3 Example Projects

Two example projects are included with this application note to show the implementation of low-power mode techniques described in the Implementing Low-Power Operations in Application Code section. These example projects require the BLE Pioneer Kit and the following software to build and test:

- PSoC Creator 3.3 or later with PSoC Programmer 3.23.0 or later
- CySmart™ PC application

3.1 Example Project 1: Low-Power Modes in Advertisement

This project shows how to implement low-power modes in a PSoC 4/PRoC BLE device during the advertising state. You can also use this project to configure and measure the device's current consumption for different advertising intervals. Refer to Appendix A: Advertising State Current Profile for details on how the low-power mode transitions occur when the device is in the advertising state.

The project configures the device in the Peripheral role with the default settings shown in Table 4. The advertising interval can be set per your requirement. The Central device sleep-clock accuracy is assumed to be 0 ppm to 20 ppm. The IMO clock is 16 MHz in the example, but it can be lower in real applications.

Table 4. Advertisement Settings

GAP role	Peripheral
Advertising type	Nonconnectable advertising
IMO clock	16 MHz
Transmit power	0 dBm
WCO clock accuracy	0–20 ppm
ADV packet length	14 bytes

The default clock settings used for the project are listed in Table 5.

Table 5. Clock Settings

IMO	Enabled
ECO	Enabled
Direct_Sel	IMO (16 MHz)
DBL_Sel	
PLL_Sel	
SYSCLK Divider	1
WCO	Enabled
WCO Power Mode	Low Power
LFCLK	WCO (32 kHz)

After you build and program the project into the kit, measure the current and capture the current profile for analysis. You can modify these configuration parameters and choose the optimum parameters according to your application.

Note: The cy_boot Component in the project should be updated to version 5.3 or later.

3.2 Example Project 2: Low-Power Modes in Connection

This example project shows you how to implement low-power modes in a PSoC 4/PRoC BLE device in a Peripheral role. The project can also be used to measure the current consumption of the device for various connection intervals. Refer to Appendix B: Connection State Current Profile for details on how the low-power mode transitions occur in the connection state.

The project uses the default connection settings listed in Table 6. The Central-side sleep-clock accuracy is assumed to be 0 ppm to 20 ppm.

Table 6. Connection Settings

GAP role	Peripheral
Connection interval	100 ms
Slave latency	0
IMO clock	16 MHz
WCO clock accuracy	0-20 ppm
Remote device sleep clock accuracy	0-20 ppm
Transmit power	0 dBm

The default clock settings used in the project are shown in Table 7.

Table 7. Clock Settings

IMO	Enabled
ECO	Enabled
Direct_Sel	IMO (16 MHz)
DBL_Sel	
PLL_Sel	
SYSCLK Divider	1
WCO	Enabled
WCO Power Mode	Low Power
LFCLK	WCO (32 kHz)

After you build and program the project into the kit, you can measure the current and capture the current profile for analysis. You can also modify these configuration parameters and choose the optimum parameters for your design.

Note: The cy_boot Component in the project should be updated to version 5.3 or later.

3.3 Average Current Measurement

The example projects can be used to measure the average current consumption and observe the current profile.

The instantaneous current consumed by the device is not a steady value but varies depending on the state of the chip that dynamically changes with the power-mode transitions. Therefore, it is practically impossible to measure each individual instantaneous current with a handheld multimeter because the duration of these current bursts is very small. Therefore, you should use a multimeter that provides the option to set the “aperture” of the measurement. The aperture is the period “T” during which the multimeter measures the instantaneous currents, integrates them, and then displays the average current for the period “T”. For accurate measurements, the aperture of the multimeter must be set to be the same as the advertising or the connection interval.

To measure the current, follow this procedure:

1. Build the chosen example project provided with this application note and program the binary into the BLE Pioneer Kit.
2. Connect a multimeter across jumper J15 of the BLE Pioneer Kit. A multimeter such as the Keysight 34410A digital multimeter should be used.
3. Set the aperture for the measurement based on the advertising interval or connection interval. If the exact aperture is not available, then set it to an integral multiple of the advertising or connection interval.
4. Measure the current. The current measured is the average current over one interval.

Note For advertising/connection intervals where the aperture is more than that supported by the instrument, a different method is required to measure the average current. For example, the Keysight 34410A multimeter has a maximum one-second aperture. Measuring the average current for intervals greater than one second requires a different approach. For these cases, set the integration method on the instrument to Number of Power Line Cycles (NPLC) and set the NPLC number to 100. Enable the “stats” option in the “math” menu. This will run an averaging filter on the current samples measured. After allowing a few samples (approximately 100x interval time), the value settles to a stable value, which is the average current.

3.3.1 Average Current in Advertising and Connection States

Table 8 and Table 9 list the average currents measured for different values of advertising and connection intervals using the example projects. The remote device used is another BLE Pioneer Kit. You can also use the kit USB dongle with the CySmart PC tool for connection-related measurements. The currents measured will be higher, however, due to the higher WCO clock inaccuracy of the dongle WCO crystal.

Table 8. Advertising Interval Versus Average current

Advertising Interval (ms)	CY8C4XX7-BL Current (µA)	CY8C4XX8-BL Current (µA)
100	261.8	276.06
120	219.5	231.0
150	175.6	184.8
180	147.0	154.6
200	132.3	139.1
250	106.4	111.8
300	88.8	93.43
400	66.5	70.26
500	53.3	56.61
1000	26.6	29.05
4000	7.5	8.32

Table 9. Connection Interval Versus Average Current

Connection Interval (ms)	CY8C4XX7-BL Average Current (µA)	CY8C4XX8-BL Average Current (µA)
10	1501.3	1691.0
20	783.6	861.1
30	522.7	574.1
40	389.0	428.5
50	313.4	343.9
60	264.2	289.3
70	224.9	246.7
80	197.3	216.2

Connection Interval (ms)	CY8C4XX7-BL Average Current (µA)	CY8C4XX8-BL Average Current (µA)
90	175.4	192.48
100	157.2	173.2
200	80.3	87.85
300	53.9	58.84
400	41.0	44.6
500	33.2	36.2
600	27.9	30.47
700	24.3	26.44
800	21.4	23.2
900	19.3	20.89
1000	17.6	19.03
4000	5.8	6.21

Note: For 256-KB flash devices (CY8C4XX8-BL), the current consumption will be 5% - 10% higher than the current consumption for the 128-KB flash variants (CY8C4XX7-BL) due to the increase in Deep Sleep and Sleep currents.

Figure 7 and Figure 8 show graphs of the measured current values for different advertising and connection intervals, respectively. The graphs can be used to get a quick estimate of the average current for any interval in the range.

