CE220960 – PSoC 6 MCU BLE Upgradable Stack Bootloader

Objective

This example demonstrates over the air (OTA) bootloading with a PSoC® 6 MCU with Bluetooth Low Energy (BLE) connectivity. The BLE stack code is shared between applications to reduce flash usage. The bootloader may download updates to the BLE stack or to the application.

Overview

This example demonstrates how to use the "Stack and Profile" and "Profile only" options of the BLE Component to enable code sharing of the BLE stack between the stack and the user application, reducing considerably the amount of flash memory used by the applications. Additionally, it demonstrates an architecture that allows upgrading the BLE stack for future-proofing.

Requirements

Tool: PSoC Creator™ 4.2; Peripheral Driver Library (PDL) 3.0.1 with Bootloader SDK 2.10; CySmart 1.2.1.711

Programming Language: C (Arm® GCC 5.4.1 and Arm MDK 5.22)

Associated Parts: All PSoC 63 MCU BLE parts

Related Hardware: CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit

Hardware Setup

Set the VDD Select Switch (SW5) of the CY8CKIT-062-BLE kit to 3.3 V to fully use the RGB LED.

For BLE communications, the BLE USB dongle (CY5677) provided with the CY8CKIT-062-BLE kit is required.

Software Setup

Install the latest CySmart software on your computer to use the BLE USB dongle.

Operation

The bootloader can download a user application (App2), a stack application (App1) update, or both. Additionally, the bootloader can transfer control to a previously downloaded user application. Table 1 shows a list of the steps to be taken depending on the development phase of your project. When developing, it is faster to reprogram the device than to bootload an update.

Table 1. Operation Section Order

Development and Testing	Field Update
Develop applications: 1. Program the PSoC 6 MCU Device 1.1. Test applications Test Bootloader functionalities: 2. Configure CySmart 3. Switch to the Bootloader 4. Update the Stack and User Applications 5. Switch to the Bootloader 6. Switch to the User Application	Deploy an update: 1. Configure CySmart 2. Switch to the Bootloader 3. Update the Stack and User Applications 3.1. Updating the User Application 3.2. Updating the Stack and User Application

Program the PSoC 6 MCU Device

1. Plug the CY8CKIT-062-BLE kit board into your computer's USB port.
2. Build the projects in the order App0, App1, and App2. Any change to App0 or App1 requires subsequent projects to be rebuilt. For more information on how to build a project or program a device, see PSoC Creator Help.

Note: In some cases, you may be prompted to replace files from your project with files from the PDL. These files are templates. Do not replace the customized files for the project. Click **Cancel**.

3. Set App2 as the active project and program it into the PSoC 6 MCU. When App2 is built, App0 and App1 are merged into App2. This results in all apps being programmed. Similarly, programming App1 results in App0 and App1 being programmed.
4. Confirm that the kit LED blinks green, indicating that App2 is running.

Configure CySmart

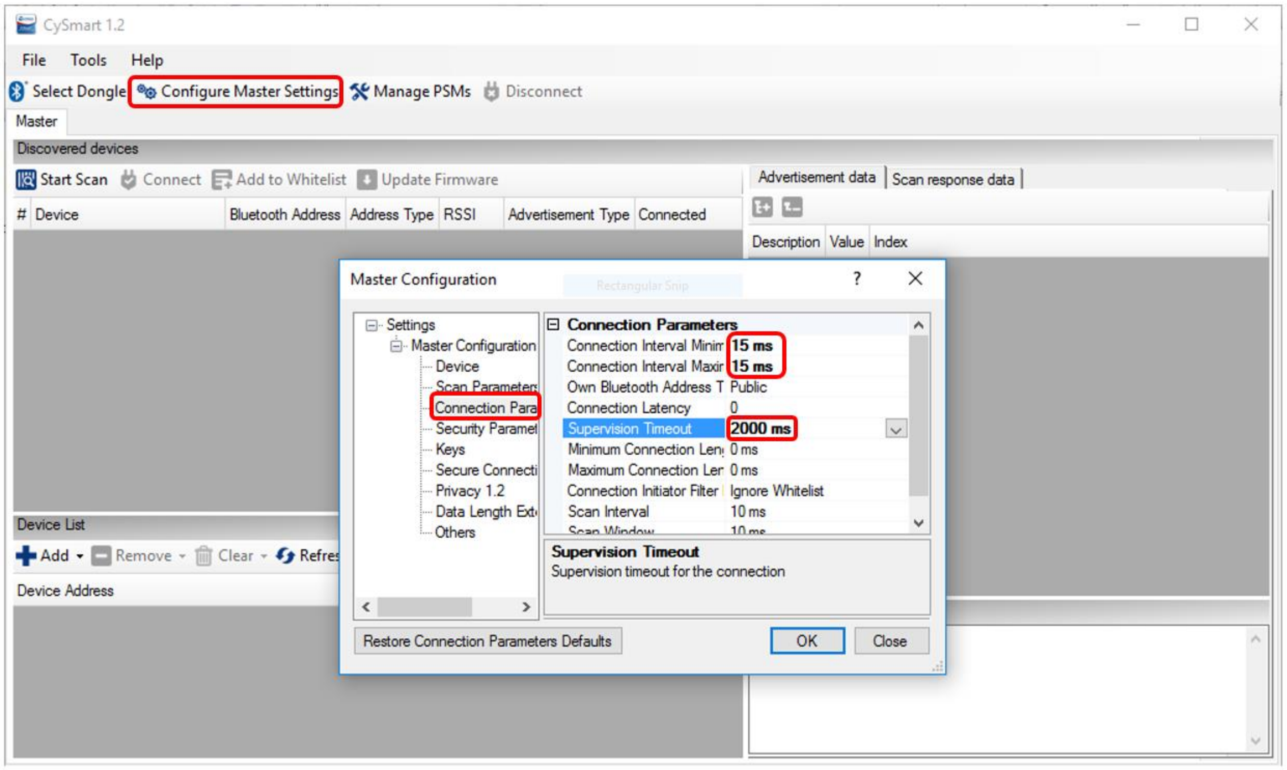
The default Bluetooth connection interval settings of the CySmart PC tool must be increased to allow enough time for flash memory write operations during the bootloading process. This section explains how to change the default settings.

1. Connect the BLE USB dongle (CY5677) provided with the CY8CKIT-062-BLE kit to your computer.
2. Run the CySmart tool on your computer and connect to the BLE USB dongle.
3. Click **Configure Master Settings**.
4. Go to **Connection Parameters**. Change the **Connection Interval Minimum**, **Connection Interval Maximum**, and **Supervision Timeout** to 15, 15, and 2000 ms, respectively, as Figure 1 shows. Click **OK**.

Note: Using a lower connection interval speeds up the application transmission at an increased risk of losing the connection.

These steps must be redone every time CySmart is reopened.

Figure 1. CySmart Connection Interval Settings



Connection Parameters

Connection Interval Minim	15 ms
Connection Interval Maxir	15 ms
Own Bluetooth Address T	Public
Connection Latency	0
Supervision Timeout	2000 ms
Minimum Connection Len	0 ms
Maximum Connection Ler	0 ms
Connection Initiator Filter	Ignore Whitelist
Scan Interval	10 ms
Scan Window	10 ms

Supervision Timeout
Supervision timeout for the connection

Switch to the Bootloader

To receive an update either to the user application or to the stack application, the user application must transfer control to the stack application, which contains the bootloader. The following methods are provided to switch to the bootloader.

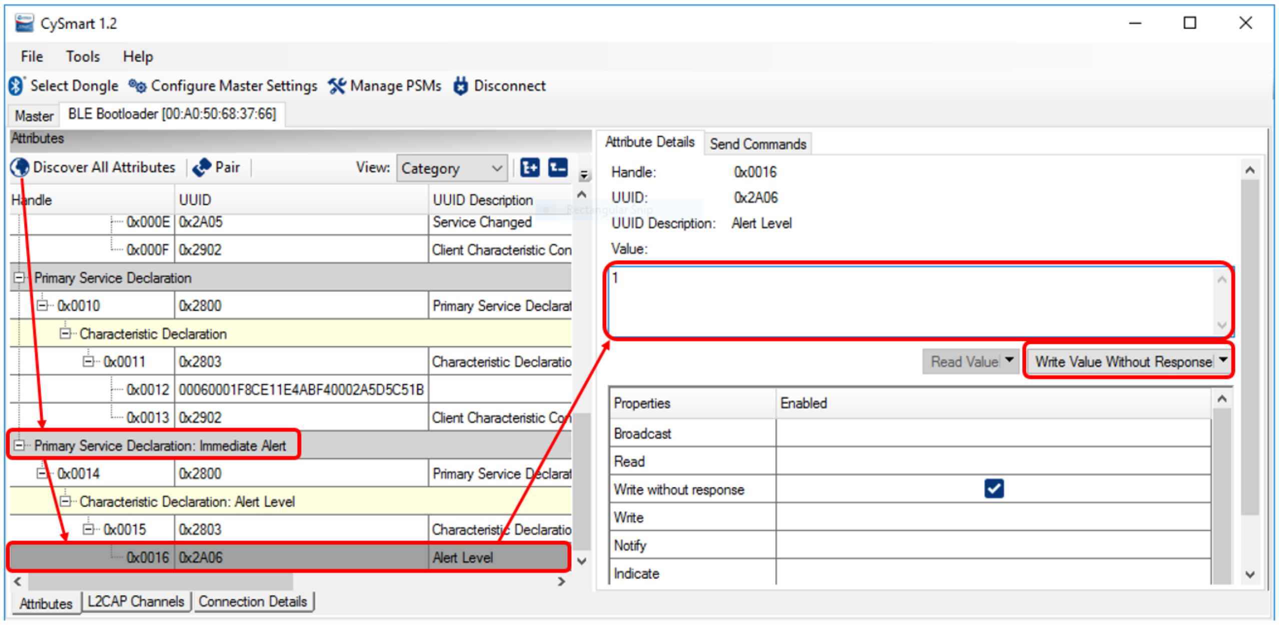
Switching Using IAS

The user application switches to the stack application when it receives a nonzero Immediate Alert Service (IAS) alert level. If the CySmart PC tool is not already running, do the steps described in Configure CySmart.

1. Press and release the kit user button (SW2) if the PSoC 6 MCU device is hibernating (indicated by a steady red LED).
2. In CySmart, click **Start Scan** to start scanning for the user application. In this example, the user application is the “BLE Keyboard” device. When the device is listed, select it.
3. Click **Connect** to connect to the device.
4. Click **Pair** to pair with the device. The Pair button may be hidden if the window size is small.

   Note: If pairing fails, disconnect from the device and clear the device list in the CySmart tool. Go back to Step 2.
5. Click **No** when prompted to add the device to the resolving list.
6. Click **Discover All Attributes**.
7. Navigate to the **Immediate Alert** service at the bottom of the attributes list. Click **Alert Level**. Enter **1** in the **Value** textbox and click **Write Value Without Response**, as Figure 2 shows.

Figure 2. Using the IAS to Switch between Applications



Handle	UUID	UUID Description
0x000E	0x2A05	Service Changed
0x000F	0x2902	Client Characteristic Con
Primary Service Declaration		
0x0010	0x2800	Primary Service Declarat
Characteristic Declaration		
0x0011	0x2803	Characteristic Declaratio
0x0012	00060001F8CE11E4ABF40002A5D5C51B	
0x0013	0x2902	Client Characteristic Co
Primary Service Declaration: Immediate Alert		
0x0014	0x2800	Primary Service Declarat
Characteristic Declaration: Alert Level		
0x0015	0x2803	Characteristic Declaratio
0x0016	0x2A06	Alert Level

Handle: 0x0016
UUID: 0x2A06
UUID Description: Alert Level
Value:
1

Properties	Enabled
Broadcast	
Read	
Write without response	✓
Write	
Notify	
Indicate	

8. Confirm that the LED blinks white once every two seconds, indicating that the bootloader is running.

Switching Using the User Button and a Hardware Reset

App0 starts after a hardware reset. It supervises the user button and switches to the stack application if the button is pressed.

1. Press the reset and user buttons at the same time.
2. Release the reset button.
3. Wait until the LED starts blinking white before releasing the user button. The bootloader is now running.

Switch to the User Application

The stack application switches automatically to the user application after 300 seconds of Bluetooth inactivity. The following methods describe how to immediately transfer control to the user application.

Switching Using IAS

The stack application switches to the user application when it receives a nonzero Immediate Alert Service (IAS) alert level. If the CySmart PC tool is not already running, perform the steps described in Configure CySmart.

