MPLAB Harmony v3 Application Development Guide for MPLAB Harmony v2 Users

AN3388

Introduction

This document is intended to guide MPLAB® Harmony v2 users on how to develop applications using MPLAB Harmony v3.

MPLAB Harmony is a software framework consisting of compatible and interoperable modules, such as peripheral libraries (PLIBs), drivers, system services, middleware, and third-party libraries. MPLAB Harmony v3 has the same basic principles as MPLAB Harmony v2; however, new features and enhancements were implemented for this release.

Table of Contents

1. Tools and Installation

MPLAB Harmony v2 is packaged in a zip file which contains all the software components or modules required to develop the application, such as MPLAB Harmony Configurator (MHC) files, PLIBs, drivers, multiple middleware libraries, and third party libraries.

In contrast, MPLAB Harmony v3 has a modular download feature. The MPLAB Harmony v3 resources are grouped in different packages and users can download only the package required for the project. These resources are easily configurable through the MPLAB Code Configurator (MCC) graphical user interface (GUI). MPLAB Harmony v3 packages can be downloaded from GitHub. For additional information on the MPLAB Harmony v3 environment setup, refer to the document How to Setup MPLAB Harmony v3 Software Development Framework.

Note: Throughout this document, the words ‘modules’ and ‘device resources’ are used interchangeably.

2. MPLAB Code Configurator GUI

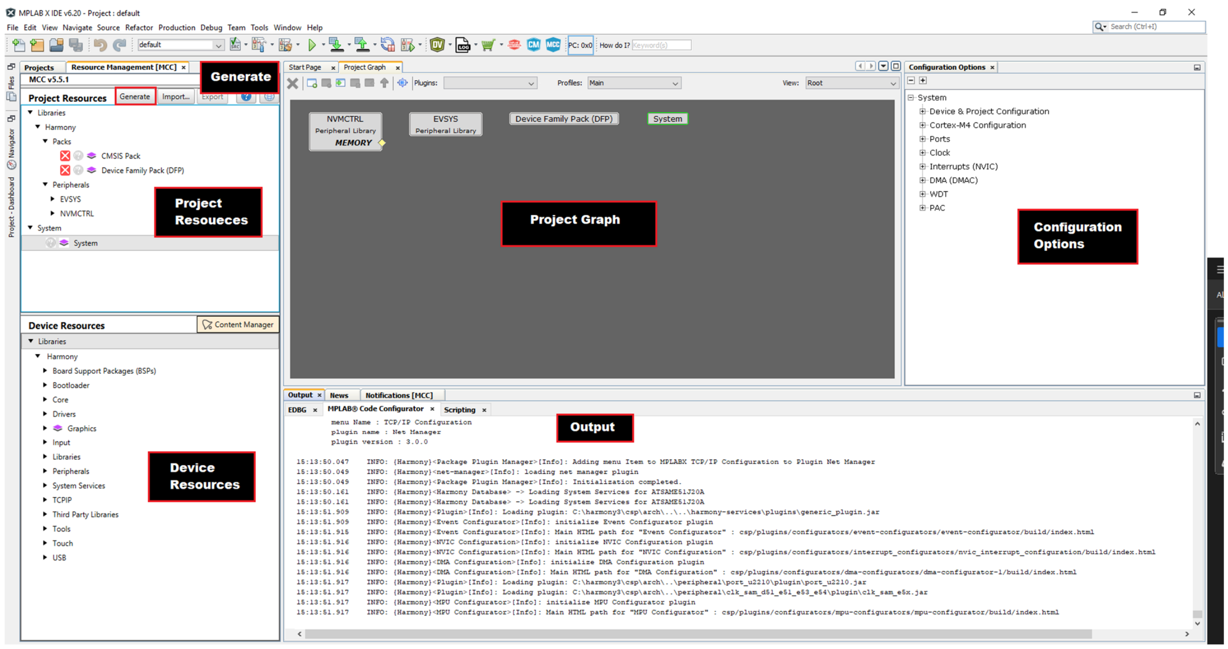
The MPLAB Code Configurator (MCC) is a GUI-based tool that provides an easy way to enable, configure and generate codes for various MPLAB Harmony modules. In MPLAB X IDE, the MCC plugin is installed by default.