Figure 7. Average Current Versus Advertising Interval

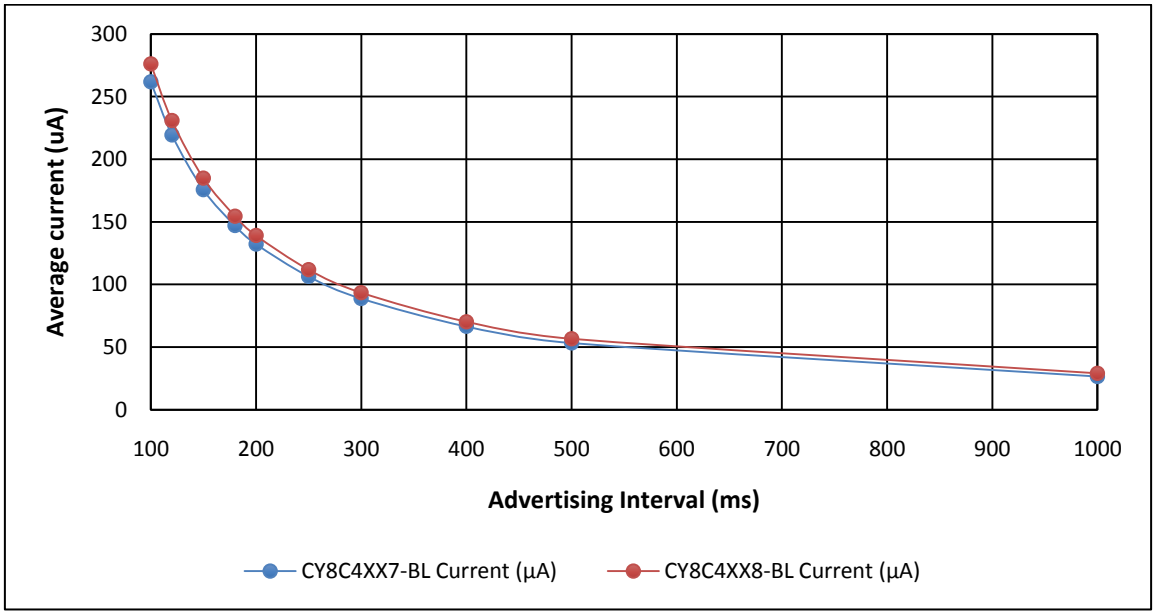
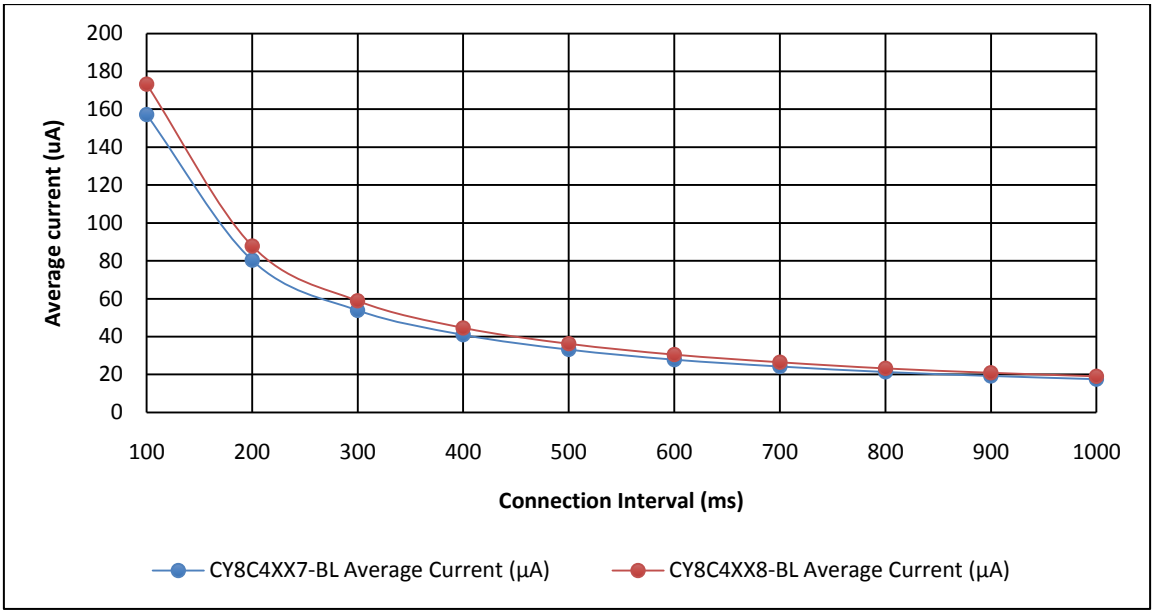


Figure 8. Average Current Versus Connection Interval



3.3.2 Current Profile for Advertising and Connection States

To observe the current profile described previously for the advertising and connection events, follow this procedure.

1. Connect a current probe such as Tektronix TCP0030 across the jumper wire on the J15 connector.
2. Connect the probe to a high-performance oscilloscope such as Tektronix DPO 4054.
3. Set the range of resolution for the current axis between 2 mA and 5 mA and the range of resolution for the time axis between 500 µs and 800 µs.
4. Trigger the oscilloscope to capture the waveform.

Figure 9 and Figure 10 show the current profile oscilloscope captures for the advertising interval and connection interval of 100 ms each, respectively. The letters marked in the capture map to the different power stages the device goes through during the advertising and connection states, as described in Appendix A: Advertising State Current Profile and Appendix B: Connection State Current Profile.

Figure 9. Current Profile Scope Capture for Advertising Interval

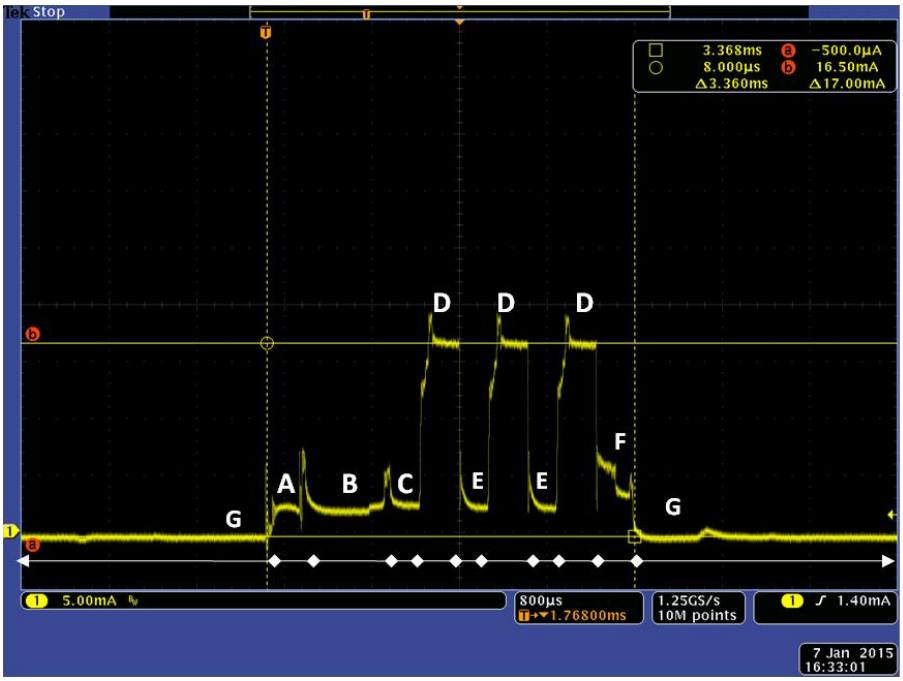


Figure 10. Current Profile Scope Capture for Connection Interval



3.4 Power Calculator

To make it easy to estimate the current consumption for your design, a power calculator based on an Excel worksheet is provided with this application note. This tool allows you to calculate the average current consumption for PSoC 4/PRoC BLE devices for the advertising and connection states of different system settings and BLESS parameters. You can use this tool during the system design stage to derive the optimum set of parameters to achieve the required low current consumption. Note that the calculator provides only an estimation of the current consumption. The actual current measured in your system may be different depending on your system design and how you use the various low-power modes.

The workbook has two calculators: the advertisement calculator and the connection calculator. In each calculator sheet, the user input parameters are entered in the yellow cells provided in the Input Parameters table. The calculated values are shown in the blue cells. The grey cells are for information purposes only. You are required to edit only the yellow cells and not modify the other cells. The allowed range of values for each input parameter is highlighted when you click on any input cell.

The advertisement calculator allows you to calculate the average current for different advertisement settings. The advertisement calculator input parameters are shown in Table 10. Select the advertising type, interval, and size of the data based on the configuration done in the BLE Component of your project. The CPU clock frequency can be 6, 12, or 16 MHz. You should also select the transmit power level configured in the BLE Component configuration.

Table 10. Advertisement Calculator Input Parameters

Table A - Input Parameters		
Advertisement Settings (From BLE Component Configuration	**Value**	**Units**
Advertisement Type	Unconnectable Advertising	
Advertisement Interval	1000	ms
Advertisement payload	14	bytes
System Settings		
CPU clock (IMO) frequency	16	MHz
Transmit Power	0	dbm
Estimate Average Current Consumption	**27.67**	uA

Based on the inputs, the sheet automatically updates the average current in the blue cell.

To estimate the battery life for an application such as iBeacon, where the device is in the Broadcaster role, select the battery capacity and the number of hours in a day the device is active. It is assumed that for the rest of the time, the device is in the OFF state and does not consume any power. Based on these inputs, the battery life is estimated, as given in Table 11.

Table 11. Advertisement – Battery Life Calculation

Table B - Battery Life Estimator		
hours of usage per day	24	hours
Battery Capacity	2200	mAh
Estimate Battery Life	**3313**	**days**

The connection calculator allows you to estimate the current consumption of a PSoC 4/PRoC BLE device in the connection state in a Peripheral role. The connection calculator input parameters are shown in Table 12. Select the connection interval and the slave latency based on the configuration done in the BLE Component of your project. The CPU clock frequency can be 6, 12, or 16 MHz. You should also select the crystal clock accuracy of the WCO crystal used in your design. The clock accuracy of the Central device, typically 031_TO_050_PPM, must also be entered. The transmit power level configured in the BLE Component is also selected.

Table 12. Connection Calculator Input Parameters

Table A - Input Parameters		
Connection Settings (From BLE Component Configuration)	**Value**	**Units**
GAP Role	Peripheral	
Connection Interval	1000	ms
Slave Latency	0	
Number of bytes to be sent	0	
System Settings		
CPU clock (IMO) frequency	16	MHz
WCO clock accuracy	000_TO_020_PPM	ppm
Remote Device clock accuracy	000_TO_020_PPM	ppm
Transmit Power	0	dbm
Estimate Average Current Consumption	**17.49**	uA

Based on the inputs, the sheet updates the average current in the blue cell.