1. In CySmart, click **Start Scan** to start scanning for the bootloader device. Select the "Bootloader BLE" device when listed.
2. Click **Connect** to connect to the device.
3. Click **Pair** to pair with the device. The Pair button may be hidden if the window size is small.
4. Click **Discover All Attributes**.
5. Navigate to the **Immediate Alert** service at the bottom and click **Alert Level**. Enter **1** in the **Value** textbox and click **Write Value Without Response**, as Figure 2 shows.
6. Confirm that the LED is blinking green, indicating that the user application is running.

Switching to the User Application Using the User Button

1. Press and hold the user buttons for 0.5 seconds.
2. Wait until the LED starts blinking green. The user application is now running.

Update the Stack and User Applications

The user and stack applications may be updated to fix bugs or bring new features. To update an application, the bootloader must be running and the CySmart tool must be correctly configured. See Switch to the Bootloader and Configure CySmart.

Updating the User Application

In most cases, only the user application needs updating. The following steps describe how to update the user application.

1. In CySmart, click **Start Scan** to start scanning for the bootloader device. Select the "Bootloader BLE" device when listed.
2. Click **Stop Scan** to stop scanning.
3. Click **Update Firmware**.
4. Select the **Application only update** option and click **Next**.
5. Select the new user application firmware image file (*Bootloader_BLE_Upgradable_Stack_App2.cyacd2*) located in the project folder **Bootloader_BLE_Upgradable_Stack_App2.cydsn** > **CortexM4** > **[compiler name]** > **Debug**. This file is generated when App2 is built.
6. Click **Update**.
7. Wait for the application firmware to be downloaded. While the firmware is downloaded, the white LED blinks twice every two seconds.
8. Confirm that the LED is blinking green, indicating that the updated user application is running.

Updating the Stack and User Applications

A stack update may bring new features for the BLE Component. After a stack update, the user application must be updated. The following steps describe how to update the stack and user applications in a single operation. Note that updating the stack is a critical phase because it contains the bootloader.

1. In CySmart, click **Start Scan** to start scanning for the bootloader device. Select the "Bootloader BLE" device when listed.
2. Click **Stop Scan** to stop scanning.
3. Click **Update Firmware**.
4. Select the **Application and Stack update** option and click **Next**.
5. Select the new stack application firmware image file (*Bootloader_BLE_Upgradable_Stack_App1.cyacd2*) located in the project folder **Bootloader_BLE_Upgradable_Stack_App1.cydsn** > **CortexM4** > **[compiler name]** > **Debug**. This file is generated when App1 is built.

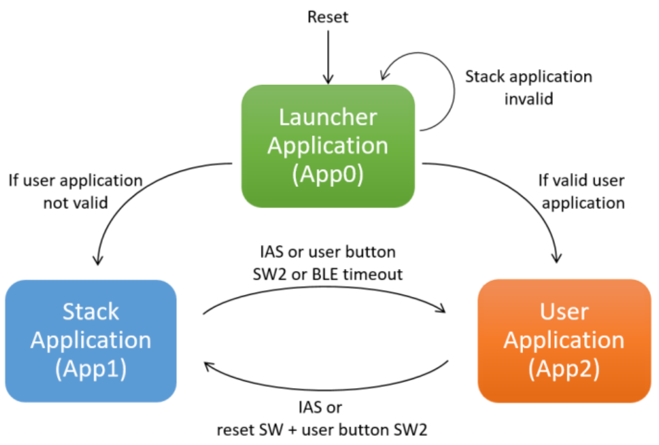
6. Select the new user application firmware image file (*Bootloader_BLE_Upgradable_Stack_App2.cyacd2*) located in the project folder **Bootloader_BLE_Upgradable_Stack_App2.cydsn** > **CortexM4** > **[compiler name]** > **Debug**. This file is generated when App2 is built.
7. Click **Update**.
8. Wait for the stack firmware to be downloaded. While the firmware is downloaded, the white LED flashes twice every two seconds.
9. Confirm that the LED is purple, indicating that App0 is updating the stack. This operation may take a few seconds.
10. Confirm that the LED flashes white twice every two seconds indicating that the bootloader is running again and downloading the user application.
11. Confirm that the LED is blinking green, indicating that the updated user application is running.

Application Switching and LED Status Overview

Figure 3 shows an overview of how applications can transfer control to one another.

The launcher application is located at the start of the user flash. Its purpose is to copy stack updates to the proper location. The stack application contains the bootloader. For more information, see Design and Implementation.

Figure 3. Application Switching Overview


Reset
Stack application invalid
Launcher Application (App0)
If user application not valid
If valid user application
IAS or user button SW2 or BLE timeout
Stack Application (App1)
User Application (App2)
IAS or reset SW + user button SW2

An overview of the LED status for each application is shown in Table 2.

Table 2. LED Status Overview

Application	Color	State	Description
Launcher Application (App0)	Purple	Steady	Launcher is copying the stack update to the persistent location.
	Yellow	Steady	Stack application is invalid. Device must be reprogrammed.
Stack Application (App1)	White	Blinks once every 2 seconds	Bootloader is advertising.
	White	Blinks twice every 2 seconds	An application is being received.
	None	OFF	Bootloader is connected but not receiving an application.
	Red	Steady	Hibernating
User Application (App2)	Green	Blinking	BLE Keyboard is advertising.
	None	OFF	BLE Keyboard is connected.
	Blue	Steady	Caps Lock is ON.
	Red	Steady	Hibernating

Note: If the specified V_{DDD} within the **Design Wide Resources** > **System** is less than 2.7 V, only the red LED is used in the stack and user applications. Caps Lock status is not shown.

Note: To enable the blue LED on App2, connect to the PSoC 6 MCU via Bluetooth without using the BLE dongle. Press the **Caps Lock** key on your keyboard or press the user button on the kit.

Design and Implementation

This example has three applications: “App0”, “App1”, and “App2”. Each application is a separate PSoC Creator project with the following features:

- App0 is the launcher application. It copies a stack update to the proper location and starts either the stack or the user application depending on the availability or if the user button is pressed. This application cannot be updated by OTA bootloading.
- App1 is the stack application. It contains the bootloader and BLE stack. The bootloader can download an update to the user application or to the stack.
- Updating the stack requires placing the update in a temporary location and switching to the launcher for copying because the stack cannot overwrite itself.
- App2 is the user application. The BLE Component contains only profiles without the supporting stack code. Required stack code and variables are shared from the stack application, considerably reducing the size of the user application.
- Modifying the stack application requires recompiling the user application because the location of the stack functions/variables may have changed.
- A RAM region is reserved for stack variables; the user application cannot use this.
- The stack and the user application transfer control between them using a BLE event or by pressing of buttons.
- The provided user application demonstrates several Bluetooth services. It is based on CE215121 BLE HID Keyboard.

Comparison with the Standard BLE Bootloader

The Upgradable Stack Bootloader has the following advantages when compared to the standard BLE Bootloader:

- Lower total flash memory usage, because the BLE stack is shared between applications.
- Faster update when updating only the user application, because of the reduced size.
- Depending on the size of the user application and the available flash memory on the device, the standard BLE Bootloader may not be possible to implement.

It also has the following disadvantages:

- The BLE Component must run on the same CPU core configuration for the stack and user application projects.
- Updating the stack and the user application takes longer than the standard BLE Bootloader update.
- The user application RAM is slightly reduced due to the memory reserved for the stack.
- Every stack update requires the user application to be updated, because the location of the stack functions/variables may have changed.
- Updating the stack is a critical phase as it contains the bootloader. Receiving a non-functional bootloader renders the device unusable.

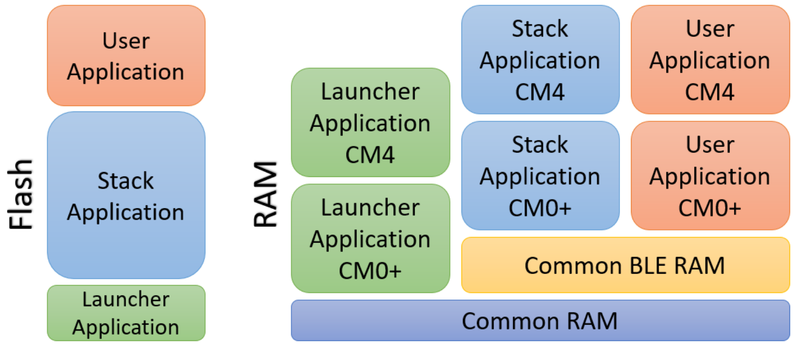
Common Files

The three applications contain common linker configuration files that control where the applications are placed in flash memory and how the RAM is divided between CPUs and applications. The linker files are heavily customized versions of the Bootloader Source Development Kit (SDK) linker files to enable the upgradable stack and code sharing functionality and for ease of switching between the BLE Component CPU core configurations. The BLE Component runs on the CM4 CPU in this example. For more information on how to change the CPU core configuration, see Appendix A: BLE Component CPU Core Configuration.

For more information on customizing PSoC Creator projects for the Bootloader Source Development Kit (SDK), see the PSoC 6 MCU Bootloader SDK Guide.

Figure 4 shows an overview of how the RAM and flash memory is distributed. The actual distribution depends on the compiler used and on the BLE Component CPU core configuration.

Figure 4. RAM and Flash Memory Overview

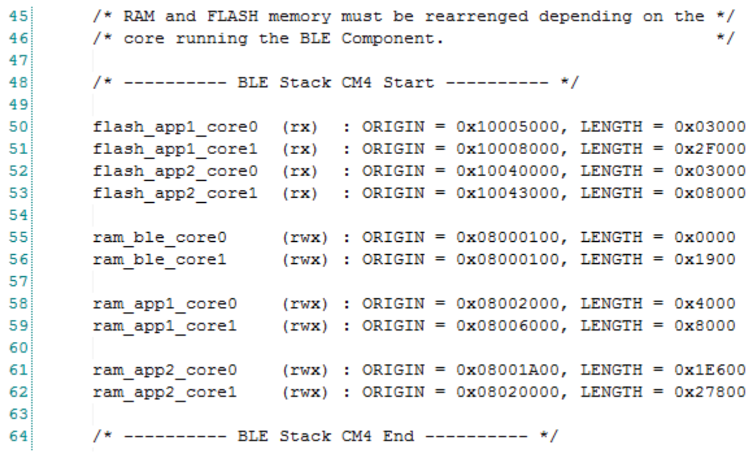


Three sets of configurations are provided in the common linker files for each possible CPU core configuration of the BLE Component (CM0+, CM4, or Dual). These configurations distribute the correct amount of flash memory and RAM to each application.

Figure 5 shows one configuration of the common linker script for the GCC compiler. The configuration can be changed by commenting or uncommenting these sections. MDK and IAR common linker files use a define statement to switch between configurations.

Note: All applications must have the same common linker files. Modifications to a common linker file of an application must be propagated to the other applications by the user.

Figure 5. GCC Common Linker Script Configuration with BLE Stack on CM4



```
    /* RAM and FLASH memory must be rearrenged depending on the */
    /* core running the BLE Component.                         */

    /* ---------- BLE Stack CM4 Start ---------- */

    flash_app1_core0  (rx)  : ORIGIN = 0x10005000, LENGTH = 0x03000
    flash_app1_core1  (rx)  : ORIGIN = 0x10008000, LENGTH = 0x2F000
    flash_app2_core0  (rx)  : ORIGIN = 0x10040000, LENGTH = 0x03000
    flash_app2_core1  (rx)  : ORIGIN = 0x10043000, LENGTH = 0x08000

    ram_ble_core0     (rwx) : ORIGIN = 0x08000100, LENGTH = 0x0000
    ram_ble_core1     (rwx) : ORIGIN = 0x08000100, LENGTH = 0x1900

    ram_app1_core0    (rwx) : ORIGIN = 0x08002000, LENGTH = 0x4000
    ram_app1_core1    (rwx) : ORIGIN = 0x08006000, LENGTH = 0x8000

    ram_app2_core0    (rwx) : ORIGIN = 0x08001A00, LENGTH = 0x1E600
    ram_app2_core1    (rwx) : ORIGIN = 0x08020000, LENGTH = 0x27800

    /* ---------- BLE Stack CM4 End ---------- */
```

Linker files are provided for GCC, MDK, and IAR compilers. Table 3 lists the common linker configuration files.