To update the MCC plugin to the new version for MPLAB Harmony v3, follow these steps:

1. Open MPLAB X IDE.
2. Select *Tools > Plugins*, and then click on the **Installed** tab. The MCC can be found in the list of installed plugins.
3. In the Installed tab, if an update is available, an Update button will appear. Click on the **Update** button to install the latest version of the MCC plugin.
4. Select **Restart Now** to apply the updates.

In MPLAB Harmony v2, all modules are arranged in a tree structure, and can be configured by expanding the '+' sign. However, in MPLAB Harmony v3, multiple windows are available to make the configuration easier and intuitive. The following figure shows the MPLAB Code Configurator window.

Figure 2-1. MPLAB Code Configurator Window



```
     menu Name : TCP/IP Configuration
     plugin name : Net Manager
     plugin version : 3.0.0

15:13:50.047   INFO: {Harmony}<Package Plugin Manager>[Info]: Adding menu Item to MPLABX TCP/IP Configuration to Plugin Net Manager
15:13:50.049   INFO: {Harmony}<net-manager>[Info]: loading net manager plugin
15:13:50.049   INFO: {Harmony}<Package Plugin Manager>[Info]: Initialization completed.
15:13:50.161   INFO: {Harmony}<Harmony Database> -> Loading System Services for ATSAME51J20A
15:13:50.161   INFO: {Harmony}<Harmony Database> -> Loading System Services for ATSAME51J20A
15:13:51.909   INFO: {Harmony}<Plugin>[Info]: Loading plugin: C:\harmony3\csp\arch\..\..\harmony-services\plugins\generic_plugin.jar
15:13:51.909   INFO: {Harmony}<Event Configurator>[Info]: initialize Event Configurator plugin
15:13:51.915   INFO: {Harmony}<Event Configurator>[Info]: Main HTML path for "Event Configurator" : csp/plugins/configurators/event-configurators/event-configurator/build/index.html
15:13:51.916   INFO: {Harmony}<NVIC Configuration>[Info]: initialize NVIC Configuration plugin
15:13:51.916   INFO: {Harmony}<NVIC Configuration>[Info]: Main HTML path for "NVIC Configuration" : csp/plugins/configurators/interrupt_configurators/nvic_interrupt_configuration/build/index.html
15:13:51.916   INFO: {Harmony}<DMA Configuration>[Info]: initialize DMA Configuration plugin
15:13:51.916   INFO: {Harmony}<DMA Configuration>[Info]: Main HTML path for "DMA Configuration" : csp/plugins/configurators/dma-configurators/dma-configurator-1/build/index.html
15:13:51.917   INFO: {Harmony}<Plugin>[Info]: Loading plugin: C:\harmony3\csp\arch\..\peripheral\port_u2210\plugin\port_u2210.jar
15:13:51.917   INFO: {Harmony}<Plugin>[Info]: Loading plugin: C:\harmony3\csp\arch\..\peripheral\clk_sam_d51_e51_e53_e54\plugin\clk_sam_e5x.jar
15:13:51.917   INFO: {Harmony}<MPU Configurator>[Info]: initialize MPU Configurator plugin
15:13:51.917   INFO: {Harmony}<MPU Configurator>[Info]: Main HTML path for "MPU Configurator" : csp/plugins/configurators/mpu-configurators/mpu-configurator/build/index.html
```

2.1 Device Resources

All the modules, for which corresponding package has been downloaded, are listed in this window. Board Support Package (BSP) and Peripherals list the modules which are applicable for the device for which project is created. The rest of the components are listed for all the devices. The MPLAB Harmony drivers and system services can be found under the Harmony component.

2.2 Project Resources

This window lists all the modules used in the project. To use a module, users need to select the module and click on the button (green cross) from the Device Resources list. By default, some of the modules are added in the Project Resources list, such as Device Family Pack (DFP), System. To remove a module from use or from the Project Resources list, select the module and then click on the button (red cross) appearing in the left top-side of the Project Resources window.