4 BLE Applications

4.1 BLE System Power

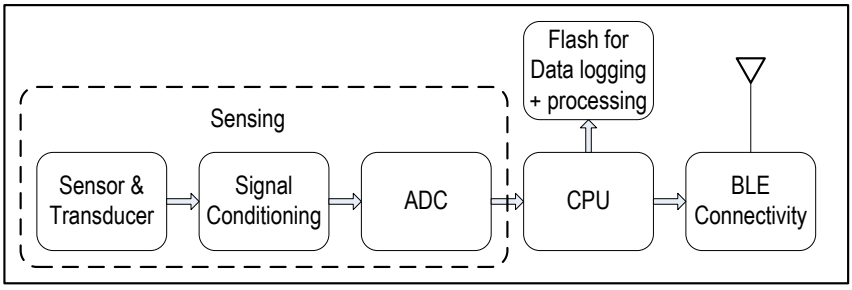
A typical BLE application involves sending the sensor data from a BLE Peripheral device (such as a heart-rate monitor) to a BLE Central device (such as a smartphone). The system-level architecture of a Peripheral device for such applications has three major functional blocks, as shown in Figure 11: sensing, data and event processing, and BLE connectivity. Each block contributes to the average current of the overall system.

4.1.1 Sensing Block

The sensing block comprises a sensor and transducer hardware that converts the sensor information into an analog electrical signal, followed by a signal-conditioning circuit that is used for filtering and amplification of the analog signal. The conditioned analog signal is converted into digital data for processing by an analog-to-digital converter (ADC). The key factors that determine the current consumption in the sensing operation are the following:

- Signal conditioning and analog-to-digital conversion: The signal-conditioning circuit and the ADC consume significant current during the sensing operation.
- Sampling rate: The sampling rate is the minimum rate at which the sensor data must be captured by the ADC to have enough samples for accurate processing of the sensor data.
- Scan rate: The scan rate is the rate at which a sensor must be polled to detect an activity. This allows the system to be in low-power states if there is no activity.

Figure 11. System Block Diagram



4.1.2 Data and Event Processing Block

The sensor data requires processing to derive meaningful and actionable information. You may also need to compress the data to reduce the over-the-air bandwidth required to transfer the data to the peer device.

Sometimes, a BLE device may use more than one sensor. For example, a heart-rate monitor device may measure the battery level in the device to detect a low-battery condition, in addition to heart-rate sensing. A BLE touch mouse may have an optical sensor, a touch sensor, a scroll key, a trackpad, and multiple buttons to detect key presses. In such systems, you need to detect activities in multiple sensors, process the data from multiple sensors, and send the data from multiple sensors over the BLE link. Each sensor's scan rate can be asynchronous to the others and the BLE link. Efficient management of these multiple asynchronous activities is an important factor that contributes to the current consumption.

4.1.3 BLE Connectivity Block

The sensor data is sent to a Central device through BLE notifications or through Read requests from the Central device. The time taken to process the notifications and send the sensor data to the peer device efficiently will determine the current consumption.

4.2 Battery Life

Most BLE devices are battery operated. A long battery life is typically the most important requirement for a battery-operated device. A long battery life helps reduce the maintenance cost of replacing the batteries often. It ensures high availability of the devices in the deployed environment. It also allows using a lower-capacity battery for a given lifetime of device usage, reducing the battery cost and allowing the devices to be smaller (See Table 13).

Table 13. CR Battery Comparison

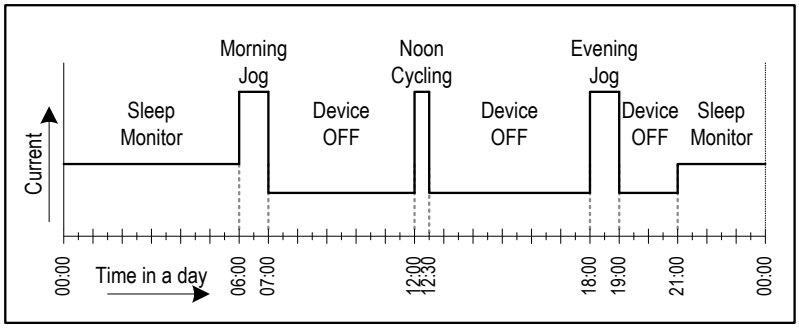
Type	Capacity (mAh)	Size (d x h, mm)
CR1025	30	10 × 2.5
CR1220	35–40	12.5 × 2.0
CR1616	50–55	16 × 1.6
CR1620	75–78	16 × 2.0
CR1632	140	16 × 3.2
CR2032	225	20 × 3.2
CR2450	610–620	24.5 × 5.0

The battery life is a function of the time that the device is used over the lifetime of the battery and the average current consumption of the system during its use. A device usage profile is the time pattern of a typical user using the BLE device through a normal day or a week—how long the device is used for the intended function, how long the device is idle, how long the device is switched OFF, and so on.

Based on the usage profile, the device can be put through different low-power states to reduce current consumption. The duration that the device is put in a specific state is determined by the usage profile and the application requirements. The current consumed in each state is determined by the system architecture, low-power modes used, and the current consumed by the different parts of the system in that state. Therefore, it is important to determine the usage profile and a current profile in various system states to estimate the battery life.

An illustration of the usage and current profile for an activity-monitoring (sleep and activity levels) device is shown in Figure 12. A user uses this device through the day; the user may switch it OFF intermittently. The current consumed varies accordingly.

Figure 12. Usage and Current Profile



The battery life can be measured by a simple formula that combines the usage profile and the current profile, as shown in Equation 1:

Equation 1. Battery Life

$$Battery\ Life, L\ hours = \frac{Battery\ Capacity(mAh)}{Average\ Current, Iav(mA)}$$

$$Iav = \frac{\{\sum_{i=0}^{n} Ii * Ti\}}{\sum_{i=0}^{n} Ti}$$

Where,

$Iav = Average\ Current\ (mA)$

$n = number\ of\ different\ device\ states$

$Ii = Average\ Current\ in\ state\ i$

$Ti = Time\ spent\ in\ state\ i$

The current in any state is a function of the current consumed in the different blocks in that state. Broadly speaking, this is the sum of the current consumed in the three major blocks, shown as:

Equation 2. Average current in a state

$$Is = Isense + Iprocess + Ile$$

Where,

$Is = Current\ in\ any\ state$

$Isense = Current\ in\ sensing\ block(mA)$

$Iprocess = Current\ used\ for\ processing\ (CPU\ subsystem)(mA)$

$Ile = Current\ in\ BLE\ subsystem\ (mA)$

4.3 Techniques for Increasing Battery Life

From these equations, it is evident that battery life can be increased by using the following approaches to reduce the average current consumed:

- Reduce the Active mode current: The sensing and processing functions contribute equally, if not more, to the overall system current consumption. Therefore, you should take a system view and reduce the current consumption in all active operations such as sensing, data processing, and BLE transactions.
- Reduce the active time: Reduce the time spent in CPU or RF operations so that the system can be in the idle or OFF states more often. The current consumed in these idle or OFF states is as important as the current consumed during active operations. The current consumption in these states should be reduced by using the available low-power modes.

The following is a list of suggestions to reduce the average current in PSoC 4/PRoC BLE devices and effectively increase the battery life.

4.3.1 Reducing the Active Current

4.3.1.1 Operate at Lower CPU Clock Frequency

The application should be designed to allow the CPU to operate at lower clock frequencies to reduce the current. For example, power-optimal algorithms and implementations tuned to the CPU architecture should be used to reduce the processing power required for sensor data processing.

4.3.1.2 Shut Down Unused Resources

The application can put the unused peripherals and clocks into the available low-power modes permanently. In addition, peripherals and clocks that are not required often or that are required only in specific usage modes should be put into their lowest power modes, wherever possible.

For example, you can turn the ILO OFF for most BLE applications. The WCO can be used for all LFCLK requirements such as the watchdog timer or BLESS.

4.3.1.3 Utilize Chip-Level Integration

Integration of the sensing circuit and other external interfaces in a single chip reduces the overall system current consumption due to improved performance, reduced I/O switching, and communication overheads. PSoC 4 BLE is a highly integrated device that includes Cypress CapSense®, programmable analog with 12-bit ADC, four opamps with comparator mode, two low-power comparators that can operate in the Deep-Sleep mode, a programmable digital block, and up to 32 GPIOs multiplexed with different communication interfaces such as I^2C, SPI, and UART. This system integration must be utilized to reduce the external components that need to be present on a PCB for the application and improve the overall system-level current consumption.

4.3.1.4 Reduce Transmit Power

Most applications require 0-dBm RF transmit power. However, the transmit power can be lowered if the device is designed to work for ranges of less than 5 m. This reduces the RF current consumption. For example, by operating the device at –6 dBm, the transmit current can be reduced by 3 mA.