Table 3. Common Linker Configuration Files

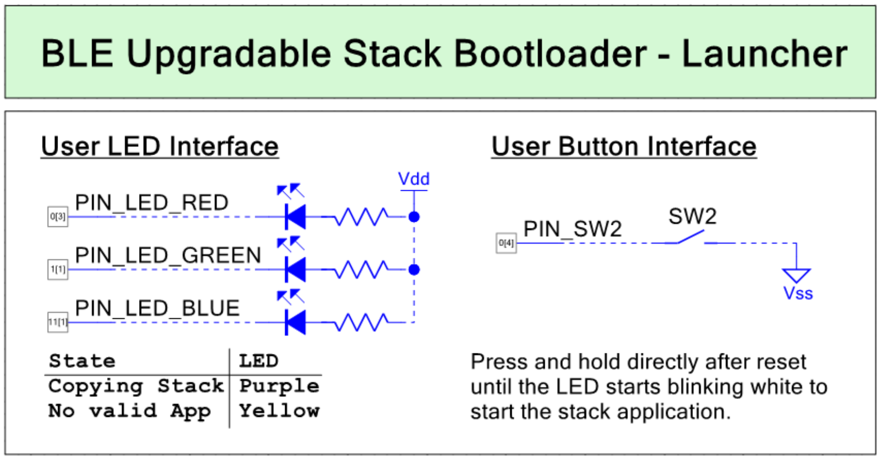
File	Description
bootload_common.ld	GCC linker script. It describes the memory layout and the locations in memory for each CPU in each application. It is included in the *bootload_cm0p.ld* and *bootload_cm4.ld* linker scripts.
bootload_mdk_common.h	MDK linker configuration header file. It describes the memory layout and the locations in memory for each CPU in each application. It is included in the *bootload_cm0p.scat* and *bootload_cm4.scat* scatter files.
bootload_common.icf	IAR linker configuration file. It describes the memory layout and the locations in memory for each CPU in each application. It is included in the *bootload_cm0p.icf* and *bootload_cm4.icf* IAR linker configuration files.

Launcher Application Design Firmware

The launcher application can start the stack or user application. Additionally, it copies a stack update from a temporary location to the persistent location if a flag is set. If the launcher is the only valid application on the PSoC 6 MCU, the device must be reprogrammed.

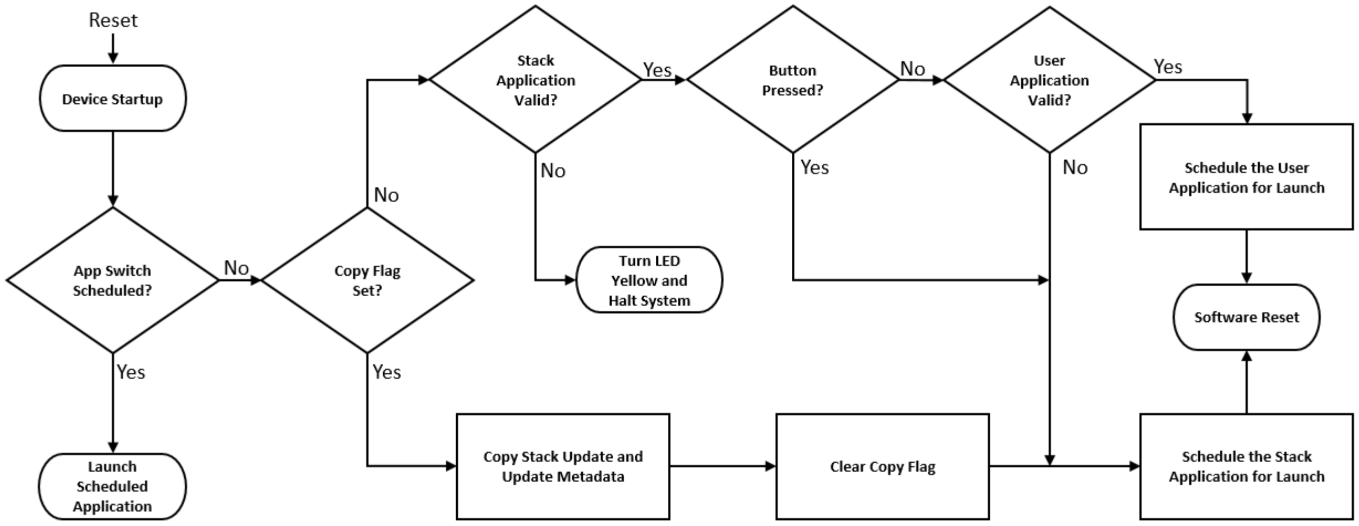
Figure 6 shows the launcher's project schematic.

Figure 6. Launcher Application Project Schematic



The CM0+ CPU executes the launcher application. The CM4 CPU remains in Deep Sleep. Figure 7 shows the firmware flow of the launcher application.

Figure 7. Firmware Flow of the Launcher Application

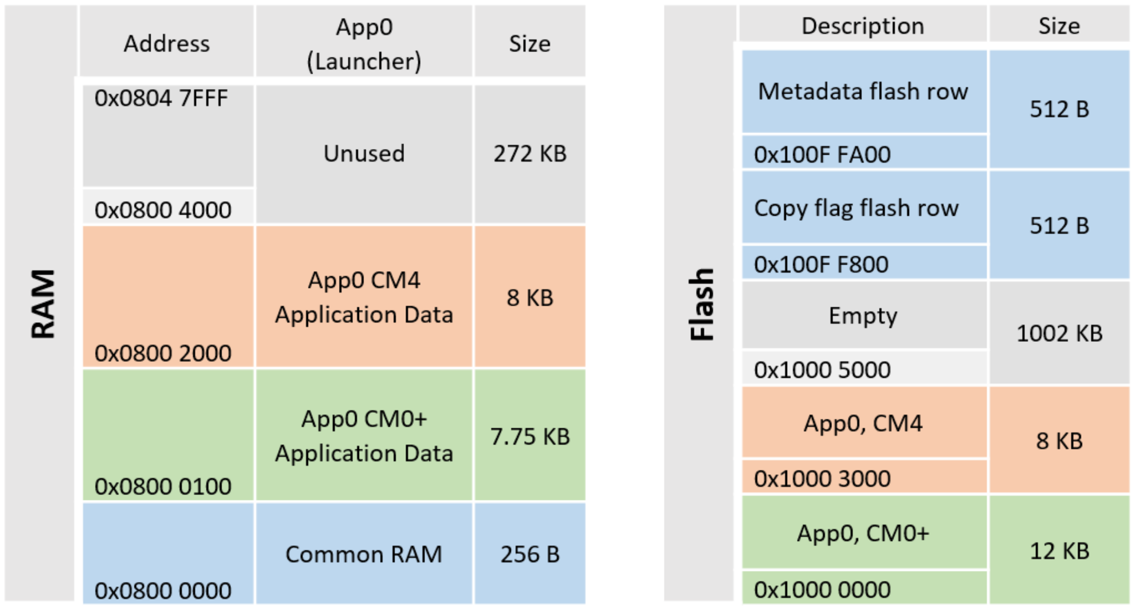

Reset
Device Startup
App Switch Scheduled?
No
Yes
Launch Scheduled Application
Copy Flag Set?
No
Yes
Stack Application Valid?
Yes
No
Turn LED Yellow and Halt System
Button Pressed?
No
Yes
User Application Valid?
Yes
No
Schedule the User Application for Launch
Software Reset
Copy Stack Update and Update Metadata
Clear Copy Flag
Schedule the Stack Application for Launch

If using the MDK compiler, the `CyReturnToBootloaddableAddress` variable is reset at startup. This variable contains an address to return to when reinitializing the stack variables within the user application.

Memory Layout

The memory layout of the launcher application is independent of the compiler used and of the BLE Component configuration. The metadata row is defined in the *bootload_user.c* file of the launcher and is populated with data defined in the common linker configuration files. Figure 8 shows the layout of the RAM and flash memory for the launcher application.

Figure 8. Memory Layout of the Launcher Application



Design Files

Table 4 lists the files used in the launcher application and describes their functionality.

Table 4. Design Firmware Files of the Launcher Application

File	Description
main_cm4.c, main_cm0p.c	Contains the main() function for each CPU. Launcher functionality is implemented in the main function of the CM0+ CPU.
cy_bootload.c / .h	Bootloader software development kit (SDK) files
bootload_user.h	Contains `#define` user-editable statements that control the operation and enable features in the SDK.
bootload_user.c	Defines the metadata initial values. Contains two functions (`Cy_Bootload_ReadData` and `Cy_Bootload_WriteData`) that control access to the internal memory for copying and validating applications.
bootload_cm0p.ld, bootload_cm4.ld	Custom GCC linker scripts. These files place the code and data sections for each of the CPUs as well as the bootloader and other regions.
bootload_cm0p.scat, bootload_cm4.scat	MDK scatter files. These files place the code and data sections for each of the CPUs as well as the bootloader and other regions.
bootload_cm0p.icf, bootload_cm4.icf	IAR linker configuration files. These files place the code and data sections for each of the CPUs as well as the bootloader and other regions.
post_build_core1.bat	Copies the resulting ELF file into the project's root folder for merging with App1 and App2.

Stack Application Design Firmware

The stack application consists of the BLE stack and a BLE bootloader. The BLE stack code is shared with the user application via symbol extraction from the output file of the linker using a post-build command.

The *BLE_Symbols.txt* file contains a list of all the BLE-related symbols to look for and extract. It also includes symbols to enable reinitializing the stack's variables from the user application. An assembly file, which defines the BLE symbols and their addresses, is generated for each CPU and placed in the user application project.

The bootloader can download an update for the user application or for the stack and user applications. Figure 9 shows the flow of the application update process. To customize the bootload operation and enable Bootloader SDK features, update the `#define` statements as needed in the *bootload_user.h* file. For more information on the SDK, see the Bootloader SDK Guide.

When downloading an update for the stack, the bootloader receives the new metadata of the update. This new metadata is the persistent location of the update and is stored in a virtual, non-existent application metadata entry. The stack update is placed in a temporary location in flash, the address of which is also stored in another virtual application metadata entry. The launcher reads these metadata entries and updates the stack accordingly. The goal is to not modify the metadata of the current stack application until the launcher has successfully copied it.

The temporary location for the stack update is set per default to overwrite the user application. To change the temporary location, modify the `temporaryLocation` variable located in the Cy_Bootload_WriteData function of the *bootload_user.c* file with the desired starting address.

Figure 9. Application Update Process

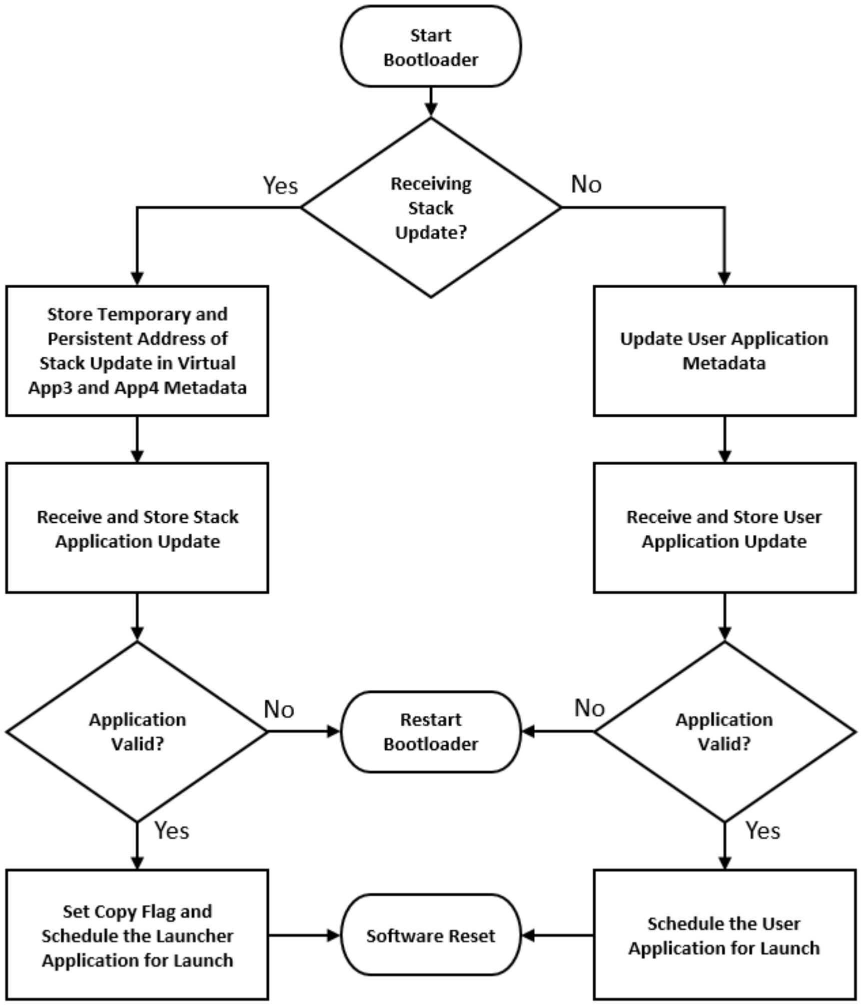
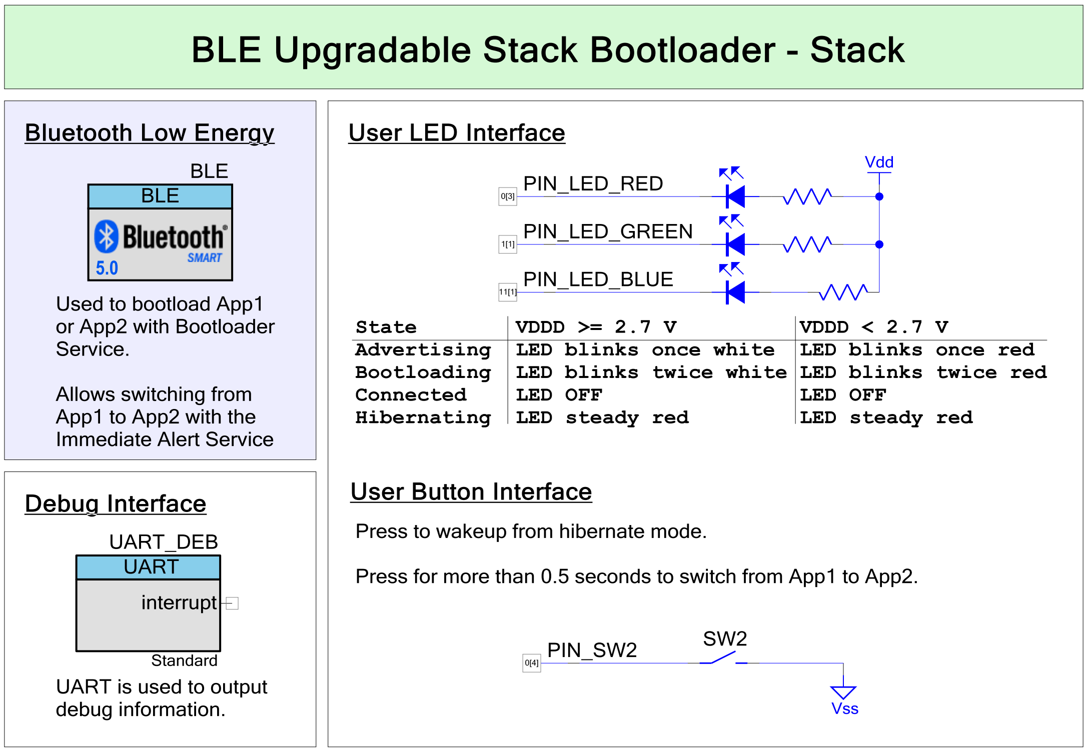