Notes: The following MPLAB Harmony v3 modules must be available in the Project Resources list:

- System
- DFP
- CMSIS (appears for Arm Cortex devices only)

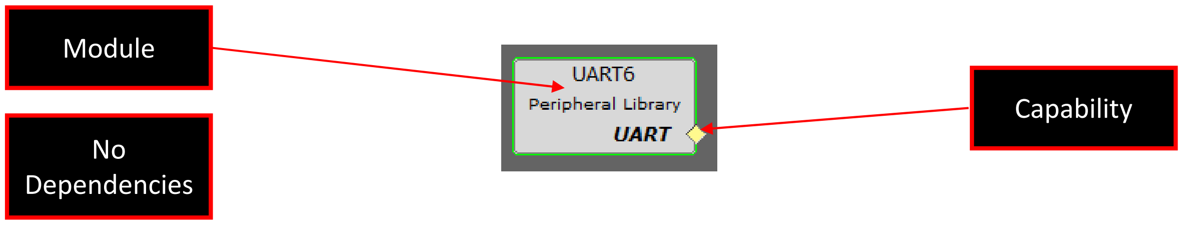
2.3 Project Graph

The Project Graph shows all the modules being used in the project with more details. In the MPLAB Harmony software framework, few modules are closely associated with Microcontroller hardware, such as Peripheral libraries, and few modules are not directly associated with the hardware, but are dependent on the hardware associated modules. For example, MPLAB Harmony drivers are dependent on peripheral libraries. There are also software modules which are dependent on MPLAB Harmony drivers, such as File system service, middleware libraries.

Since MPLAB Harmony has such varied software modules with dependency structure, a monolithic configuration tree, which was present in MPLAB Harmony v2, sometimes becomes difficult to manage and configure. This problem is addressed in MPLAB Harmony v3 by the Project Graph window. In the Project Graph window, each module is shown in the form of a rectangular box. Towards the left-side edge of these rectangular boxes, there are small boxes (Dependency box) shown, which indicate the dependency of the module. Having a dependency means, the module is dependent on some other module for its correct functionality. For example, the USART Driver module has dependency on the UART (or USART) PLIB. Similarly, towards the right-edge of the rectangular boxes, there will be small boxes (Capability box) shown, indicating capability of the module. Having capability means the module is exposing its features to other modules. For example, the UART PLIB module has the capability of the UART, the SERCOM PLIB module has the capabilities of the UART, I^{2}C and SPI. The following figures show examples of modules, their dependency, and capability:

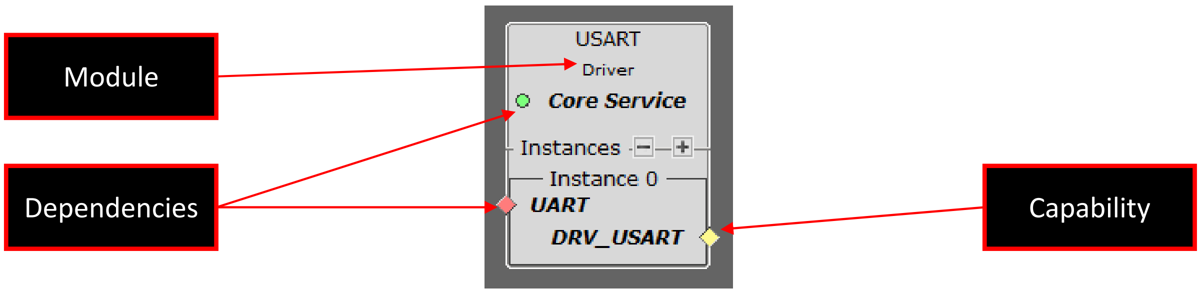
Note: Few modules will not have any dependency or capability, for example, PLIB modules do not have any dependency and the System module has neither dependency nor capability.