4.3.2 Reducing the Active Time

Schedule Activities Around the Connection Event

The application should align the sensing and data processing operations to the beginning or end of BLE connection events. The application can then complete all the processing and put the system into the Deep-Sleep mode along with the BLESS until the next BLE connection event. This is more power efficient than processing asynchronous to the BLE events, because it allows the system to be in the Deep-Sleep mode for longer times (refer to Figure 1).

However, note that the application activity should be avoided during the BLE events when the BLESS is transmitting or receiving packets. This reduces the peak current consumption that adversely impacts the battery life of some types of batteries such as coin-cell batteries.

Lower Sensor Scan Rates

If no sensor use or activity is detected for some time, the device can reduce the rate at which it scans for the activity. In addition, it can increase the connection interval of the BLE link and enter low-power modes to reduce the system current. While this may introduce latency in reacting when an activity occurs, the overall system power can be significantly reduced. For example, in trackpad designs, scanning alternate sensors instead of all the sensors to detect a movement can save power. The sampling rate for data can also be minimized where and when possible.

Lower Data Sampling Rates

The sampling rate of the SAR ADC in the device should be reduced to the minimum frequency required to effectively reproduce the analog signal being sampled. Based on the sampling frequency, the SAR ADC configurations (such as the ICONT_LV register fields) that allow low-power operation should be used. In addition, the channel resolution can also be reduced to decrease the conversion time and thus save power.

Use Asynchronous Wakeup

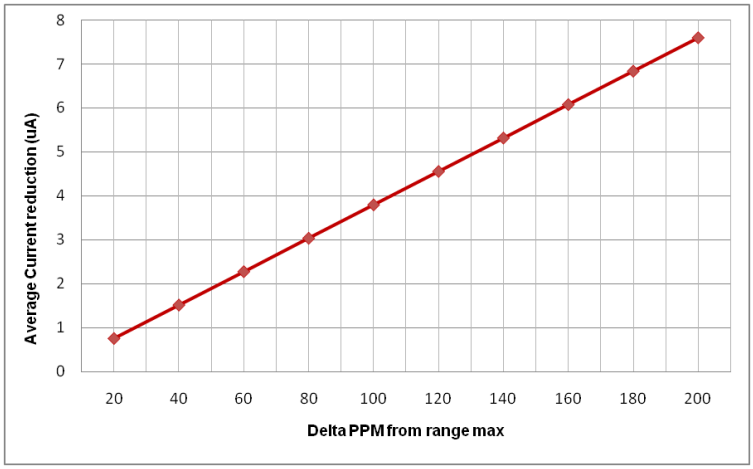
PSoC 4 BLE provides a low-power comparator that continues to operate while the system is in the Deep-Sleep mode. If any sensor activity is detected through the comparator, it can wake the entire system from the Deep-Sleep mode. This avoids keeping the CPU active for periodic scanning for sensor activity. In addition, the GPIOs are available as asynchronous wakeup sources, which can be used to wake up the system from the Deep-Sleep mode based on the detection of an external activity.

Reduce WCO Crystal ppm

The WCO crystal accuracy (measured in ppm) affects the listening window at the connection events in a Peripheral device. Due to the drift in timing caused by an inaccurate crystal, the Peripheral will have to listen for a larger window to “locate” the packet from the Central device. The higher the inaccuracy, the larger is the listening window and the higher is the current consumption. Refer to AN95089 – PSoC 4/PRoC BLE Crystal Oscillator Selection and Tuning Techniques for details on WCO crystal selection and tuning.

Figure 13 shows the improvement in average current for a one-second connection interval in PSoC 4 BLE as the ppm is reduced. A 500-ppm crystal is considered for the chart.

Figure 13. ppm Versus Current Reduction



By using a more accurate crystal, the active time for which the radio is listening can be reduced. For example, in the PSoC 4/ PRoC BLE device, reducing the WCO inaccuracy by 50 ppm reduces the average current by approximately 2 µA, thus increasing the battery life.

Increase Connection Interval

The connection interval for a BLE link is set by the Central device when a connection is created with the Peripheral device. It is preferable that the Peripheral have a connection interval that matches the rate at which the sensor data from the Peripheral is sent to the Central device. If the initial connection interval set by the Central is lower than necessary, then the Peripheral should request the Central to increase the connection interval to reduce current consumption.

Use Slave Latency

Slave latency is a BLE feature that allows a Peripheral to listen to the Central device on connection events at a reduced rate. This is useful if the Peripheral device is inactive for some time and has to wait for some activity to occur to send data to the Central device. If no activity is detected for some time, the Peripheral can enter slave latency by sending a connection update request to the Central. The Peripheral can extend the interval between the connection events at which it listens to the Central. This allows the BLESS and the system to be put into the Deep-Sleep mode for longer time. The key advantage of using slave latency is that when the Peripheral application detects an activity, it can switch the BLESS back ON to listen to the Central at the original connection interval until the data transfer is completed. This avoids the latency in sending the data upon the first activity.

For example, maintaining an idle connection with a 100-ms interval consumes 148 µA. If the same connection is updated with a slave latency value of 9, the current consumption drops to 17 µA.

These points are illustrated with two example BLE use cases: a fitness application and an HID application.

4.4 Example Application: Heart-Rate Monitor

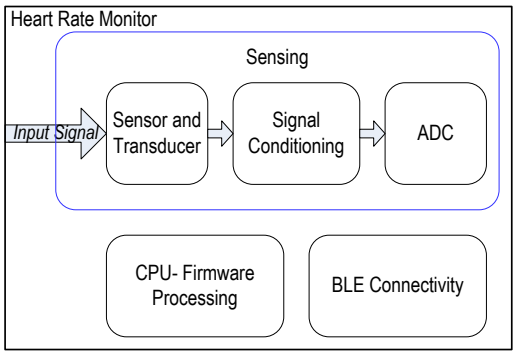
A common usage segment of BLE connectivity is the fitness and wearable segment. Many BLE devices track distance cycled, number of steps climbed, heart rate, number of hours slept, and so on. A heart-rate monitor (HRM) is one of the most common fitness applications that use BLE today to monitor fitness levels.

4.4.1 System Architecture

An HRM performs the following high-level activities (see Figure 14):

- Sensing: The HRM has an analog front end to capture signals from the human body. In general, it consists of one or more of the following activities:
  - Sensor: The sensor captures the input signal. Optical sensors (LED-photodiode pairs) and electrodes are commonly used.
  - Filtering and amplification: The input signal is filtered and amplified for accurate detection. Usually, operational amplifiers and hardware filters are used for this purpose.
  - Analog-to-digital conversion: This stage converts the analog signal to a digital signal, which is sent to the application firmware for further processing.
  - Firmware processing: The application implements further filtering and amplification, followed by an algorithm to convert the signal into a heart-rate value by using a timing procedure.
- BLE connectivity: The HRM maintains a BLE connection when in use and transmits the converted heart-rate value to a heart-rate collector device, usually with a refresh rate of once per second.

Figure 14. HRM System Block Diagram



4.4.2 HRM Usage Profile

An HRM is typically used in one of these ways.

1. Using the HRM during a training or exercise routine, generally for 1-2 hours per day.
2. Wearing the HRM all day in an ultra-low-power mode and turning it active multiple times during the day for short durations. The average active time is approximately one hour per day.

