ModusToolbox™ software

creating custom BSP

About this document

Scope and purpose

A board support package is a set of files that allow ModusToolbox™ software to understand how to interact with hardware. A BSP specifies several critical items, including:

- MCU and wireless connectivity devices used
- hardware configuration files for the device(s)
- startup code and linker files for the device(s)
- information on how to build applications for the board using make
- other libraries that are required to support a board

ModusToolbox™ software supports a wide variety of evaluation boards including those with PSoC™ 6 MCU paired with AIROC™ CYW43xxx Wi-Fi & Bluetooth® combo devices provided by Infineon Technologies and its module partners. When developing a prototype or production product, you may require a custom BSP to allow changes to any or all of the above items.

This document helps you create a custom BSP to fully utilize the solutions offered through ModusToolbox™ software. It explains the roles performed by the BSP in brief, shows how to create a custom BSP and modify the connectivity devices, and discusses how the firmware binaries are delivered in Infineon Technologies software resources. The document uses PSoC™ 6 and AIROC™ CYW43xxx Wi-Fi & Bluetooth® as examples but the same concepts apply to other devices such as Infineon XMC™, PMG1, and AIROC™ Bluetooth discrete devices.

Document conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, menus and sub-menus
Italics	Denotes file names and paths.
`Courier New`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets
File > New	Indicates that a cascading sub-menu opens when you select a menu item

Abbreviations and definitions

The following define the abbreviations and terms used in this document:

- BSP – board support package
- HAL – hardware abstraction layer

Table of contents

1 BSP contents

After creating an application from a ModusToolbox™ code example, the BSP is stored in the application's *libs* directory by default, using the nomenclature *TARGET_<BSP_NAME>*. If you open the directory, you will see a number of files and subdirectories that accomplish different functions.

1.1 Configuration files

The BSP contains code for initializing the device and loading the default configuration. This initialization code can be found in the source files *cybsp.c* and *cybsp.h*. The default configuration is specified in a design file located in the *COMPONENT_BSP_DESIGN_MODUS* directory. The directory contains the following design files:

- *design.modus*: This contains details of clock configuration, pin aliases, power modes, and peripheral configuration.
- *design.cycapsense*: This contains details of the CAPSENSE™ widgets which correspond to the button and slider interfaces on the board. This file is specific to devices with CAPSENSE™ support such as PSoC™ 6.
- *design.cyqspi*: This contains details of the QSPI memory part loaded on the evaluation board. This file will not appear in BSPs that do not have an external QSPI memory on the board.

The build infrastructure uses the Device Configurator tool provided in the ModusToolbox™ installation on these design files to generate C files that represent the specified device configuration. The generated C files are referenced in *cybsp.c*.

1.2 Startup and Linker scripts

The BSP provides startup and linker scripts. PSoC™ 6 MCUs contain two CPUs: Arm® Cortex® CM0+ CPU and Arm® Cortex® CM4 CPU. The startup and linker scripts for each CPU can be found in the *COMPONENT_CM0P* and *COMPONENT_CM4* directories under the corresponding toolchain directories.

1.3 BSP makefile

The BSP includes a makefile with the name *<BSP>.mk* that sets make variables specific to the BSPs. For example:

- `COMPONENTS`: The software libraries provided in ModusToolbox™ software use compartmentalization by using the COMPONENT keyword to selectively include files during the build process. The build infrastructure uses the value of the variable COMPONENTS to make the selection. All the directories whose names match with the variable name prefixed with COMPONENT are included in the build process. Please note that the names are case-sensitive. Some of the COMPONENT variables are needed by all applications using a given BSP. The BSP Makefile lists these components.
- `DEVICE`: This refers to the PSoC™ 6 MCU device part number.
- `ADDITIONAL_DEVICES`: This refers to the AIROC™ Wi-Fi & Bluetooth® combo device part number.
- `CORE`: The CPU core for which the application is built. This defaults to the CM4 CPU unless specified in the application *Makefile*.

1.4 Dependencies

The BSP alone is not sufficient to develop applications, there are a few additional libraries that provide core functions for the build process as well as APIs to use the peripherals on the device and board. These libraries are pulled in by default by the Project Creator when creating the application.

The Project Creator tool refers to the manifest files located on GitHub to know these dependencies and downloads them to the workspace. If you need offline access to the manifest files, refer to KBA230953.