Figure 2-2. Capability of a PLIB Module



- UART6 Peripheral Library (PLIB) is the module
- No dependency for UART6 PLIB
- UART6 PLIB has one capability: UART

Figure 2-3. Dependency and Capability of a Driver Module

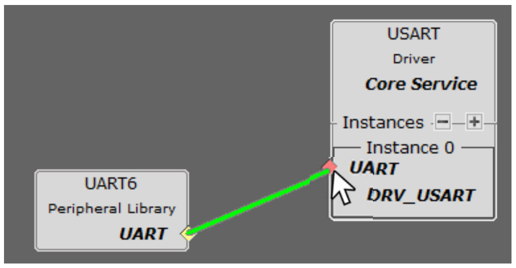


- USART Driver is the module
- Two dependencies on the USART driver: Core Service and UART
- USART Driver has one capability: DRV_USART

On the project graph, a dependent module can be connected to a corresponding capable module to use dependent module correctly. This step of connecting two modules to use the dependent module is called as satisfying the dependency. The dependency can be satisfied in the following three different ways:

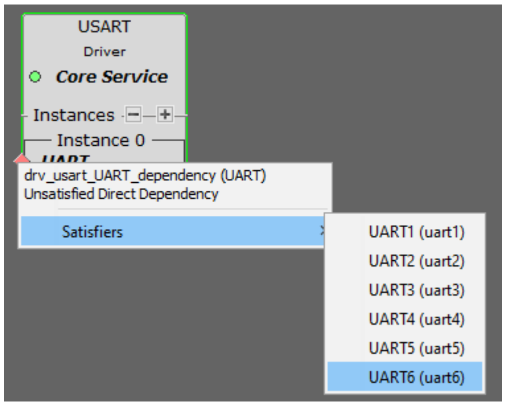
1. Click and then hold and drag the mouse pointer between the dependency box and capability box as shown in the following figure:

   Figure 2-4. Satisfying Dependency 1

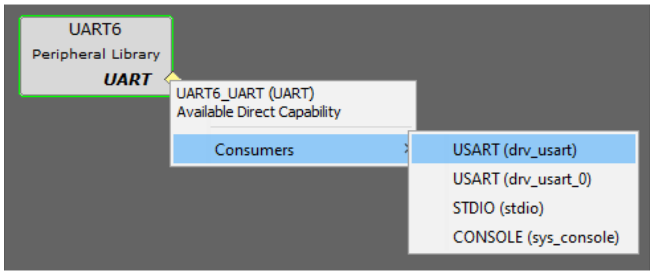
2. Right-click on the dependency box, and then select the appropriate satisfier as shown in the following figure:

Figure 2-5. Satisfying Dependency 2



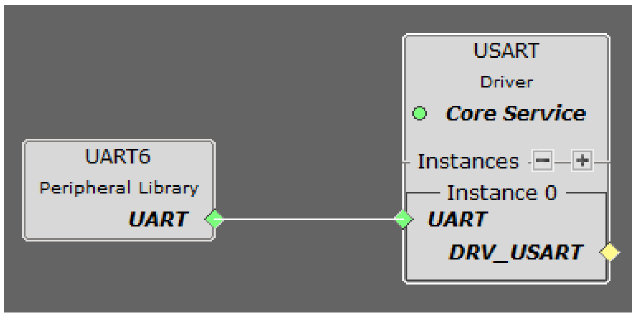
3. Right-clcik on the capability box, and then select the appropriate consumer as shown in the figure below:

Figure 2-6. Satisfying Dependency 3



Ensure that all the dependencies of the modules are satisfied before configuring the modules and generating the code. The following figure shows how the modules look after satisfying dependency:

Figure 2-7. Connected Modules



Additionally, users can move module boxes within the Project Graph Window (click and then hold and drag) to arrange them appropriately to ensure the project graph appears organized.