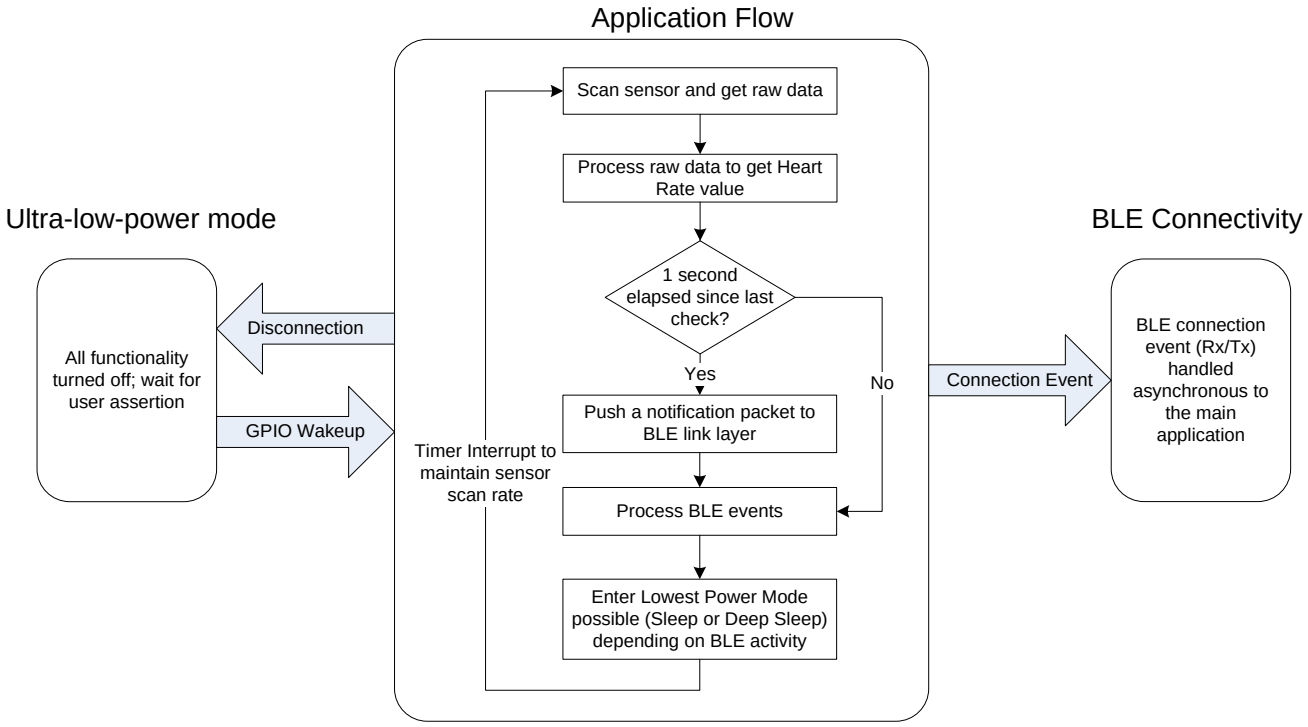
4.4.3 Application Design for Low Power

For the typical usage profile, the application design is shown in Figure 15. When the device is in use, it scans the heart-rate sensor at a constant rate and processes the sensor output to get a heart-rate value. For an HRM, the sensor scan rate is usually about 20 milliseconds. During the sensor scan, the system is put in the Sleep mode.

The BLE connection events are asynchronous to this operation, and the connection interval is much larger than the sensor scan rate. The typical connection interval is one second for HRM applications. At every connection event, the device sends the sensor data to the heart-rate collector device through notifications. The system then goes to the Deep-Sleep mode when all the BLE and system processing is completed.

When the user no longer needs the heart-rate information, the device disconnects from the peer device and enters an idle state with a very low current consumption (such as the Hibernate or Stop mode) where all the system functionality is turned OFF and a low-power wakeup source (such as a GPIO or the WAKEUP pin) is set up to resume functionality. For example, an external piezo can be used as a wakeup source by driving a GPIO. The piezo is activated and triggers an interrupt when the user wears the HRM. For this application, the SAR ADC and three CTBms available on the chip (PSoC 4 BLE only) are used to interface to the external sensor.

Figure 15. HRM Firmware Design



4.5 Example Application: Remote Control

An emerging application of BLE connectivity is in the HID market with products such as the smart remote control for smart TVs and set-top boxes. These applications differ from the wearable device applications mainly in the BLE data rate and the usage profile that involves start-idle-stop cycles. This example discusses a simple BLE remote control with a trackpad. It demonstrates an additional level of complexity in the system compared with the HRM and shows how a low-power design should be done in such a case.

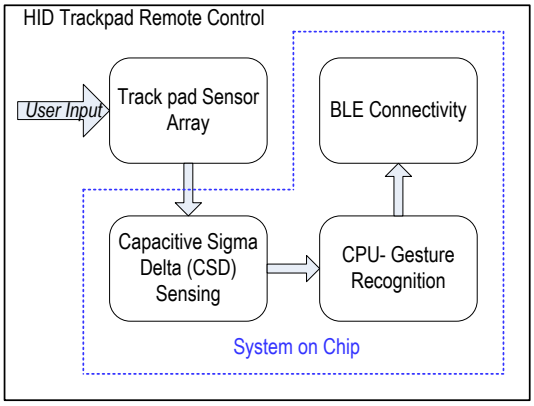
4.5.1 System Architecture

A BLE remote control with a trackpad as a BLE Peripheral device performs the following functions:

- Trackpad sensing: The touchpad of the remote device will have an array of sensor rows and columns. The number of rows and columns will vary depending on the size and resolution of the touchpad required. The trackpad sensor array implements capacitive sensing (CapSense). The CapSense block converts the capacitance of the sensor to a digital quantity, which can be processed by the firmware.
- Firmware processing: The sensor array data from the CapSense block is processed by the application firmware to detect mouse pointer movements and recognize gestures such as zoom, scroll, rotate, click, double-click, and so on. This data is sent as an HID report over BLE to the BLE Central device.
- BLE connectivity: The remote control device sends the HID reports to its host typically at a 10-ms connection interval whenever there is a user activity. However, in the absence of a user activity, this interval should be increased using slave latency to save power and maintain a BLE connection.

The overall system can be represented as shown in Figure 16.

Figure 16. Remote Control System Architecture



4.5.2 Remote Control Usage Profile

A typical usage model for the remote control follows:

- If the remote is used for 25 minutes a day, it will be in a low-power state for the rest of the day.
- Out of the total usage of 25 minutes, it is estimated that the user will actively use the remote with intermittent breaks when the remote is idle. The remote is assumed to be in active use for 80 percent of the time and idle for the remaining 20 percent of the time.

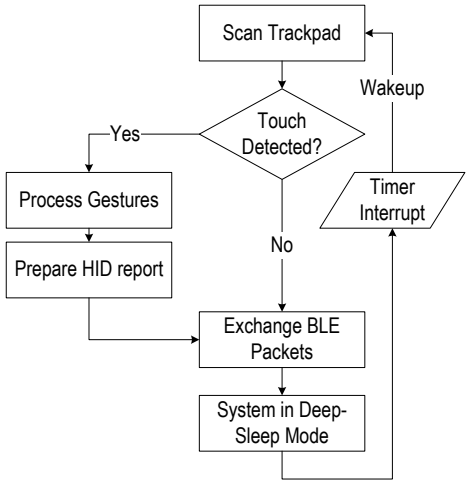
4.5.3 Application Design for Low Power

The activities that are carried out by the application are as follows:

- Scanning the trackpad
- Processing the trackpad data
- Sending the trackpad report over BLE
- Staying in a low-power mode for the rest of the time

A watchdog timer is used to interrupt the PSoC BLE/ PRoC BLE device periodically to scan the trackpad to detect user activity. A low current consumption can be achieved by keeping the system in a low-power mode between interrupts. If an activity is detected on the trackpad, the trackpad is scanned to process the finger movements and to send the data over BLE to the host device. Thus, the firmware of the remote implements a simple algorithm, as shown in Figure 17.

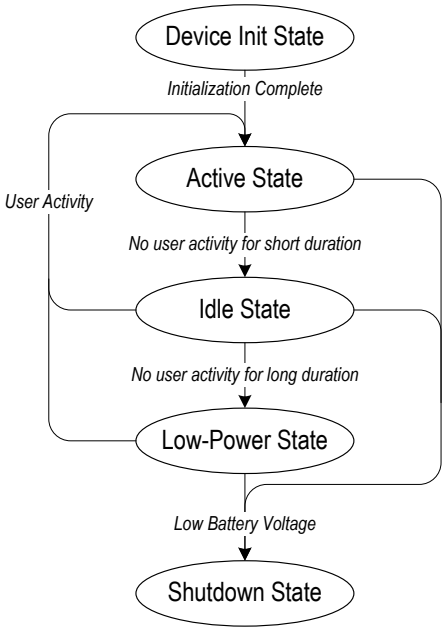
Figure 17. Remote Control Firmware Flowchart



The trackpad scan is aligned with the BLE connection events so the system can be in the Deep-Sleep mode for the maximum possible time. Scanning of sensors and processing of gestures take a finite amount of time every scan cycle. When these activities are completed at the connection event, the system is put into the Deep-Sleep mode to save power. The connection interval must be chosen appropriately. If the connection interval is higher, the HID report rate for the device may be lower than required, which will degrade the user experience due to latency in transmitting the data. If the connection interval is lower, then the current consumption increases. To satisfy the conflicting needs of a good user experience and a power-efficient system, the application needs to switch adaptively between the required connection intervals.

A similar challenge is faced with respect to the trackpad scan rate. The trackpad must be scanned at a higher rate to recognize gestures, but it should be scanned at a lower rate when there is no movement for a long time. The lower rate must be just enough to detect user activity. To meet these requirements, the application implements a state machine, as shown in Figure 18.