Start Bootloader
Receiving Stack Update?
Yes
No
Store Temporary and Persistent Address of Stack Update in Virtual App3 and App4 Metadata
Update User Application Metadata
Receive and Store Stack Application Update
Receive and Store User Application Update
Application Valid?
No
Restart Bootloader
No
Application Valid?
Yes
Yes
Set Copy Flag and Schedule the Launcher Application for Launch
Software Reset
Schedule the User Application for Launch

Figure 10 shows the stack's project schematic.

Figure 10. Stack Application Project Schematic


BLE Upgradable Stack Bootloader - Stack
Bluetooth Low Energy
BLE
BLE
Bluetooth SMART 5.0
Used to bootload App1 or App2 with Bootloader Service.
Allows switching from App1 to App2 with the Immediate Alert Service
Debug Interface
UART_DEB
UART
interrupt
Standard
UART is used to output debug information.
User LED Interface
PIN_LED_RED
PIN_LED_GREEN
PIN_LED_BLUE
Vdd
State | VDDD >= 2.7 V | VDDD < 2.7 V
Advertising | LED blinks once white | LED blinks once red
Bootloading | LED blinks twice white | LED blinks twice red
Connected | LED OFF | LED OFF
Hibernating | LED steady red | LED steady red
User Button Interface
Press to wakeup from hibernate mode.
Press for more than 0.5 seconds to switch from App1 to App2.
PIN_SW2
SW2
Vss

By default, the linker removes unused sections of the BLE stack. The user application may use sections from the BLE stack that are not required by the stack application and that are removed by the linker. These sections must be explicitly kept to avoid their removal.

RAM is partitioned to reduce the reserved space for the BLE stack. The reserved memory size depends on the compiler used and the configuration of the BLE CPU core. A Common BLE RAM section is defined to store variables required by the stack that must be updated with the application's settings (Clock, IPC, etc.).

Table 5 shows the tasks executed by each CPU depending on the configuration of the BLE Component.

Table 5. CPU Tasks per BLE Configuration

BLE Component CPU Core Configuration	Cortex-M0+	Cortex-M4
Complete Component on CM0+	Bootloads a stack update or a user application update Supervises the user button and IAS value for App Switching	Does nothing
Complete Component on CM4	Does nothing	Bootloads a stack update or a user application update Supervises the user button and IAS value for App Switching
Controller on CM0+. Host and Profiles on CM4	Services the BLE sub-system (BLESS) controller interrupt	

Stack Code Sharing Implementation

GCC is the default compiler used by PSoC Creator. The following section explains the implementation of the code sharing functionality for GCC. For MDK and IAR compilers, see Appendix B: Stack Code Sharing Implementation for MDK and IAR.

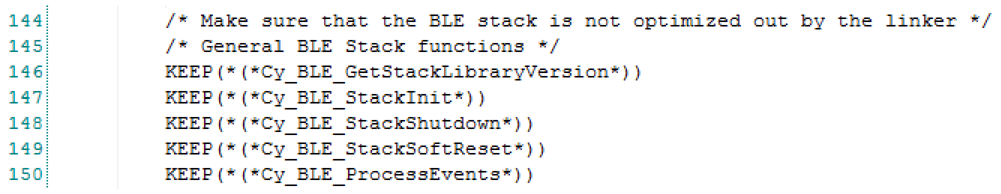
The Bootloader SDK CPU-specific linker scripts (*bootload_cm0p.ld* and *bootload_cm4.ld*) were modified to do the following:

- Explicitly keep the BLE stack functions from being removed
- Place the required RAM data of the BLE stack in a specific section to be reinitialized by the user application
- Expand the copy and zero init tables to initialize the stack related variables
- Provide memory configurations for each BLE Component CPU mode

Unused sections are removed by the linker. Stack functions may be used by the user application and not by the stack application. These functions must be explicitly kept to avoid having them removed by the linker. The CPU-specific linker scripts contain entries to keep the stack functions, as shown partially in Figure 11.

Note: All BLE functions are kept in this example. Functions may be removed to reduce flash usage.

Figure 11. Keeping BLE Stack Sections with GCC

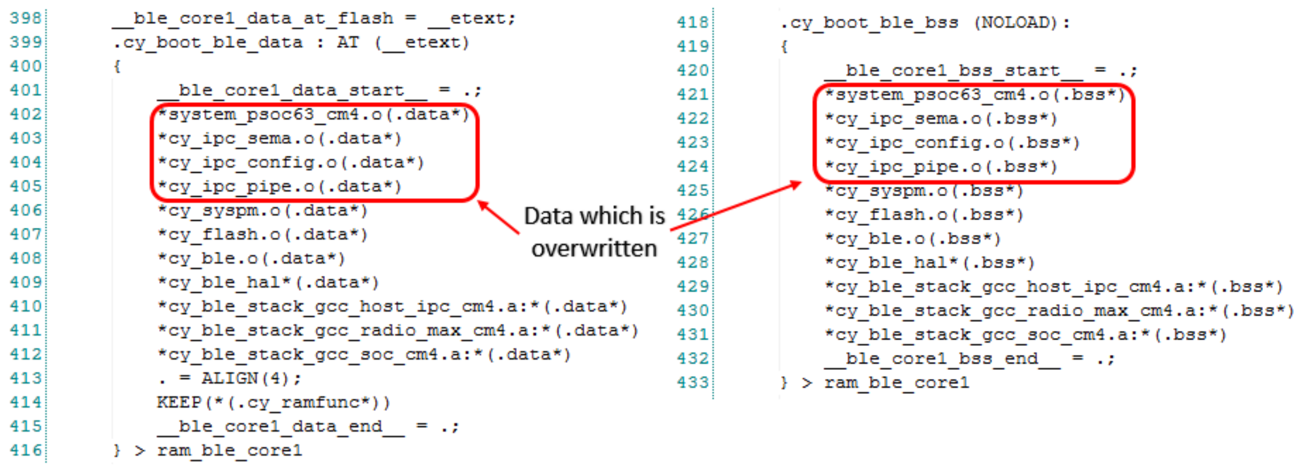


```
    /* Make sure that the BLE stack is not optimized out by the linker */
    /* General BLE Stack functions */
    KEEP(*(*Cy_BLE_GetStackLibraryVersion*))
    KEEP(*(*Cy_BLE_StackInit*))
    KEEP(*(*Cy_BLE_StackShutdown*))
    KEEP(*(*Cy_BLE_StackSoftReset*))
    KEEP(*(*Cy_BLE_ProcessEvents*))
```

The BLE stack requires common variables, such as the clock settings, values of which depend on the running application and that are updated by the firmware. These variables must be overwritten by the user application and therefore placed in a specific section and order in RAM. Other BLE stack-related variables and RAM functions are placed directly after them.

The Common BLE RAM section is defined only if the BLE Component runs on the specific CPU. Figure 12 shows the definition of the Common BLE RAM section for the CM4 CPU. Highlighted entries contain variables that are overwritten when executing the user application.

Figure 12. Common BLE RAM Definition for GCC



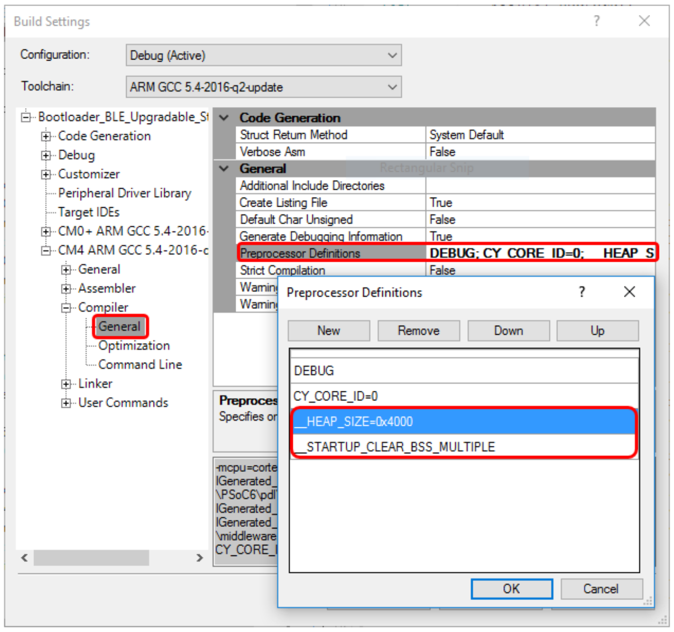
```
__ble_core1_data_at_flash = __etext;
.cy_boot_ble_data : AT (__etext)
{
    __ble_core1_data_start__ = .;
    *system_psoc63_cm4.o(.data*)
    *cy_ipc_sema.o(.data*)
    *cy_ipc_config.o(.data*)
    *cy_ipc_pipe.o(.data*)
    *cy_syspm.o(.data*)
    *cy_flash.o(.data*)
    *cy_ble.o(.data*)
    *cy_ble_hal*(.data*)
    *cy_ble_stack_gcc_host_ipc_cm4.a:*(.data*)
    *cy_ble_stack_gcc_radio_max_cm4.a:*(.data*)
    *cy_ble_stack_gcc_soc_cm4.a:*(.data*)
    . = ALIGN(4);
    KEEP(*(.cy_ramfunc*))
    __ble_core1_data_end__ = .;
} > ram_ble_core1
```

```
.cy_boot_ble_bss (NOLOAD):
{
    __ble_core1_bss_start__ = .;
    *system_psoc63_cm4.o(.bss*)
    *cy_ipc_sema.o(.bss*)
    *cy_ipc_config.o(.bss*)
    *cy_ipc_pipe.o(.bss*)
    *cy_syspm.o(.bss*)
    *cy_flash.o(.bss*)
    *cy_ble.o(.bss*)
    *cy_ble_hal*(.bss*)
    *cy_ble_stack_gcc_host_ipc_cm4.a:*(.bss*)
    *cy_ble_stack_gcc_radio_max_cm4.a:*(.bss*)
    *cy_ble_stack_gcc_soc_cm4.a:*(.bss*)
    __ble_core1_bss_end__ = .;
} > ram_ble_core1
```

The stack application must provide a way for the user application to reinitialize its BLE stack variables. This is realized by expanding the copy and zero initialization tables with symbols to initialize the BLE stack-related data. Afterwards, these symbols are exported to the user application and placed in its copy and zero initialization tables.

By default, the zero initialization table is disabled; this must be enabled. Additionally, The BLE Component requires an increased amount of heap memory when configured with the “Stack and Profile” option. Figure 13 shows the project configuration to increase the heap size of the stack application and enable the zero initialization table.