2.3.1 Dependency

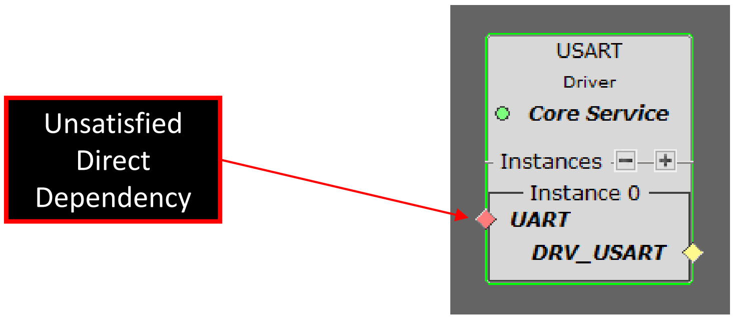
If a module has any dependency, it is shown in the form of a small box (circular, rhombus, or square) in the left side of the module block in the Project Graph:

- The green color of the box indicates a dependency is already satisfied.
- The pink color of the box indicates it needs to be satisfied.
- The yellow color of the box indicates it is optional to satisfy this dependency.

The following are three dependencies:

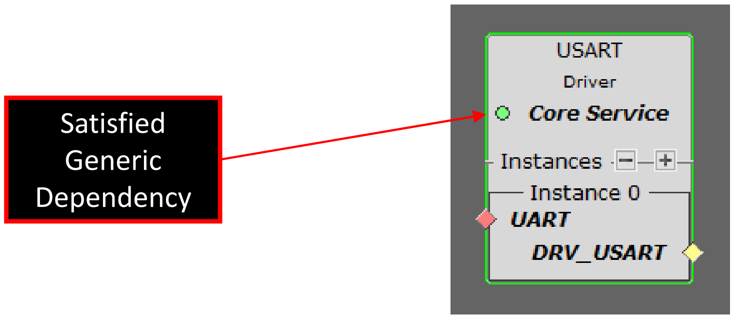
- **Direct Dependency:** Needs manual connection to satisfy them. They are indicated by a small rhombus symbol.

Figure 2-8. Direct Dependency



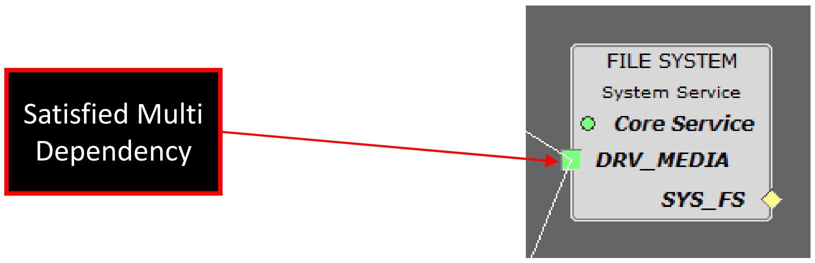
- **Generic Dependency:** Do not need any connections to satisfy them. They turn to green (get satisfied) automatically when any other module present in the project graph providing the capability. They are indicated by a small circular symbol.

Figure 2-9. Generic Dependency



- **Multi Dependency:** Enables many connections to be made to satisfy them. They are indicated by a small square symbol.

Figure 2-10. Multi Dependency



2.3.2 Capability

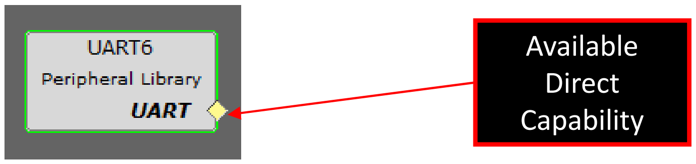
If a module has any capability, it is shown in the form of a small box (circular, rhombus, or square) in the right-side of the module block in the Project Graph.

- The green color of the box indicates capability is already satisfied.
- The pink color of the box indicates it needs to be satisfied.
- The yellow color of the box indicates it is optional to satisfy this dependency.

The following are three capabilities:

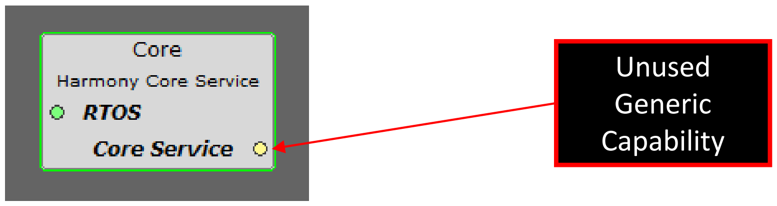
- **Direct Capability:** Need a manual connection to be made to use them. They are indicated by a small rhombus symbol.