Figure 18. Remote Control State Machine



- **Active state:** The Active state denotes that all the scanning and processing activities are carried out at the highest required rate. Typically, the trackpad scan is done every 10 ms in the Active state. Therefore, most of the current consumption of the system originates in the Active state. The firmware will remain in the Active state as long as user activities are happening.
- **Idle state:** When using the trackpad, users may keep the trackpad idle for several seconds between touch activities. The Idle state saves power when the trackpad in not being used for an idle period. This state is entered if no touch activity is detected for a specified timeout in the Active state. In this state, the trackpad is scanned at a rate lower than needed for a smooth mouse movement, but sufficient to detect the start of a touch activity without an observable delay. The BLE connection uses slave latency to reduce the use of the radio, while maintaining readiness to send the data without latency when a user activity is detected. The system and BLESS are in the Deep-Sleep mode for most of the time.
- **Low-Power state:** The system is expected to stay in the low-power state for most of the time. The application enters this state from the Idle state if a user activity is not detected during the Idle state for additional time. In this state, a BLE connection is sustained with no application data exchange. The application may also increase the slave latency to reduce the use of the radio while sustaining the connection. The trackpad still needs to be scanned to detect user activity. To detect a touch activity, a high resolution of the touch is not needed. Therefore, only the alternate rows of the trackpad are scanned. The system is in the Deep-Sleep mode for most of the time.
- **Shutdown state:** When the battery voltage drops below the operable voltage, the system stops all the activities and shuts down. This is done by putting the system in the Hibernate or Stop mode.

It should be noted that in each of these states, the system undergoes power mode transitions among the Deep-Sleep, Sleep, and Active modes for optimizing power, based on the device activity. Only the durations for which the device remains in the Active or Deep-Sleep mode are different in different states.

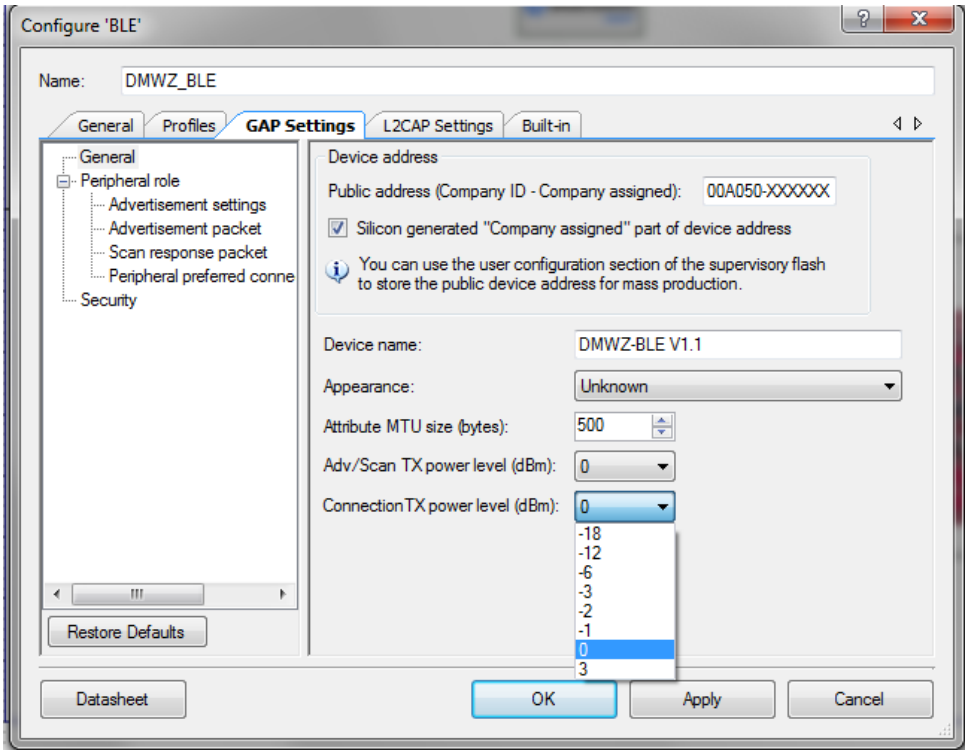
4.6 Implementing System Design Recommendations

4.6.1 RF Transmit Power

RF transmit power can be reduced to conserve power. To configure the RF transmit power in your project, open the **Configure BLE** window from the BLE Component instance, as shown in Figure 19.

1. Go to the **GAP Settings tab**.
2. Configure the **TX power level (dBm)** to the desired setting from the list.

Figure 19. Changing the RF Transmit Power Level



4.6.2 Connection Parameter Update

The PSoC 4/PRoC BLE device supports the connection parameter update feature of the BLE specification. The feature allows a Peripheral to negotiate the connection parameters, connection interval, and slave latency with the Central device as required by the application.

To update either the connection interval or the slave latency parameters, the application on the Peripheral device must call the `CyBle_L2capLeConnectionParamUpdateRequest()` function with the new connection interval and slave latency parameters. The function is called when the `CYBLE_EVT_GAP_DEVICE_CONNECTED` event is received upon device connection.

```
void AppCallBack(uint32 event, void* eventParam)
{
    static CYBLE_GAP_CONN_UPDATE_PARAM_T hrmConnectionParam =
    {
      1000,         /* Minimum connection interval required */
      1000,         /* Maximum connection interval required */
      0,           /* Slave latency */
      500         /* Supervision timeout */
    };

    switch(event)
    {
      /* Handle Stack events */
      case CYBLE_EVT_GAP_DEVICE_CONNECTED:
      /* Send Connection Parameter Update request to the Central */
      CyBle_L2capLeConnectionParamUpdateRequest \
            (cyBle_connHandle.bdHandle, &hrmConnectionParam);

    }
}
```

5 Summary

PSoC 4/PRoC BLE devices support multiple systems and BLESS low-power modes that enable you to design a highly power-efficient solution. The advertising and connection current profiles indicate that the Deep-Sleep mode current is equally important to the RF transmit/receive currents when considering a reduction in the overall average current. The advertising and connection power calculator can help you choose the right set of parameters.

The battery life of a BLE application depends on the system-level current consumption. In most applications, sensing and processing can consume more power than the BLE operations, so attention must be paid to reduce the overall system current. The system-level low-power techniques suggested should help you in designing a low-power BLE solution by including some suggestions, where possible, at the system design stage. You can also use the power calculator tool to help you make decisions on the system parameters to improve the battery life.

6 References

a. Bluetooth Low Energy – How does it achieve low power?

https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=227336

b. Example projects with Low Power Template

https://github.com/cypresssemiconductorco/BLE/tree/master/BLE Low Power Template Examples

A Appendix A: Advertising State Current Profile

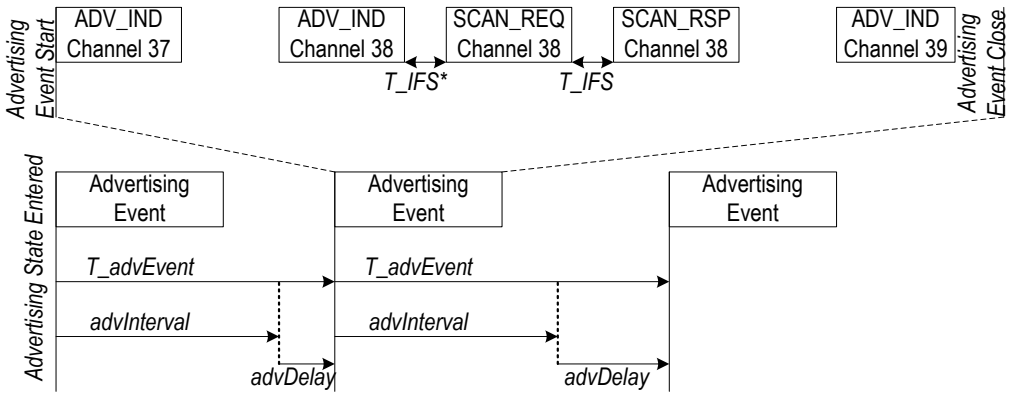
The PSoC 4/PRoC BLE device in the advertising state transmits packets containing the advertising data in periodic events. Three advertising packets may be sent in an advertising event (one on each of the three enabled advertising channels). The device may optionally listen for and respond to a peer BLE device that receives the advertising packets and requests for additional data.

The time between the start of the two consecutive advertising events (*T_adv_event)* is specified as follows:

T_adv_event = advInterval + advDelay

The *advInterval* is an integer multiple of 0.625 ms and has different ranges for different advertising packet types. It is typically chosen in the range 100 ms to 10.24 s. The *advDelay* is a pseudo-random value with a range of 0 ms to 10 ms. Advertising events are illustrated in Figure 20, with one advertising event as an example.