Figure 13. Heap Size and Zero Table Definitions



Build Settings

Configuration: Debug (Active)

Toolchain: ARM GCC 5.4-2016-q2-update

- Bootloader_BLE_Upgradable_St
  - Code Generation
  - Debug
  - Customizer
  - Peripheral Driver Library
  - Target IDEs
  - CM0+ ARM GCC 5.4-2016-
  - CM4 ARM GCC 5.4-2016-c
    - General
    - Assembler
    - Compiler
      - General
      - Optimization
      - Command Line
    - Linker
    - User Commands

Code Generation	
Struct Return Method	System Default
Verbose Asm	False
General	
Additional Include Directories	
Create Listing File	True
Default Char Unsigned	False
Generate Debugging Information	True
Preprocessor Definitions	DEBUG; CY_CORE_ID=0; __HEAP_S
Strict Compilation	False

Preprocessor Definitions

New | Remove | Down | Up

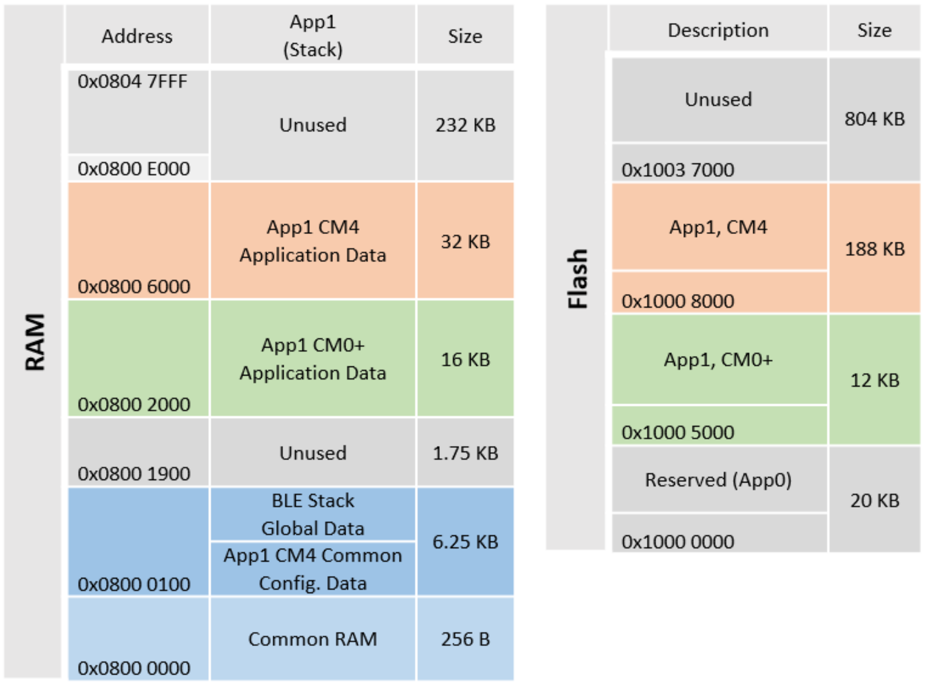
```
DEBUG
CY_CORE_ID=0
__HEAP_SIZE=0x4000
__STARTUP_CLEAR_BSS_MULTIPLE
```

OK | Cancel

Memory Layout

The memory layout of the stack application depends on the compiler used and on the CPU core configuration of the BLE Component. Figure 14 shows the layout of the RAM and flash memory using the GCC compiler and the BLE Component running on the CM4 CPU. The BLE stack reserves 6.25 KB of RAM on this configuration.

Figure 14. Memory Layout of the Stack Application



Design files

Table 6 lists the files used in the stack application and describes their functionality.

Table 6. Design Firmware Files of the Stack Application

File	Description
main_cm4.c, main_cm0p.c	Contains the `main()` function for each CPU and the required functions to reinitialize the stack's variables for MDK and IAR compilers. Calls the bootloader main function or goes into Deep Sleep depending on the BLE configuration.
bootloader.c	Contains the bootloader main and supporting functions
ias.c / .h	Immediate Alert Service (IAS) files. Used to implement IAS for communication between BLE central and peripheral. When the device receives a non-zero value with the IAS, it switches applications.
debug.c / .h	UART `printf` implementation and LED status notification
cy_bootload.c / .h	Bootloader software development kit (SDK) files
bootload_user.h	Contains `#define` user-editable statements that control the operation and enable features in the SDK.
bootload_user.c	Contains user functions required by the SDK: ▪ Five functions that control communications with the bootloader host. These are also called *transport functions.* ▪ Two functions (`Cy_Bootload_ReadData` and `Cy_Bootload_WriteData`) that control access to the internal or external memory. ▪ Supporting functions for `Cy_Bootload_Read/WriteData` to perform checks on data prior writing/reading.
transport_ble.c / .h	Contains bootloader transport functions for the BLE Component. These functions are typically called by the transport functions in *bootload_user.c.*
bootload_cm0p.ld, bootload_cm4.ld	Custom GCC linker scripts. These files place the code and data sections for each of the CPUs as well as the bootloader and other regions.
bootload_cm0p.scat, bootload_cm4.scat	MDK scatter files. These files place the code and data sections for each of the CPUs as well as the bootloader and other regions.
bootload_cm0p.icf, bootload_cm4.icf	IAR linker configuration files. These files place the code and data sections for each of the CPUs as well as the bootloader and other regions.
post_build_core0.bat	Batch file to share code from the CM0+ CPU with the user application
post_build_core1.bat	Batch file to share code from the CM4 CPU with the user application; create the bootloadable file; and merge App0 with App1 into a single hex file.

User Application Design Firmware

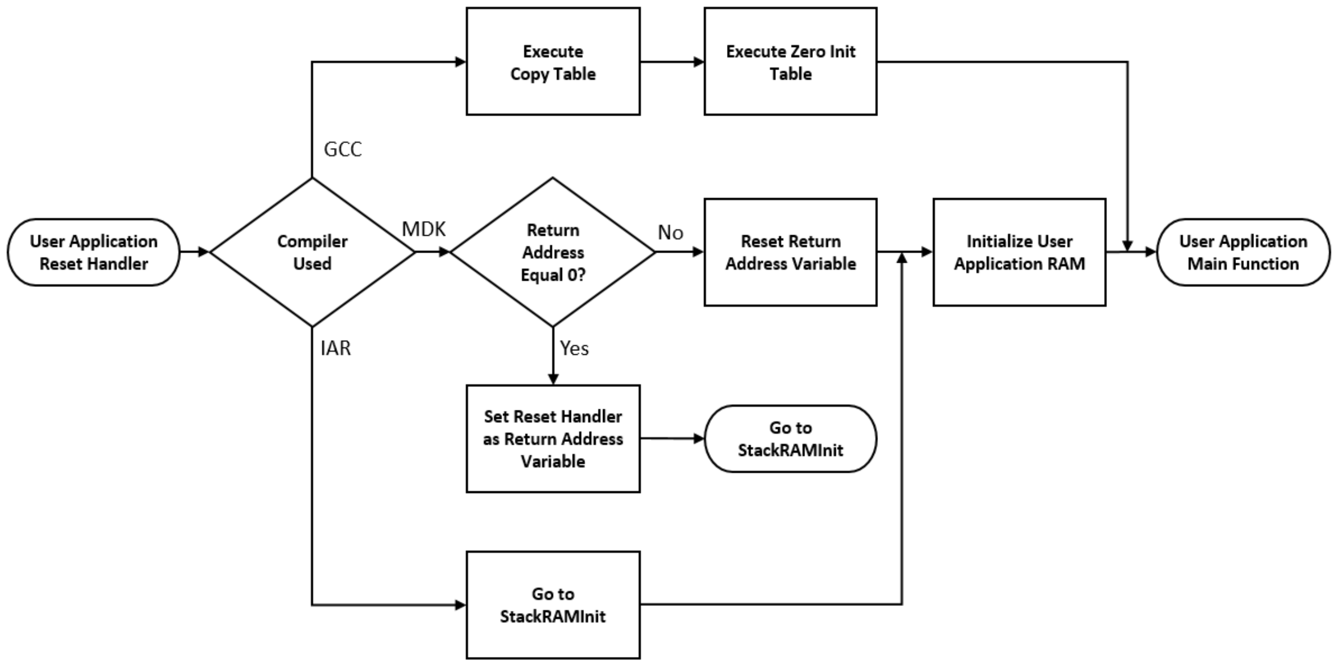
The user application demonstrates a BLE HID Keyboard. The application simulates keyboard presses and battery level. The BLE Component is configured as host and profiles only. The stack functions and variables are imported from the stack application into the *BLE _core0_shared.s* and *BLE_core1_shared.s* files.

Before using the BLE Component, the stack variables must be reinitialized. The method for initialization depends on the compiler used:

- GCC: The startup copy and zero init table were expanded to directly initialize the stack variables at startup. No explicit function is called to reinitialize the stack's variables.
- MDK: The `StackRAMInit` function is called in the reset handler with a return address as a parameter. This function is in the stack application and calls the `__main` initialization routine from the stack application. Afterwards, returns to the reset handler.
- IAR: The `StackRAMInit` function is called in the reset handler. This function is in the stack application and calls the `__iar_data_init3` initialization routine from the stack application.

Figure 15 shows a flowchart of the stack reinitialization routine for all three compilers.

Figure 15. RAM Reinitialization Flow


Execute Copy Table
Execute Zero Init Table
GCC
User Application Reset Handler
Compiler Used
MDK
Return Address Equal 0?
No
Reset Return Address Variable
Initialize User Application RAM
User Application Main Function
IAR
Yes
Set Reset Handler as Return Address Variable
Go to StackRAMInit
Go to StackRAMInit

The BLE interrupt vector must be explicitly initialized by the CPU running the BLE controller before starting the BLE Component. The following function achieves this.

```
Cy_SysInt_SetVector(BLE_bless_isr_cfg.intrSrc, &Cy_BLE_BlessInterrupt);
```

The firmware portion of the design is implemented in the files listed in Table 8. Many of these files require custom settings in both the file and the related PSoC Creator projects.

Table 7 shows the tasks executed by each CPU depending on the configuration of the BLE Component.

Table 7. CPU Tasks per BLE Configuration

BLE Component CPU Core Configuration	Cortex-M0+	Cortex-M4
Complete Component on CM0+	Demonstrates BLE services, as documented in CE215121, BLE HID Keyboard Switches to the stack application if the IAS value is greater than zero	Does nothing
Complete Component on CM4	Does nothing	Demonstrates BLE services, as documented in CE215121, BLE HID Keyboard Switches to the stack application if the IAS values is greater than zero
Controller on CM0+, Host and Profiles on CM4	Services the BLESS controller interrupt.	

Stack Code Sharing Implementation

GCC is the default compiler used by PSoC Creator. The following section explains the implementation of the code sharing functionality for GCC. For MDK and IAR compilers, see Appendix B: Stack Code Sharing Implementation for MDK and IAR.

The Bootloader SDK CPU-specific linker scripts (*bootload_cm0p.ld* and *bootload_cm4.ld*) were modified to do the following:

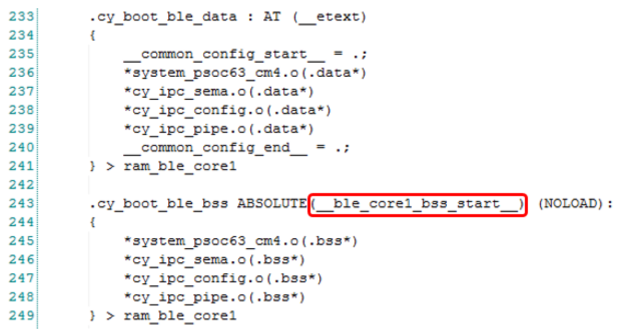
- Place the common RAM data in a specific section to overwrite the stack application's variables
- Expand the copy and zero init tables to reinitialize the stack related variables

- Provide memory configurations for each BLE Component CPU mode

The BLE stack requires common variables, such as clock settings, values of which depend on the running application and that are updated by the firmware. These variables must be overwritten by the user application and therefore placed in a specific section and order in RAM.