Figure 2-11. Direct Capability



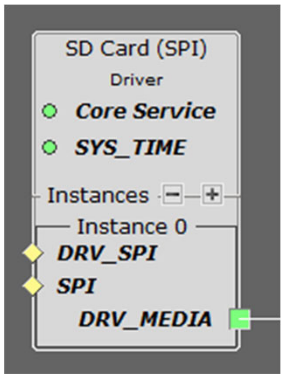
- **Generic Capability:** Do not need any connections to be made to use them. They turn to green (get used) when there is another module present in the project graph depending on them. They are indicated by a small circular symbol.

Figure 2-12. Generic Capability



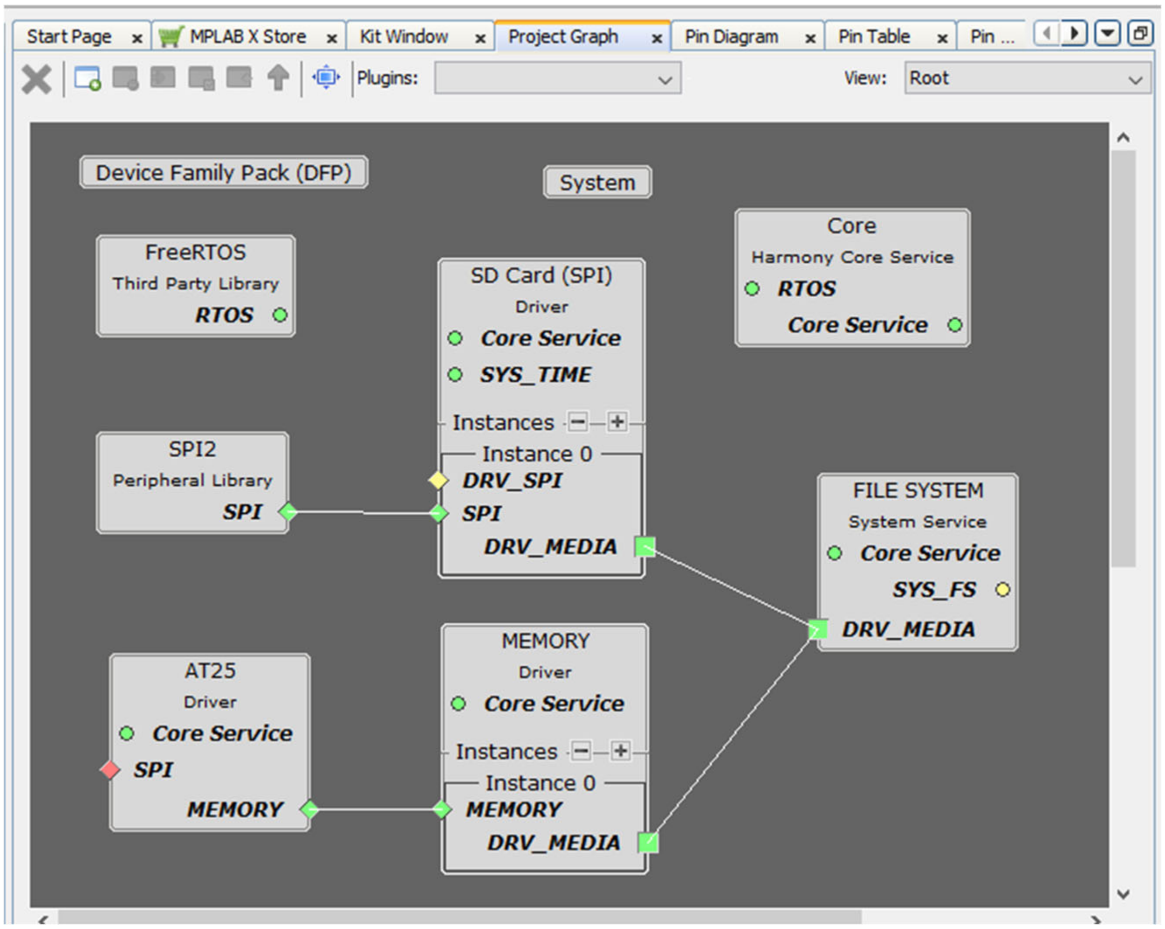
- **Multi Capability:** Must be used to satisfy the multiple dependencies of modules. They are indicated by a small square symbol.