When creating a custom BSP, the new BSP will not be available in the manifest file, so dependencies must be specified in a different way. For example, a custom BSP will contain one or more *.mtbx* files its *deps* subdirectory. These files specify the URL for the Git repo of the library, the version to use, and the location where the library should be stored on disk (either local to the application or in the shared asset location). The following is an example *.mtbx* file:

```
https://github.com/Infineon/core-lib#latest-v1.X#$$ASSET_REPO$$/core-lib/latest-v1.X
```

During the `make getlibs` process (run automatically during project creation), the libraries specified in the *.mtbx* files are added to the project as indirect dependencies. The example *.mtbx* file above includes the core-lib library into the application with the major version of 1 and the latest minor version, and it places the library in the default asset repo (for example, *mtb_shared*).

2 Custom BSP process

2.1 Create application

In order to create a custom BSP, we will be deriving it from an existing BSP using the Eclipse IDE for ModusToolbox™ as shown in the following steps. Note that this can also be done by using the stand-alone Project Creator tool if you want to use a different IDE or the command line.

1. Launch the Eclipse IDE for ModusToolbox™.
2. Click the **New Application** link in the Quick Panel (or, use **File > New > ModusToolbox Application**). This launches the Project Creator tool.
3. Select a BSP from the "Choose Board Support Package (BSP)" page. These instructions use the CY8CKIT-062-WIFI-BT kit. You can either choose an existing BSP that is similar to the custom BSP being created, or you can select an empty starting BSP such as PSOC6-GENERIC. If possible, it is recommended to select a BSP with the same device(s) as your custom BSP.
4. Select the "Empty PSoC 6 App" example from the –"Select Application" page by enabling the check box. Optionally, change the suggested **New Application Name**.
5. Click **Create** to complete the application creation process.

2.2 Generate custom TARGET

On Windows, use the command line "modus-shell" program provided in the ModusToolbox™ installation instead of a standard Windows command-line application. This shell provides access to all ModusToolbox™ tools. You can access it by typing modus-shell in the search box in the Windows menu. In Linux and macOS, you can use any command terminal application.

Navigate to the directory where the "Empty PSoC 6 App" was created and issue the following command:

```
make bsp TARGET_GEN=CUSTOM_BSP_NAME
```

You can use a different name instead of `CUSTOM_BSP_NAME` if you desire.

If you want to use a different PSoC™ 6 MCU, you can specify it using the variable `DEVICE_GEN`. For example:

```
make bsp TARGET_GEN=CUSTOM_BSP_NAME DEVICE_GEN=CY8C624ABZI-S2D44
```

The above command may result in errors if the device specified in `DEVICE_GEN` is from a different family than the one with which the application was created with. To resolve the errors, you will need to open the Device Configurator tool, apply the suggested fixes in the error panel, and save the file.

The BSP is created in the application directory and has the same content as the BSP that was selected in Project Creator.

Since we want the application to be built for the custom target we generated, update the `TARGET` variable in the makefile with the name of the custom BSP, which is `CUSTOM_BSP_NAME` in this case. Doing so ensures that the C files specifying the device configuration are re-generated to match the latest design files. For example:

```
TARGET=CUSTOM_BSP_NAME
```

Then, fetch any dependencies by issuing the following command:

```
make getlibs
```

3 Customize created BSP

Since the custom BSP files are generated starting from the standard BSP provided by Infineon Technologies, you will likely want to make changes to the newly created BSP to suit your needs.

3.1 Startup and Linker scripts

You may want to make changes to the startup or linker scripts. For example, you may want to modify linker scripts so that you can turn off a few SRAM blocks to achieve lower power in your application, or you may want to use the CM0+ CPU as a bootloader resulting in a change in the flash and SRAM usage for both CPUs. You can refer to the examples on Low-Power CAPSENSE™ and MCUboot-based Basic Bootloader to see examples on how to create custom linker scripts for your application.

3.2 Customizing device configuration

There are cases where you might need to make use of the Device Configurator tool for using more advanced hardware features. For instance, the Low-Power Assistant middleware in AnyCloud uses the Device Configurator for configuring Packet filters, ARP offload, and TCP Keep-alive offload features to help achieve lower power consumption on the host MCU. These offload features can be configured using the **Connectivity Device** tab in the Device Configurator.