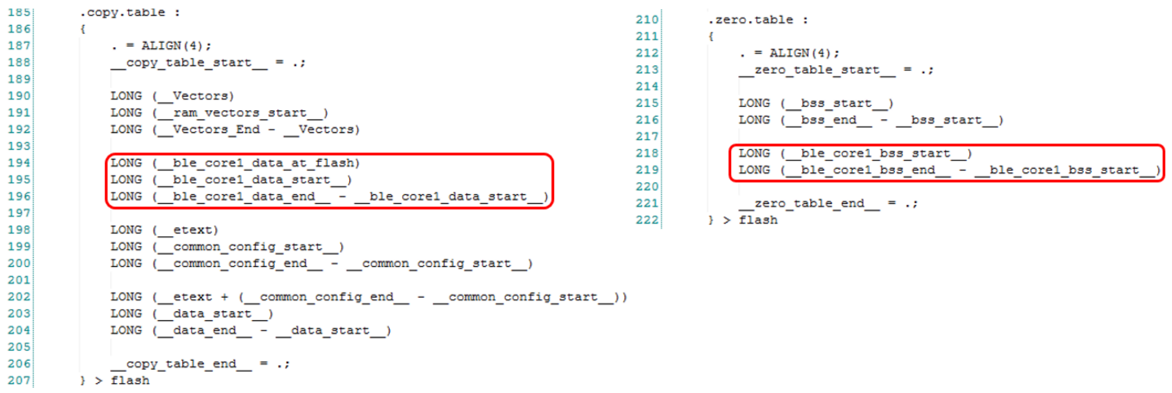
The placing of the common variables in the linker script is shown in Figure 16. The definition of the copy and zero init tables in the linker script is shown in Figure 17. Symbols that were imported from the stack application are highlighted.

Figure 16. Placement of Common Variables



```
    .cy_boot_ble_data : AT (__etext)
    {
        __common_config_start__ = .;
        *system_psoc63_cm4.o(.data*)
        *cy_ipc_sema.o(.data*)
        *cy_ipc_config.o(.data*)
        *cy_ipc_pipe.o(.data*)
        __common_config_end__ = .;
    } > ram_ble_core1

    .cy_boot_ble_bss ABSOLUTE(__ble_core1_bss_start__) (NOLOAD):
    {
        *system_psoc63_cm4.o(.bss*)
        *cy_ipc_sema.o(.bss*)
        *cy_ipc_config.o(.bss*)
        *cy_ipc_pipe.o(.bss*)
    } > ram_ble_core1
```

Figure 17. Definition of Copy and Zero Init Tables

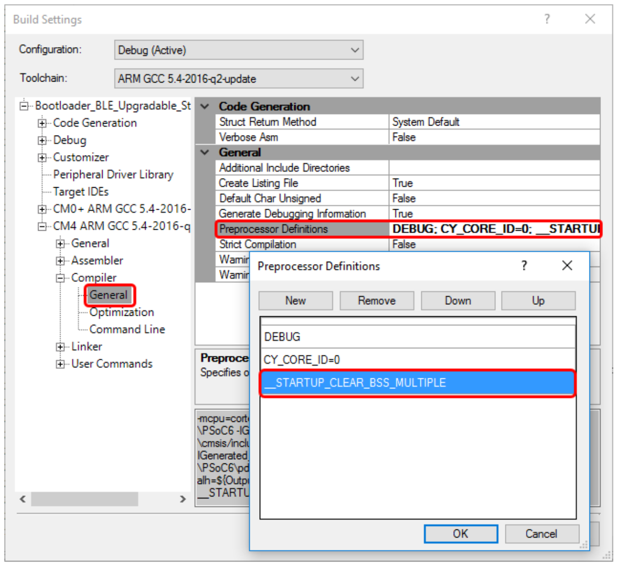


```
    .copy.table :
    {
        . = ALIGN(4);
        __copy_table_start__ = .;

        LONG (__Vectors)
        LONG (__ram_vectors_start__)
        LONG (__Vectors_End - __Vectors)

        LONG (__ble_core1_data_at_flash)
        LONG (__ble_core1_data_start__)
        LONG (__ble_core1_data_end__ - __ble_core1_data_start__)

        LONG (__etext)
        LONG (__common_config_start__)
        LONG (__common_config_end__ - __common_config_start__)

        LONG (__etext + (__common_config_end__ - __common_config_start__))
        LONG (__data_start__)
        LONG (__data_end__ - __data_start__)

        __copy_table_end__ = .;
    } > flash
```

```
    .zero.table :
    {
        . = ALIGN(4);
        __zero_table_start__ = .;

        LONG (__bss_start__)
        LONG (__bss_end__ - __bss_start__)

        LONG (__ble_core1_bss_start__)
        LONG (__ble_core1_bss_end__ - __ble_core1_bss_start__)

        __zero_table_end__ = .;
    } > flash
```

The zero init table is disabled by default and must be enabled. Figure 18 shows the project configuration to enable the zero init table.

Figure 18. Zero Table Definition



Configuration: Debug (Active)

Toolchain: ARM GCC 5.4-2016-q2-update

Code Generation	
Struct Return Method	System Default
Verbose Asm	False
General	
Additional Include Directories	
Create Listing File	True
Default Char Unsigned	False
Generate Debugging Information	True
Preprocessor Definitions	DEBUG; CY_CORE_ID=0; __STARTUI
Strict Compilation	False

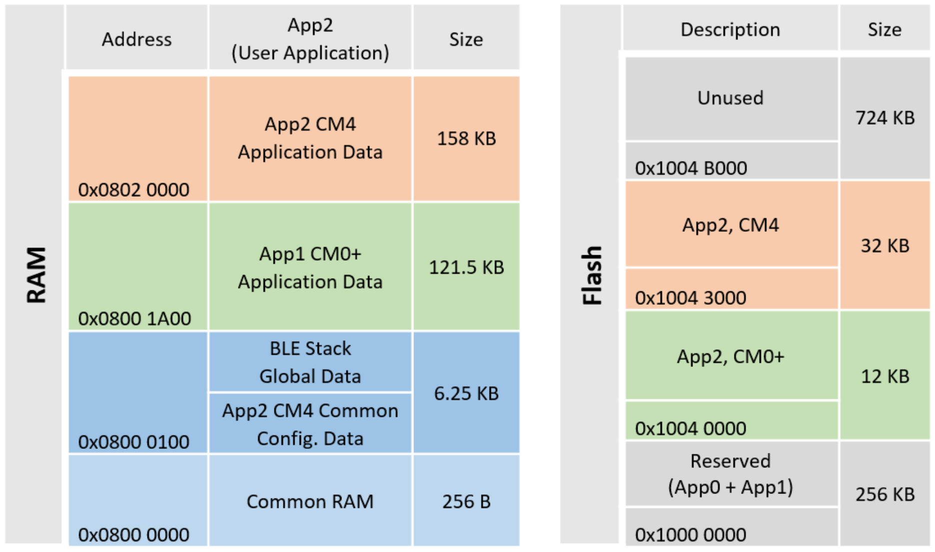
Preprocessor Definitions

```
DEBUG
CY_CORE_ID=0
__STARTUP_CLEAR_BSS_MULTIPLE
```

Memory Layout

The memory layout of the stack application depends on the compiler used and on the CPU core configuration of the BLE Component. Figure 19 shows the layout of the RAM and flash memory using the GCC compiler and the BLE Component running on the CM4 CPU. The BLE stack reserves 6.25 KB of RAM on this configuration. The rest of the RAM is divided between both CPUs.

Figure 19. Memory Layout for the User Application



RAM

Address	App2 (User Application)	Size
0x0802 0000	App2 CM4 Application Data	158 KB
0x0800 1A00	App1 CM0+ Application Data	121.5 KB
0x0800 0100	BLE Stack Global Data / App2 CM4 Common Config. Data	6.25 KB
0x0800 0000	Common RAM	256 B

Flash

Description	Size
Unused / 0x1004 B000	724 KB
App2, CM4 / 0x1004 3000	32 KB
App2, CM0+ / 0x1004 0000	12 KB
Reserved (App0 + App1) / 0x1000 0000	256 KB

Design files

Table 8 lists the files used in the user application and describes their functionality. For more information on the implementation of the BLE Keyboard, see CE215121 BLE HID Keyboard.

Table 8. Design Firmware Files

File	Description
main_cm4.c, main_cm0p.c	Contains the `main()` function for each CPU and the required external functions to reinitialize the stack's variables for MDK and IAR compilers. Calls the host main function or goes into deep sleep depending on the BLE configuration.
ias.c / .h	Immediate Alert Service (IAS) files. Used to implement IAS for communication between BLE central and peripheral. When the device receives a non-zero value with the IAS, it switches applications.
BLE_core0_shared.s, BLE_core1_shared.s	Assembly files that contain definitions from variables and functions shared from the stack application
bas.c / .h, common.h, hids.c /.h scps.c / .h user_interace.c / .h bond.c, debug.c, host_main.c	BLE Keyboard implementation files
cy_bootload.c / .h	Bootloader software development kit (SDK) files
bootload_user.h	Contains `#define` user-editable statements that control the operation and enable features in the SDK
bootload_cm0p.ld, bootload_cm4.ld	Custom GCC linker scripts. These files place the code and data sections for each of the CPUs as well as the bootloader and other regions.
bootload_cm0p.scat, bootload_cm4.scat	MDK scatter files. These files place the code and data sections for each of the CPUs as well as the bootloader and other regions.
bootload_cm0p.icf, bootload_cm4.icf	IAR linker configuration files. These files place the code and data sections for each of the CPUs as well as the bootloader and other regions.
post_build_core1.bat	Batch file to create the downloadable application and to merge App0, App1, and App2 into a single hex file

Design Considerations

Software Reset

When transferring control from one application to another, the recommended method is through a device software reset. This enables each application to initialize device hardware blocks and signal routing from a known state.

You can freeze the state of I/O pins so that they are maintained through a software reset. Defined portions of SRAM are also maintained through a software reset. For more information, see the PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual.

Components and Settings

This section describes the PSoC Creator Components used by the launcher and stack applications, how they are used in the design, and the non-default settings required so they function as intended. Additionally, BLE Component settings for the user application are shown to enable code sharing.

For information on the hardware resources used by a Component, see the Component datasheet.

Launcher Application

Table 9 lists the PSoC Creator Components used in the launcher application.

Table 9: PSoC Creator Components for the Launcher Application

Component	Instance Name	Purpose	Non-default Settings
Pin	PIN_LED_RED PIN_LED_GREEN PIN_LED_BLUE PIN_SW2	LED Status notification.	HW Connection: Unchecked External Terminal: Checked Initial drive state: High (1) Max Frequency: 1 MHz SW2 Pin Initial Drive Mode: Resistive Pull-Up

Stack Application

Table 10 lists the PSoC Creator Components used in the stack application.

Table 10. PSoC Creator Components for the Stack Application

Component	Instance Name	Purpose	Non-default Settings
Bluetooth Low Energy	BLE	Provides communication between the PSoC 6 MCU device and the Bluetooth host for bootloading and app switching	See Stack Application BLE Component Configuration.
UART	UART_DEB	Outputs Bluetooth related debug information	Interrupt mode external.
Pin	PIN_LED_RED PIN_LED_GREEN PIN_LED_BLUE PIN_SW2	LED Status notification	HW Connection: Unchecked External Terminal: Checked LED Pins Drive Mode: High Impedance Digital SW2 Pin Initial Drive Mode: Resistive Pull-Up

User Application

Table 11 shows the required PSoC 6 BLE Component configuration to enable code-sharing. For more information on the PSoC Creator Components used in the user application, see CE215121 BLE HID Keyboard.

Table 11. PSoC Creator BLE Component Required Setting.