Figure 2-13. Used Multi Capability



The figure below shows the MCC Project Graph module dependencies and capability status:

Figure 2-14. MCC Project Graph



- Device Family Pack and System modules do not have any capability or dependency.
- The core module has optional generic dependency on RTOS. Since there is no module on the project graph with capability of RTOS, this dependency remains unsatisfied. However, since it was an optional dependency, it is all right.

- The core module provides a generic capability of the Core Service. Since it is a generic capability, the SD Card Driver, AT25 Driver, MEMORY Driver, and the FILE SYSTEM modules have used it without a connection.
- SD Card Driver has generic dependency on the SYS_TIME, because there is no module in the project graph providing the SYS_TIME capability, the SD Card Driver is showing that dependency in small pink color circle. The user must add the Time System Service module to satisfy this dependency to generate the code successfully.
- SPI2 Peripheral Library has Direct capability, which is used to satisfy the dependency of the SD Card Driver Instance 0.
- AT25 Driver has a direct dependency on the SPI. A module which had SPI capability, is used by the SD Card Driver. Therefore, this dependency of the AT25 Driver remains unsatisfied, which is denoted by a small pink color rhombus on the left-side of AT25 Driver box. To proceed further, the user must add another SPI Peripheral Library component in the project graph and satisfy this dependency.
- File System has multiple dependencies, which is satisfied by the multi-capability of the SD Card Driver and MEMORY Driver.

2.4 Configuration Options

Unlike MPLAB Harmony v2, MPLAB Harmony v3 has a separate configuration tree for every module. Once the user selects a module either in the project graph or from the Device Resources list, all the configuration options of the corresponding modules are shown in this window.

2.5 Output

This window is similar to the MHC Output window of MPLAB Harmony v2. It is used to display MCC related messages. For example, if some direct dependency is not satisfied and an attempt is made to generate the code, then the output window prompts an error message.

2.6 Help

MPLAB Harmony v2 has a “Help” window in the right side of the MHC to help the user in configurate the modules. MPLAB Harmony v3 does not have this window implemented; instead, it has an advanced tree view. Several configuration items in the tree view have a tool tip which can be seen by hovering the mouse over the item.

2.7 Generate

The code generation step in MPLAB Harmony v3 is same as in MPLAB Harmony v2. After all the required modules are added and configured, users can click the Generate button to generate and add the MPLAB Harmony v3 code in the project.

Note: Unlike MPLAB Harmony v2, the MPLAB Harmony v3 project does not reference the MPLAB Harmony v3 framework files (source files for peripheral libraries, drivers, and system services) present in the MPLAB Harmony v3 repositories. Instead, all the needed source code is copied from the MPLAB Harmony v3 repositories to the local project directory. This means MPLAB Harmony v3 projects can be easily ported from one computer to another.

2.8 MCC Utilities or Plugins

MPLAB Harmony v3 is similar to MPLAB Harmony v2 as both have utilities which can be used to configure some of the modules that are difficult to configure in tree view, such as Clock Configuration, Pin configuration, ADC Configuration. Unlike MPLAB Harmony v2, in MPLAB Harmony v3 none of these utilities are launched by default. They need to be manually launched through the MCC icon menu option of the MPLAB X IDE (or from the Tools menu of the MPLAB Code Configurator window in Standalone mode). MPLAB Harmony v3 has a few additional utilities, such as DMA Configuration, Interrupt (EVIC) Configuration.