Refer to the AnyCloud WLAN Low-Power and AnyCloud WLAN Offload TCP-Keepalive examples to learn more on the configurations made. These examples also turn off High-Frequency clocks (CLK_HF) to unused peripherals, and disable the PLL to lower current consumption. To learn more about the Device Configurator and how to make changes using it refer to the Device Configurator guide.

3.3 Adding or removing dependencies

If you need add a new dependent library for your BSP, you can create a *.mtbx* file for the library in the custom BSP's deps directory. Likewise, if you no longer need a dependent library, you can remove its *.mtbx* file.

If you are adding or removing dependencies from an existing application, run `make getlibs` to update the dependencies.

4 Change connectivity device

Another change you might want to make in a custom BSP is to change the AIROC™ CYW43xxx device. As an example, the following files need to be modified to change the connectivity device from CYW4343WKUBG to CYW43012C0WKWBG:

4.1 Modifying the CUSTOM_BSP_NAME.mk

Replace the device name string assigned to the `ADDITIONAL_DEVICES` *Makefile* variable in the *CUSTOM_BSP_NAME.mk* file to `CYW43012C0WKWBG`.

The AIROC™ CYW43xxx device has platform-specific firmware files for accomplishing Wi-Fi and Bluetooth® functionality in the Wi-Fi Host Driver and Bluetooth® libraries. These files are added to the build process through the `COMPONENTS` *Makefile* variable.

The libraries differentiate between the different modules provided by Infineon Technologies and its partners through the device part family name, and the manufacturer of the module. When the same firmware works for modules belonging to the same family of AIROC™ CYW43xxx devices, only the component with device family name is used.

When changing the connectivity device in the BSP, it is important to add the correct `COMPONENTS` values in the *Makefile*. The following table lists the correct values of `COMPONENTS` for some of the connectivity modules supported in ModusToolbox™ software.

Module	**Device part number**	**COMPONENTS**
Murata LBEE5KL1DX	CYW4343WKUBG	4343W MURATA-1DX HCI-UART
Murata LBEE59B1LV	CYW43012C0WKWBG	43012 MURATA-1LV HCI-UART
Azurewave AW-CU427	CYW43438KUBG	43438 AW-CU427-P HCI-UART
USI WM-BAC-CYW-50	CYW43012TC0EKUBG	43012 WM-BAC-CYW-50 HCI-UART
Infineon CYSBSYS-RP01	CYW43012TC0KFFBH	43012 SCL CYSBSYS-RP01 HCI-UART
Sterling-LWB5+ M.2	CYW4373EUBGT	4373 STERLING-LWB5plus HCI-UART

When changing the module, please ensure the existing `COMPONENTS` field is updated with the `COMPONENTS` values suitable for the new connectivity part and that other corresponding values are removed.

Please note that in the case of replacing the connectivity device with Sterling-LWB5+ M.2 module, the following lines need to be added in the *Makefile* because the default firmware files corresponding to AIROC™ CYW4373 family of devices will not work correctly with Sterling-LWB5+ M.2 module.

```
# Ignore the default CYW4373 clm blob. This BSP uses the CLM blob for Laird Connectivity's Sterling-
LWB5+ M.2 radio module
CY_IGNORE+=$(SEARCH_wifi-host-driver)/WiFi_Host_Driver/resources/clm/COMPONENT_4373/4373A0_clm_blob.c \
$(SEARCH_wifi-host-driver)/WiFi_Host_Driver/resources/clm/COMPONENT_4373/4373A0-mfgtest_clm_blob.c \
$(SEARCH_wifi-host-driver)/WiFi_Host_Driver/resources/clm/COMPONENT_4373/clm_resources.h
```

4.2 Modifying design.modus

Open *design.modus* in a text editor. This file is located in the BSP under *COMPONENT_BSP_DESIGN_MODUS*.

The design file is in the xml file format and has two entries under Devices -one for the PSoC™ and one for the connectivity device. Change the name for the connectivity device to the device you are using such as CYW43012C0WKWBG. For example:

```
<Device mpn="CYW43012C0WKWBG">
```

Now, save the *design.modus* file. Since we already changed the `TARGET` variable in the *Makefile* in Generate custom TARGET, the C files specifying the device configuration will be re-generated during the next application build.

5 Remove connectivity device

If you want to remove the connectivity device, you need to again modify the *design.modus* and *Makefile* as done in the previous step.