Figure 20. Advertising Event and Interval

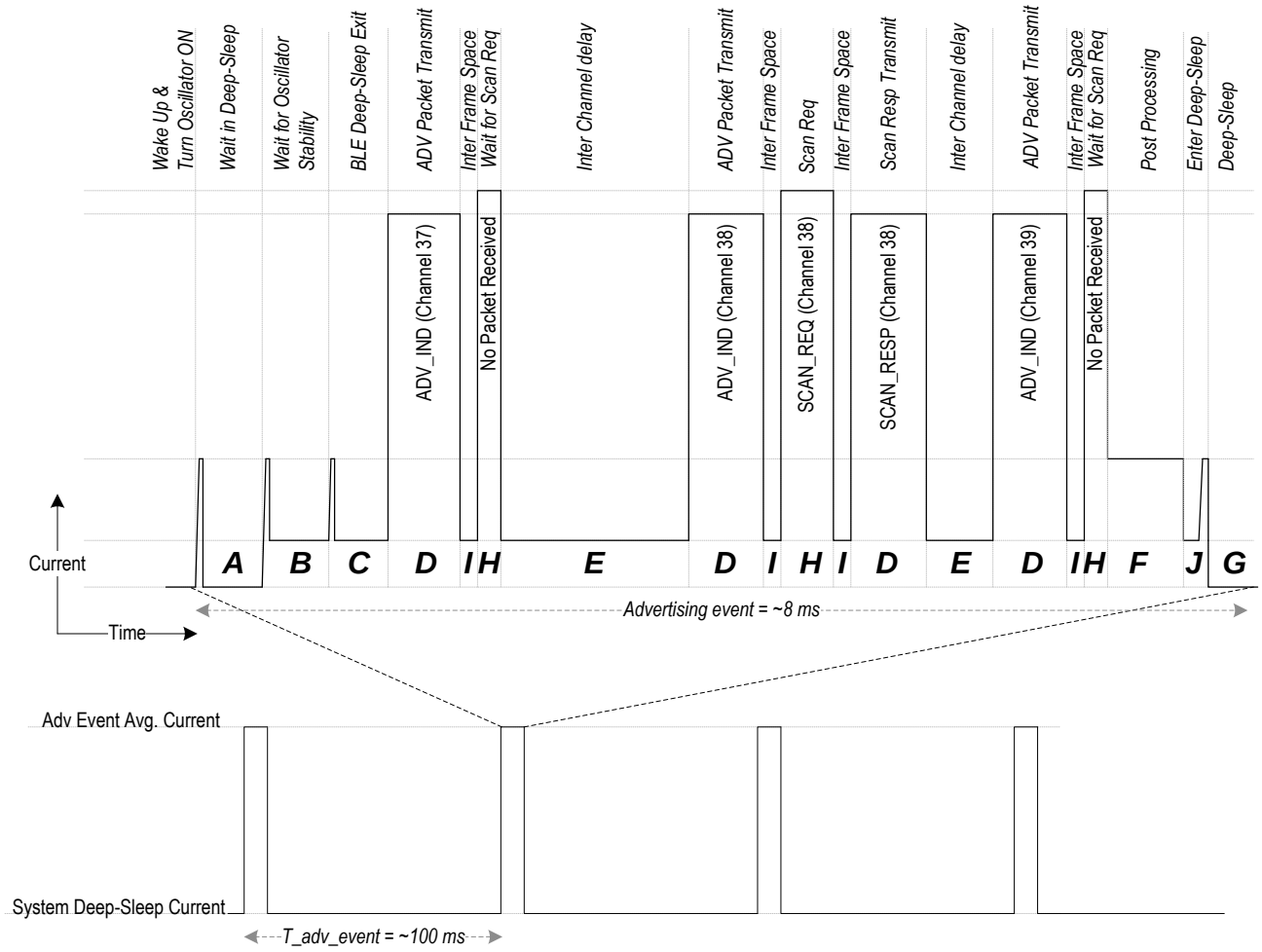


** T_IFS is Inter Frame Space, which is defined to be 150 µs. This is to allow the radio to switch between transmit and receive states.*

A.1 Current Profile

The advertising state current profile for a PSoC 4/PRoC BLE Peripheral device implementing the recommendations for low power is shown in Figure 21. It illustrates the power mode transitions, along with the BLE state transitions and relative current levels across the states.

Figure 21. Advertising Current Profile



The stages of the current profile curve, labeled with the letters A to H, are listed in Table 14 along with the BLESS internal states and the system and BLESS power modes used. Note that in some of these states, the system is shown to be in more than one power mode. This is because an interrupt at the beginning of the state transitions the system into the Active mode. After the interrupt is processed, the system is put back into a low-power mode.

Table 14. Advertising Current Profile Stages

Stage	BLESS Internal State	BLESS Operation	Power Modes	
			BLESS	System
A	CYBLE_BLESS_STATE_ECO_ON	Oscillator Startup	DEEPSLEEP	Active
				Deep-Sleep
B	CYBLE_BLESS_STATE_ECO_STABLE	Oscillator Stabilization	DEEPSLEEP	Active
				Sleep
C	CYBLE_BLESS_STATE_ACTIVE	Event Start Delay	ACTIVE	Active
				Sleep
D	CYBLE_BLESS_STATE_ACTIVE	ADV Transmit	ACTIVE	Sleep
I	CYBLE_BLESS_STATE_ACTIVE	Inter-Frame Space	ACTIVE	Sleep
H	CYBLE_BLESS_STATE_ACTIVE	Receive	ACTIVE	Sleep
E	CYBLE_BLESS_STATE_ACTIVE	Inter-Channel Delay	ACTIVE	Sleep
F	CYBLE_BLESS_STATE_EVENT CLOSE	Post-Processing	ACTIVE	Active
J	CYBLE_BLESS_STATE_DEEPSLEEP	Enter Deep-Sleep (takes two LFCLK cycles)	DEEPSLEEP	Sleep
G	CYBLE_BLESS_STATE_DEEPSLEEP	Deep-Sleep	DEEPSLEEP	Deep-Sleep

The BLESS is in the DEEPSLEEP mode (stage G) between advertising events. The system may also be in the Deep-Sleep mode if no processing is required. The BLESS maintains the link timing by using the WCO clock because the ECO is OFF. The BLESS automatically wakes up at the programmed instant before the start of the advertising event and generates an interrupt.

A – The BLESS interrupt wakes up the system from the Deep-Sleep mode. The BLE Component turns ON the ECO at the beginning of the stage. The ECO ramps up by the end of stage A with a stable clock amplitude. The ramp up takes approximately 400 µs. The application can put the system back into the Deep-Sleep mode until the ECO is ready. The ECO interrupts the CPU once the amplitude is stable.

B – The ECO requires more time to stabilize the frequency. The application puts the system into the Sleep mode during this period. This stabilization takes approximately 400 µs. At the end of stage B, the BLESS wakes up from the DEEPSLEEP mode. It switches from the WCO clock to the stable ECO clock and generates an interrupt.

C – The BLESS interrupt wakes up the CPU. The BLESS enters the ACTIVE mode. If no processing is required, the application puts the system again into the Sleep mode. The system may remain in the Sleep mode until all BLE transactions are completed in stage F. The system remains in this stage until the start of the advertising event.

D – In the advertising event, the Peripheral transmits the advertising packets. The entire BLESS including the RF is active during this period. The advertising packets are transmitted on the three advertising channels.

I – After transmitting the advertising packet, the Peripheral waits for an Inter-Frame Space time interval (T_IFS) of 150 µs (per the BLE specification) to listen to the peer device. This stage is optional and depends on the advertising type settings.

H – After an Inter-Frame Space time interval, the Peripheral listens on the same channel for a packet from the peer device. The BLESS times out and stops listening if no packet is received within a specific time. If a packet is received, it continues to receive until the end of the packet. This state is optional and depends on the advertising type settings.

E – The three packets are spaced at an interval of 1.25 ms. The BLESS is idle during this interval between the transmissions.

F – After transmitting the three advertising packets, the BLESS generates an “end-of-event” interrupt that wakes up the CPU. The BLE stack and application complete any event-specific or other application-specific activities in this period. The application then programs the BLESS to enter the DEEPSLEEP mode by calling the `CyBle_EnterLPM()` function in the BLE Component. The BLESS takes two LFCLK cycles (approximately 120 µs) to switch to the WCO and enter the DEEPSLEEP mode internally. The BLE Component therefore puts the system into the Sleep mode until the DEEPSLEEP mode entry is complete.

J – The system is in the Sleep mode in this state, waiting for the BLESS to enter the DEEPSLEEP mode. Once the transition is complete, the BLESS generates an interrupt to indicate successful entry into the DEEPSLEEP mode, which wakes up the CPU.

G – When the application receives the interrupt at the end of stage J, the application puts the system also into the Deep-Sleep mode no processing is required.

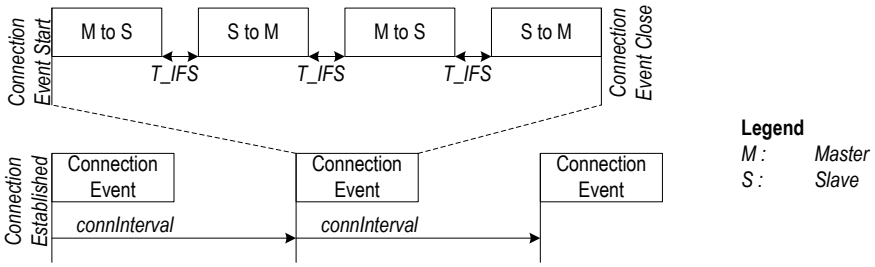
Note: In the profile described, it is assumed that the application is only managing the BLE advertising events and not handling any system-level tasks and interrupts. Additional system-level processing will change the current profile depending on the implementation.