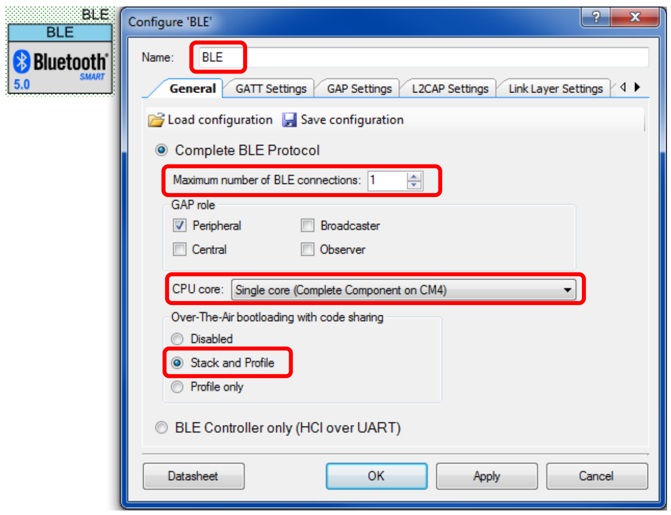
Component	Instance Name	Purpose	Non-default Settings
Bluetooth Low Energy	BLE	Provides BLE communication support	Over-The-Air bootloading with code sharing: Profile Only

Stack Application BLE Component Configuration

General tab (see Figure 20)**:**

- Maximum number of BLE Connections: 1
- CPU core: Single core (Complete Component on CM4)
- Over-The-Air bootloading with code sharing: Stack and Profile

Figure 20. BLE Component, General Tab Configuration



Name: BLE

Complete BLE Protocol

Maximum number of BLE connections: 1

GAP role: [✓ Peripheral] Broadcaster Central Observer

CPU core: Single core (Complete Component on CM4)

Over-The-Air bootloading with code sharing: Disabled [✓ Stack and Profile] Profile only

BLE Controller only (HCI over UART)

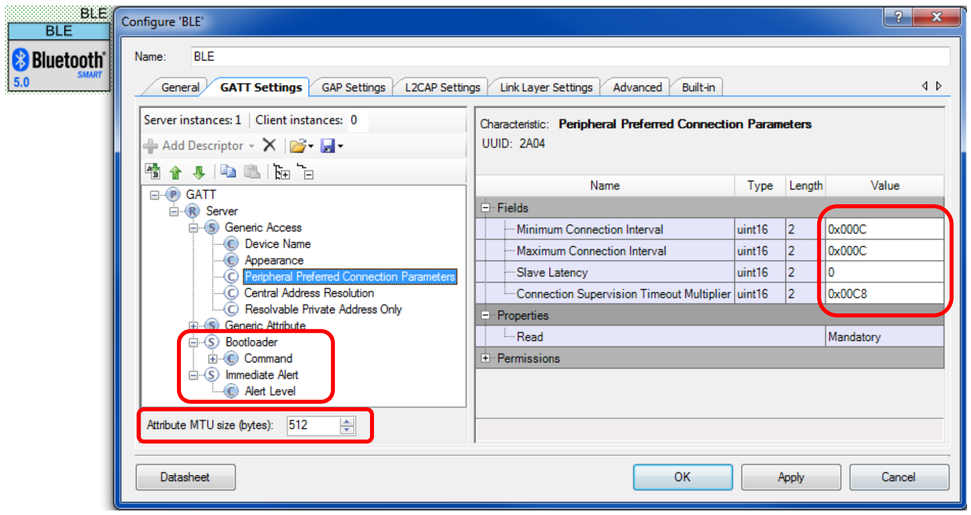
GATT Settings tab (see Figure 21)**:**

- Generic Access, Peripheral Preferred Connection Parameters:
  - Minimum Connection Interval: 0x000C
  - Maximum Connection Interval: 0x000C
  - Connection Supervision Timeout Multiplier: 0x00C8

  The above intervals are selected to minimize bootloading time.

- Bootloader service for BLE bootloading
- Immediate Alert service for app switching
- Attribute MTU size (bytes): 512

Figure 21. BLE Component, GATT Settings Tab Configuration



Name: BLE

Server instances: 1 | Client instances: 0

```
GATT
  Server
    Generic Access
      Device Name
      Appearance
      Peripheral Preferred Connection Parameters
      Central Address Resolution
      Resolvable Private Address Only
    Generic Attribute
    Bootloader
      Command
    Immediate Alert
      Alert Level
```

Attribute MTU size (bytes): 512

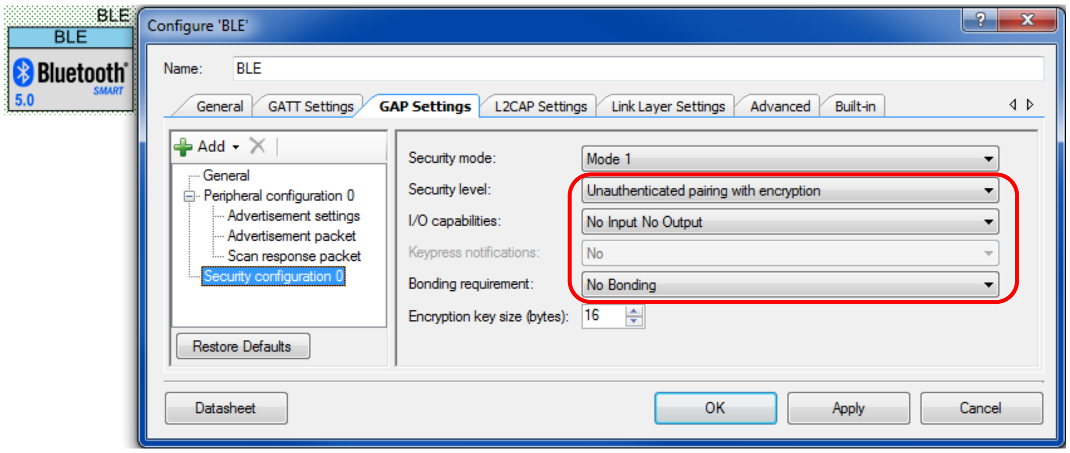
Characteristic: **Peripheral Preferred Connection Parameters**
UUID: 2A04

Name	Type	Length	Value
Fields			
Minimum Connection Interval	uint16	2	0x000C
Maximum Connection Interval	uint16	2	0x000C
Slave Latency	uint16	2	0
Connection Supervision Timeout Multiplier	uint16	2	0x00C8
Properties			
Read			Mandatory
Permissions			

GAP Settings tab:

- Device Name: "BLE Bootloader"
- Peripheral Configuration 0, Advertisement packet: Local Name checked and set to Complete
- Security configuration 0 (see Figure 22)
    - Security level: Unauthenticated pairing with encryption
    - I/O capabilities: No Input No Output
    - Bonding requirement: No Bonding

Figure 22. BLE Component, GAP Settings Tab Configuration



Name: BLE

Security mode: Mode 1
Security level: Unauthenticated pairing with encryption
I/O capabilities: No Input No Output
Keypress notifications: No
Bonding requirement: No Bonding
Encryption key size (bytes): 16

Link Layer Settings tab:

- Link layer max TX and RX payload size (bytes): 251

Related Documents

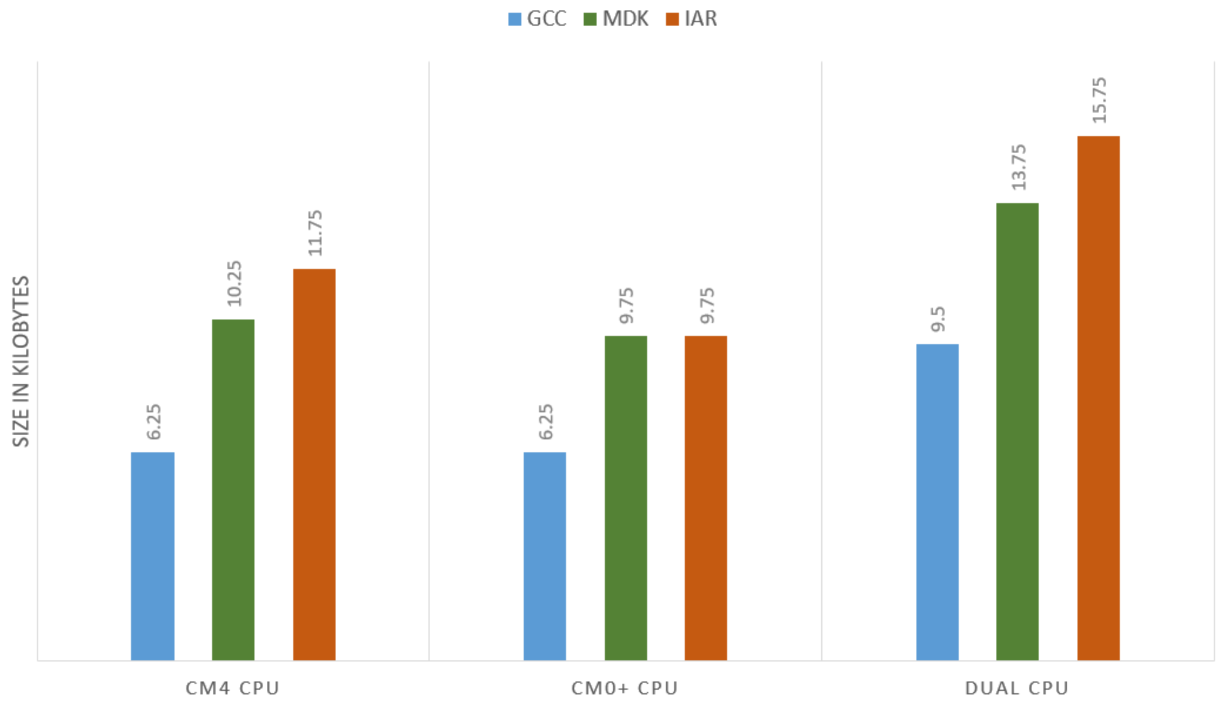
Application Notes	
AN210781 – Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity	Describes PSoC 6 MCU with BLE Connectivity devices and how to build your first PSoC Creator project
AN213924 – PSoC 6 MCU Bootloader Software Development Kit (SDK) Guide	Provides information on how to use the Bootloader SDK, as well as information on bootloading in general
PSoC 6 MCU Bootloader-Related Code Examples	
CE213903 **–** Basic Bootloaders	Describes a UART, I2C and SPI Bootloader for PSoC 6
CE216767 **–** BLE Bootloader	Describes a BLE Bootloader for PSoC 6
CE220959 **–** BLE Bootloader with External Memory	Describes a BLE Bootloader for PSoC 6 that uses SMIF external memory
PSoC Creator Component Datasheets	
BLE	Provides information on Bluetooth Low Energy (BLE) settings and API
UART	Provides information on UART settings and API
Device Documentation	
PSoC 6 MCU: PSoC 63 with BLE Datasheets	PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual
Development Kit Documentation	
CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit	

Appendix A: BLE Component CPU Core Configuration

This example is configured to use the BLE Component on the CM4 CPU. Presets are available to easily switch to the CM0+ or dual-CPU configuration.

Figure 23 shows the comparison of the reserved RAM size for the BLE stack between the GCC, MDK, and IAR compilers depending on the BLE Component core configuration.

Figure 23. Size of Reserved RAM for the BLE Stack



The following steps describe how to change the CPU core configuration to CM0+ or dual-CPU mode.

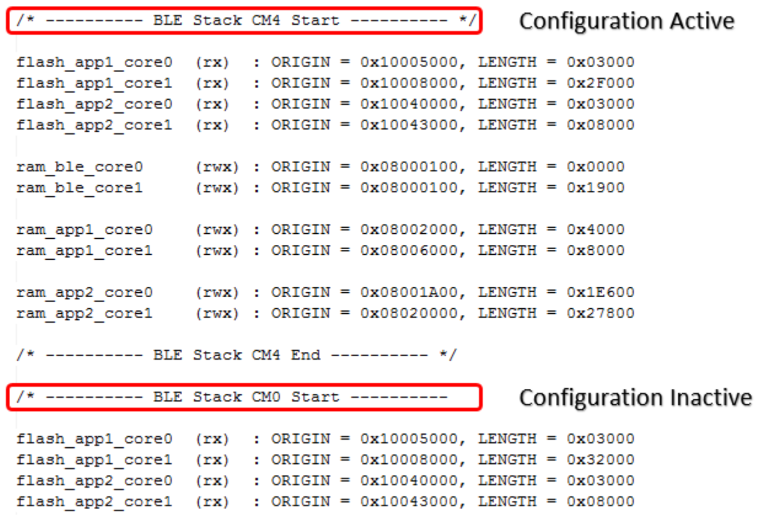
1. Change the BLE Component CPU core configuration to the desired configuration in both the stack and user application schematics.
2. Set the interrupts to the appropriate CPUs in the user and stack applications.
   - 2.1. Set the BLE interrupt to the CM0+ CPU.
   - 2.2. Set all other interrupts of this example to the CM4 CPU if using the dual-CPU mode; else set them to the CM0+ CPU.
3. If the BLE Component is on the CM0+ CPU:
   - 3.1. Move the bootloader folder located in the CM4 folder of the stack application into the CM0+ folder.
   - 3.2. Move the host files folder located in the CM4 folder of the user application into the CM0+ folder.

The following steps depend on the compiler used:

GCC Compiler

4. Configure the linker scripts.
   - 4.1. Open the *bootload_common.ld* linker script of the launcher application project.
   - 4.2. Enable the desired configuration by opening/closing the appropriate comment blocks, as Figure 24 shows.