5.1 Modifying the CUSTOM_BSP_NAME.mk

1. Remove the string assigned to the variable `ADDITIONAL_DEVICES` and leave it empty.
2. Remove the `COMPONENTS` associated with the connectivity device by referring to the table under Modifying the CUSTOM_BSP_NAME.mk.

5.2 Modifying the design.modus

1. Open *design.modus* in a text editor.
2. Delete the section on the connectivity device from the line starting with `<Device mpn=" CYW43012C0WKWBG">` to the line ending with `</Device>`.
3. Now, save the design.modus file. Since we already changed the TARGET variable in the Makefile in Generate custom TARGET, the C files specifying the device configuration will be re-generated during the next application build.

6 Library compartmentalization

We discussed how the software libraries use compartmentalization through the COMPONENTS makefile variable to add or drop certain libraries to the build process and how the BSP adds libraries that are required for every application using that BSP. For example, the Wi-Fi & Bluetooth® libraries utilize the COMPONENTS makefile variable to deliver firmware for a specific family of devices. Let's take a deeper look at each of the libraries to understand better.

The AIROC™ CYW43xxx Wi-Fi & Bluetooth® combo device consists of two CPUs; one for handling Wi-Fi and the other for handling Bluetooth®. The Wi-Fi and Bluetooth® firmware is supplied in the form of compiled binaries and not as source files. The Wi-Fi Host Driver and Bluetooth® libraries contain these binaries in addition to providing transport layer support for communicating between the PSoC 6™ MCU and the AIROC™ CYW43xxx device.

6.1 Bluetooth® library

In the Bluetooth® library, the device specific firmware files are located inside the *firmware* directory. The first level of hierarchy employed here is to distinguish the firmware based on the family of devices i.e., firmware files for all modules based on AIROC™ CYW4343W will reside in COMPONENT_4343W, whereas modules based on AIROC™ CYW43012 will reside in COMPONENT_43012 and so on.

In the next level of hierarchy, the firmware files are distinguished based on the vendor of the module. For instance, COMPONENT_43012 has three subdirectories inside for Murata, USI, and Infineon Technologies provided modules, which are COMPONET_MURATA_1LV, COMPONENT_CYSBSYS-RP01, and COMPONENT_WM-BAC-CYW50, respectively. The reason why different vendors require different firmware files is due to differences in the device packages used, antenna tuning properties, clock configuration etc.

The next level of hierarchy is based on the communication interface between host PSoC 6™ MCU and the Bluetooth® subsystem of the connectivity device. At the moment only UART is supported. Making any changes to the Bluetooth® firmware will require you to get in contact with Infineon Technologies engineering teams through Salesforce Tech Support cases.

6.2 Wi-Fi Host Driver

In the Wi-Fi Host Driver, the firmware files are provided in the *WiFi_Host_Driver/resources* directory of the library. There are multiple binaries provided as part of the library. There are two subdirectories under *resources* of interest – *firmware* and *nvram*:

6.2.1 firmware directory

The *firmware* directory houses the Wi-Fi firmware for the device using only one level of hierarchy based on the device family to distinguish between different devices. There is no further division based on the module vendor. There are three types of binaries provided. Each contains a C array:

- MFG-TEST - This firmware binary is used only for testing the device performance for the certification received by the device.
- Production - This firmware binary is used for real-world applications.
- CLM - This firmware binary contains the database of regulatory information for each country comprising of permitted channels of operation, maximum Tx power permitted etc. The CLM binary is available for both MFG-TEST and Production firmware as separate files.

6.2.2 nvram directory

The *nvram* directory contains C arrays defined in a header file which dictates the conditions under which the Wi-Fi firmware operates. The NVRAM file contains details of the board id, Tx/Rx antenna configuration, clock configuration etc. The *nvram* directory follows two levels of hierarchy based on the device family and the module vendor similar to the Bluetooth® firmware files.

Similar to Bluetooth® firmware files, making modifications to these binaries in the Wi-Fi Host Driver require you to get in contact with Infineon Technologies engineering teams through Salesforce Tech Support cases.

Revision history

Revision	Date	Description
**	08/09/2021	New document.
*A	09/10/2021	Updated acronyms. Removed make import_deps step.
*B	10/29/2021	Updated code for CLM blob in section 4.1.

Edition 2021-10-29

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-33767 Rev. *B