B Appendix B: Connection State Current Profile

A connection is set up between two BLE devices: one acting as a Central (referred to as the Master at the Link Layer) and the other as a Peripheral (referred to as a Slave at the Link Layer) for exchanging application data. The Central and Peripheral devices exchange data by transmitting and receiving packets at connection events. The Central device starts the connection event by transmitting first. The devices then alternate transmitting and receiving data until they stop the exchange and close the connection event. The starts of connection events are spaced with a periodic interval of *connInterval*. The *connInterval* is a multiple of 1.25 ms in the range of 7.5 ms to 4 s, per the BLE specification.

In addition to the connection interval (*connInterval)*, the *connSlaveLatency* parameter defines the number of consecutive connection events that the Peripheral device is not required to listen to the Central device. These parameters together determine the timing for connections. A BLE connection event with zero slave latency is illustrated in Figure 22.

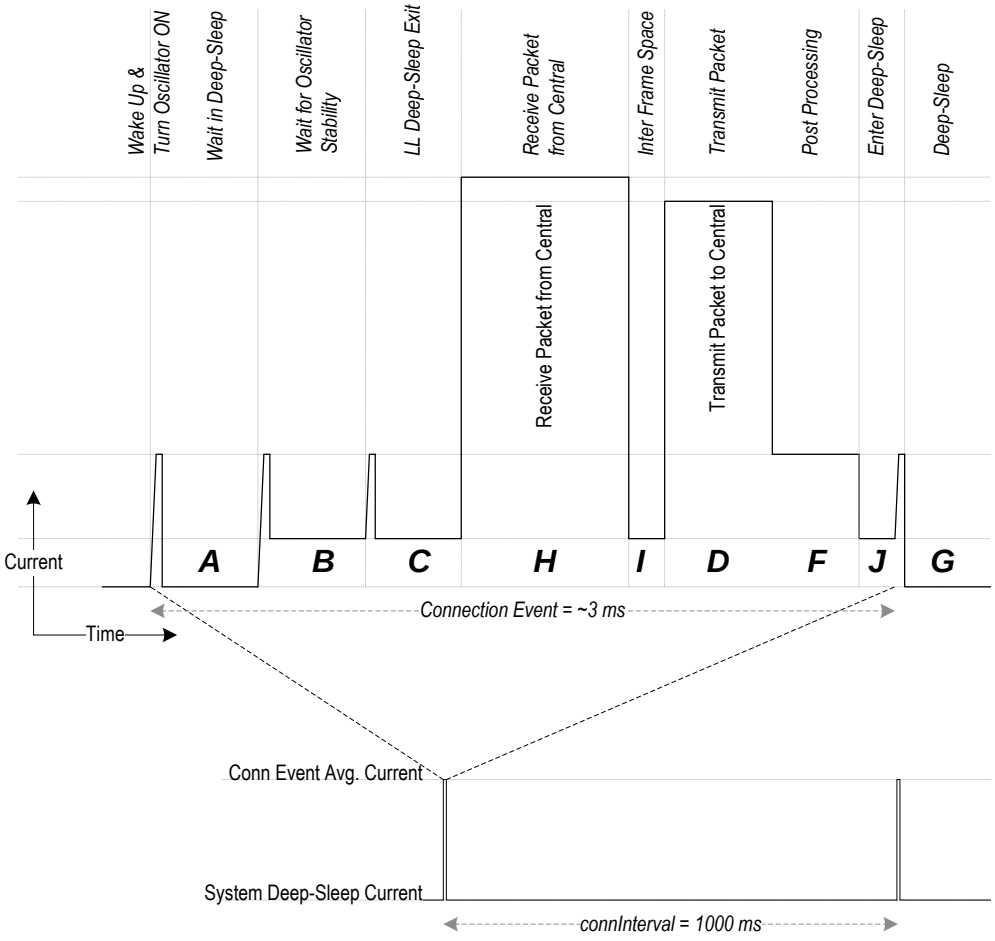
Figure 22. Connection Event and Interval



B.1 Current Profile

A connection state current profile for a PSoC 4/PRoC BLE Peripheral device considering this approach is shown in Figure 23. The figure illustrates the power modes used, along with the BLESS internal state transitions and relative current levels.

Figure 23. Peripheral Connection Current Profile



The stages of the current profile, labeled using the letters A to I, along with the BLESS internal states, BLESS power modes, and the system power modes are tabulated in Table 15.

Table 15. Connection Current Profile States

Stage	BLESS Internal State	Connection Profile State	Power Modes	
			BLESS	System
A	CYBLE_BLESS_STATE_ECO_ON	Oscillator Startup	DEEPSLEEP	Active
				Deep-Sleep
B	CYBLE_BLESS_STATE_ECO_STABLE	Oscillator Stabilization	DEEPSLEEP	Active
				Sleep
C	CYBLE_BLESS_STATE_ACTIVE	Event Start Delay	ACTIVE	Active
				Sleep
H	CYBLE_BLESS_STATE_ACTIVE	Receive	ACTIVE	Sleep
I	CYBLE_BLESS_STATE_ACTIVE	Inter-Frame Space	ACTIVE	Sleep
D	CYBLE_BLESS_STATE_ACTIVE	Transmit	ACTIVE	Sleep
F	CYBLE_BLESS_STATE_EVENT CLOSE	Post-Processing	ACTIVE	Active
J	CYBLE_BLESS_STATE_DEEPSLEEP	Enter Deep-Sleep (takes two LFCLK cycles)	DEEPSLEEP	Sleep
G	CYBLE_BLESS_STATE_DEEPSLEEP	Deep-Sleep	DEEPSLEEP	Deep-Sleep

The BLESS is in the DEEPSLEEP mode (stage G) between the connection events. The system may also be in the Deep-Sleep mode if no processing is required. The BLESS maintains the link timing by using the low-frequency WCO because the ECO is OFF. The BLESS automatically wakes up at the programmed instant before the start of the connection event and generates an interrupt to wake up the system from the Deep-Sleep mode.

Stages A, B, and C in the connection current profile remain the same as that for the advertising current profile.

H – The device first listens to a packet from the Central device. The duration of this first listening window depends on the drift caused by clock inaccuracies (measured in ppm) of the crystal oscillator (WCO in this case) used by the Central and Peripheral devices between the events.

I – After receiving a packet, the Peripheral waits for an Inter-Frame Space time interval (T_IFS) of 150 µs (per the erroBLE specification).

D – After waiting for T_IFS, the Peripheral transmits a response packet. The entire BLESS including the RF is active during this period.

F – After transmitting the three advertising packets, the BLESS generates an “end-of-event” interrupt that wakes up the CPU. The BLE stack and application complete any event-specific or other application-specific activities in this period. The application then programs the BLESS to enter the DEEPSLEEP mode by calling the `CyBle_EnterLPM()` function in the BLE Component. The BLESS takes two LFCLK cycles (~120 µs) to switch to the WCO and enter the DEEPSLEEP mode internally. The BLE Component therefore puts the system into the Sleep mode until the DEEPSLEEP mode entry is complete.

J – The system is in the Sleep mode in this state, waiting for the BLESS to enter the DEEPSLEEP mode. Once the transition is complete, the BLESS generates an interrupt to indicate successful entry into the DEEPSLEEP mode, which wakes up the CPU.

G – When the application receives the interrupt at the end of stage J, the application puts the system also into the Deep-Sleep mode, if no processing is required.

Note: In the profile described, it is assumed that the application is only managing a BLE connection event and not handling any system-level tasks and interrupts. Additional system-level processing will change the current profile depending on the implementation.

Document History

Document Title: AN92584 – Designing for Low Power and Estimating Battery Life for BLE Applications

Document Number: 001-92584

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4701892	KMVP	03/26/2015	New Application Note
*A	4765378	KMVP	05/14/2015	Updated Associated Part Family
*B	5050946	SASD	12/14/2015	1. Added section 6 ‘References’ 2. Updated snapshots for latest PSoC creator in section 2.5 ‘Low-Power Implementation’ 3. Updated Table-8 and Table-9 with CY8C4XX8-BL Average Current 4. Added note on the estimated current consumption for CY8C4XX8-BL in section 3.3 ‘Average Current Measurement’ 5. Updated Figure-7 and Figure-8 with CY8C4XX8-BL Average Current
*C	5171215	ASPV	03/14/2016	Updated to new template.
*D	5687926	BENV	03/19/2017	Updated logo and copyright

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2015-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.