Figure 24. Linker Script BLE Configuration



```
/* ---------- BLE Stack CM4 Start ---------- */

flash_app1_core0  (rx)  : ORIGIN = 0x10005000, LENGTH = 0x03000
flash_app1_core1  (rx)  : ORIGIN = 0x10008000, LENGTH = 0x2F000
flash_app2_core0  (rx)  : ORIGIN = 0x10040000, LENGTH = 0x03000
flash_app2_core1  (rx)  : ORIGIN = 0x10043000, LENGTH = 0x08000

ram_ble_core0     (rwx) : ORIGIN = 0x08000100, LENGTH = 0x0000
ram_ble_core1     (rwx) : ORIGIN = 0x08000100, LENGTH = 0x1900

ram_app1_core0    (rwx) : ORIGIN = 0x08002000, LENGTH = 0x4000
ram_app1_core1    (rwx) : ORIGIN = 0x08006000, LENGTH = 0x8000

ram_app2_core0    (rwx) : ORIGIN = 0x08001A00, LENGTH = 0x1E600
ram_app2_core1    (rwx) : ORIGIN = 0x08020000, LENGTH = 0x27800

/* ---------- BLE Stack CM4 End ---------- */

/* ---------- BLE Stack CM0 Start ----------

flash_app1_core0  (rx)  : ORIGIN = 0x10005000, LENGTH = 0x03000
flash_app1_core1  (rx)  : ORIGIN = 0x10008000, LENGTH = 0x32000
flash_app2_core0  (rx)  : ORIGIN = 0x10040000, LENGTH = 0x03000
flash_app2_core1  (rx)  : ORIGIN = 0x10043000, LENGTH = 0x08000
```

4.3. Copy the contents of the modified linker script and paste them in the *bootload_common.ld* linker scripts of the stack and user applications.

4.4. Repeat step 4.2 for the *bootload_cm0p.ld bootload_cm4.ld* linker scripts of the stack and user application.

5. Set the heap size and enable the zero init table.

5.1. Open the build settings of the stack application and set the preprocessor directives as shown in Table 12.

Table 12. GCC Preprocessor Directives of the Stack Application

BLE CPU Core Configuration	CM0+ Compiler Preprocessor Directives	CM4 Compiler Preprocessor Directives

BLE on CM0+	DEBUG;
	CY_CORE_ID=0;
	__HEAP_SIZE=0x4000;
	__STARTUP_CLEAR_BSS_MULTIPLE
		DEBUG;
		CY_CORE_ID=0;

BLE on Dual CPU	DEBUG;
	CY_CORE_ID=0;
	__HEAP_SIZE=0x2500;
	__STARTUP_CLEAR_BSS_MULTIPLE
		DEBUG;
		CY_CORE_ID=0;
		__HEAP_SIZE=0x4000;
		__STARTUP_CLEAR_BSS_MULTIPLE

5.2. Click **OK**.

5.3. Open the build settings of the user application and set the preprocessor directives as show in Table 13.

Table 13. GCC Preprocessor Directives of the User Application

BLE CPU Core Configuration	CM0+ Compiler Preprocessor Directives	CM4 Compiler Preprocessor Directives

BLE on CM0+	DEBUG;
	CY_CORE_ID=0;
	__STARTUP_CLEAR_BSS_MULTIPLE
		DEBUG;
		CY_CORE_ID=0;

BLE on Dual CPU	DEBUG;
	CY_CORE_ID=0;
	__STARTUP_CLEAR_BSS_MULTIPLE
		DEBUG;
		CY_CORE_ID=0;
		__STARTUP_CLEAR_BSS_MULTIPLE

5.4. Click **OK**.

6. **Clean and Build** all projects.

MDK Compiler

4. Configure the scatter files.
    - 4.1. Open the *bootload_mdk_common.h* header file of the launcher application project.
    - 4.2. Change the `#define BLE_CM4` line to the desired BLE configuration.
    - 4.3. Copy the contents of the modified header file and paste them in the *bootload_mdk_common.h* header files of the stack and user applications.
5. Set the heap size and disable the MicroLib library.
    - 5.1. Open the build settings of the stack application and set the assembler command line flags as shown in Table 14.

Table 14. MDK Assembler Command Line Custom Flags of the Stack Application

BLE Core Configuration	CM0+ Assembler Command Line Custom Flags	CM4 Assembler Command Line Custom Flags
BLE on CM0+	--pd "__HEAP_SIZE SETA 0x4000"	
BLE on Dual CPU	--pd "__HEAP_SIZE SETA 0x2500"	--pd "__HEAP_SIZE SETA 0x4000"

    - 5.2. Go to **Linker settings** of the CM0+ CPU set the **Use MicroLib** option to false. Do the same for the CM4 CPU if the BLE Component is on dual-CPU mode.
    - 5.3. Click **OK**.
    - 5.4. Open the build settings of the user application and repeat steps 5.2 and 5.3.
6. Update the linker options to keep the stack.
    - 6.1. Open the build settings of the stack application.
    - 6.2. Update the linker command line custom flags of both CPUs to keep the linker from optimizing the stack out. See the *BLE Stack Keep Command List.txt* file located in the stack application project for the required commands for each BLE CPU core configuration.
    - 6.3. Click **OK**.
7. **Clean and Build** all projects.

IAR Compiler

4. Modify the linker configuration files.
    - 4.1. Open the *bootload_common.icf* linker configuration file of the launcher application project.
    - 4.2. Change the `define symbol BLE_CM4 = 1;` line to the desired BLE configuration.
    - 4.3. Copy the contents of the modified linker configuration file and paste them in the *bootload_common.icf* linker configuration files of the stack and user applications.
5. Set the heap size.
    - 5.1. Right-click on the CM0+ stack application project and select **Options**.
    - 5.2. Go to the **Linker settings** and to the **Config** tab.
    - 5.3. Set the configuration file symbol definitions as shown in Table 15.

Table 15. IAR Configuration File Symbol Definitions of the Stack Application

BLE CPU Core Configuration	CM0+ Configuration File Symbol Definitions	CM4 Configuration File Symbol Definitions
BLE on CM0+	__HEAP_SIZE=0x4000	
BLE on Dual CPU	__HEAP_SIZE=0x2500	__HEAP_SIZE=0x4000

   5.4. Click **OK**.
   5.5. Right-click on the CM4 stack application project and select **Options**.
   5.6. Repeat steps 5.2 to 5.4.
6. Update the linker options to keep the stack.
   6.1. Right-click on the CM0+ stack application project and select **Options**.
   6.2. Go to the **Linker settings** and to the **Input** tab.
   6.3. Update the symbol list to keep the stack from being optimized out. See the *BLE Stack Keep Command List.txt* file located in the stack application project for the required commands for each BLE CPU core configuration.
   6.4. Click **OK**.
   6.5. Right-click on the CM4 stack application project and select **Options**.
   6.6. Repeat steps 6.2 to 6.3 to configure the linker settings for the CM4 CPU.
7. **Clean and Build** all projects.

Appendix B: Stack Code Sharing Implementation for MDK and IAR

Applications generated by IAR and MDK compilers use special routines to initialize the RAM. These routines limit the amount of reserved RAM which can be saved in comparison with GCC, and require the user application to call a function from the stack application to reinitialize the RAM, and afterwards return to the user application.

Figure 23 shows the comparison of the reserved RAM size for the BLE stack between the GCC, MDK, and IAR compilers depending on the BLE Component CPU core configuration.

MDK Compiler Implementation

The Bootloader SDK CPU-specific scatter files (*bootload_cm0p.scat* and *bootload_cm4.scat*) were modified to do the following:

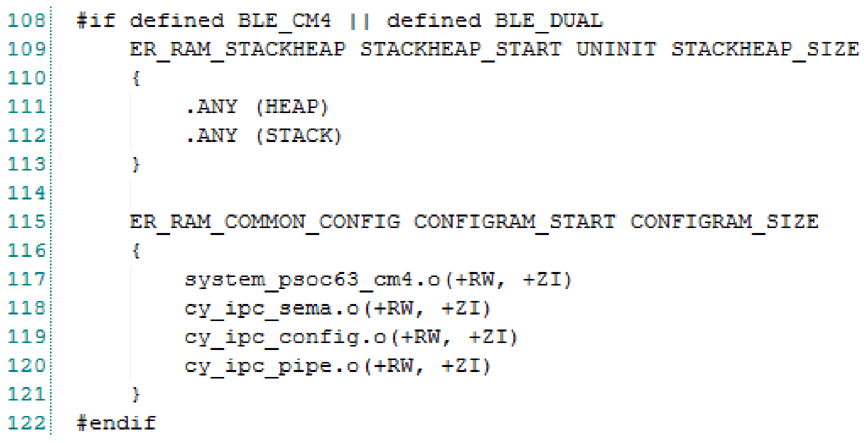
- Separate the stack and heap from the RAM data to reduce the reserved RAM size for the BLE stack
- Place the common configuration variables in a specific section to be overwritten by the user application

Unused sections are removed by the linker. Stack functions may be used by the user application and not by the stack application. These functions must be explicitly kept to not be removed. Linker command line custom flags are used to keep the stack from being removed. See the *BLE Stack Keep Command List.txt* file located in the stack application project for the required commands for each BLE CPU core configuration.

The BLE stack requires common variables, such as clock settings, values of which depend on the running application and that are updated by the firmware. These variables must be overwritten by the user application and therefore placed in a specific section and order in RAM.

The separation of the stack and heap from the RAM data and the definition of the Common BLE RAM section is made only if the BLE Component runs on the specific CPU. Figure 25 shows the definition of the Common BLE RAM section and the separation of the stack and heap from RAM data for CM4.

Figure 25. Common BLE RAM Definition for MDK



```
#if defined BLE_CM4 || defined BLE_DUAL
    ER_RAM_STACKHEAP STACKHEAP_START UNINIT STACKHEAP_SIZE
    {
        .ANY (HEAP)
        .ANY (STACK)
    }

    ER_RAM_COMMON_CONFIG CONFIGRAM_START CONFIGRAM_SIZE
    {
        system_psoc63_cm4.o(+RW, +ZI)
        cy_ipc_sema.o(+RW, +ZI)
        cy_ipc_config.o(+RW, +ZI)
        cy_ipc_pipe.o(+RW, +ZI)
    }
#endif
```

The stack application must provide a way for the user application to reinitialize its BLE stack variables. This is realized by providing the `StackRAMInit` function, which requires a return address as a parameter. The MicroLib library must be disabled for the RAM reinitialization to work.

The return address is stored in the `CyReturnToBootloadableAddress` global variable. This variable is in the common RAM and is set to 0 after every reset.

`StackRAMInit` calls the scatterload function to initialize the RAM of the stack application. Afterwards, scatterload calls the `_platform_pre_stackheap_init` weak function, which has been redefined to return to the user application using `CyReturnToBootloadableAddress`.

IAR Compiler Implementation

The Bootloader SDK CPU-specific linker configuration files (*bootload_cm0p.icf* and *bootload_cm4.icf*) were modified to do the following:

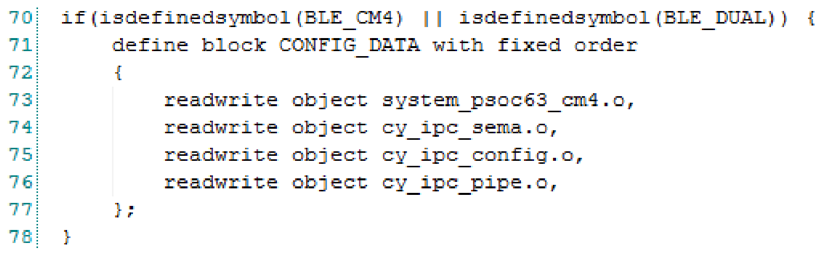
- Separate the stack and heap from the RAM data to reduce the reserved RAM size for the BLE stack
- Place the common configuration variables in a specific section to be overwritten by the user application

Unused sections are removed by the linker. Stack functions may be used by the user application and not by the stack application. These functions must be explicitly kept to not be removed. The Keep symbols list in the linker input settings is used to keep the stack from being removed. See the *BLE Stack Keep Command List.txt* file located in the stack application project for the required commands for each BLE CPU core configuration.

The BLE stack requires common variables, such as clock settings, values of which depend on the running application and that are updated by the firmware. These variables must be overwritten by the user application and therefore placed in a specific section and order in RAM.

The separation of the stack and heap from the RAM data and the definition of the Common BLE RAM section is made only if the BLE Component runs on the specific CPU. Figure 26 shows the definition of the Common BLE RAM section and the separation of the stack and heap from RAM data for the CM4.

Figure 26. Common BLE RAM Definition for IAR



```
if(isdefinedsymbol(BLE_CM4) || isdefinedsymbol(BLE_DUAL)) {
    define block CONFIG_DATA with fixed order
    {
        readwrite object system_psoc63_cm4.o,
        readwrite object cy_ipc_sema.o,
        readwrite object cy_ipc_config.o,
        readwrite object cy_ipc_pipe.o,
    };
}
```

The stack application must provide a way for the user application to reinitialize its BLE stack variables. This is realized by providing the `StackRAMInit` function. In comparison with the MDK implementation, this variable requires no parameters.

`StackRAMInit` calls the `__iar_data_init3` function to initialize the RAM of the stack application. Afterwards, the function returns to the user application.

Document History

Document Title: CE220960 – PSoC 6 MCU BLE Upgradable Stack Bootloader

Document Number: 002-20960

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6097169	CFMM	03/13/2018	New code example

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 MCU

Cypress Developer Community

Community | 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