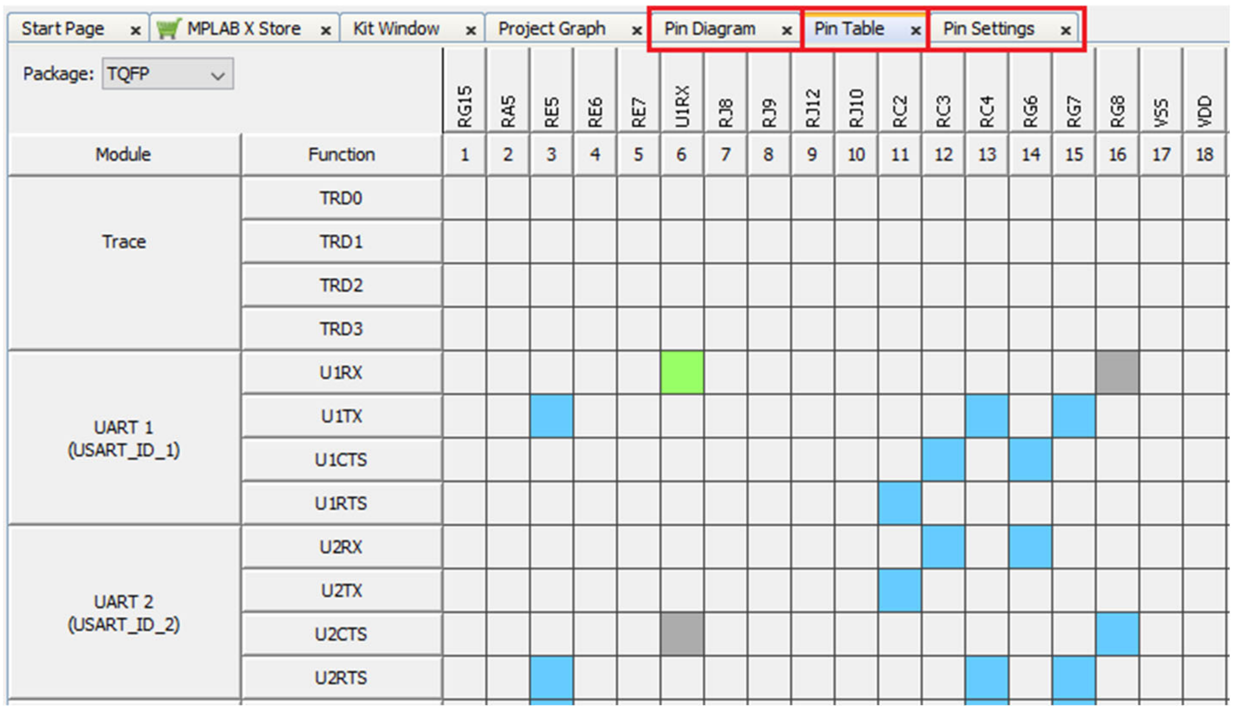
2.8.1 Clock Configuration

The clock configuration utility in MPLAB Harmony v3 is same as in MPLAB Harmony v2, but contains few improvements and bug corrections.

2.8.2 Pin Configuration

This utility in MPLAB Harmony v3 is same as in MPLAB Harmony v2 with a few improvements and bug corrections. In MPLAB Harmony v2, the Pin Table window of the Pin Configuration utility comes towards the bottom alongside the MPLAB Harmony Configurator Output window. However, in MPLAB Harmony v3, all three Pin configuration windows (Pin Diagram, Pin Table and Pin Settings) are placed together as shown in the following figure:

Figure 2-15. MCC Pin Configuration



Package: TQFP

		RG15	RA5	RE5	RE6	RE7	U1RX	RJ8	RJ9	RJ12	RJ10	RC2	RC3	RC4	RG6	RG7	RG8	VSS	VDD
Module	Function	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Trace	TRD0																		
	TRD1																		
	TRD2																		
	TRD3																		
UART 1 (USART_ID_1)	U1RX						🟩										▨		
	U1TX			🟦										🟦		🟦			
	U1CTS												🟦		🟦				
	U1RTS											🟦							
UART 2 (USART_ID_2)	U2RX												🟦		🟦				
	U2TX											🟦							
	U2CTS						▨										🟦		
	U2RTS			🟦										🟦		🟦			

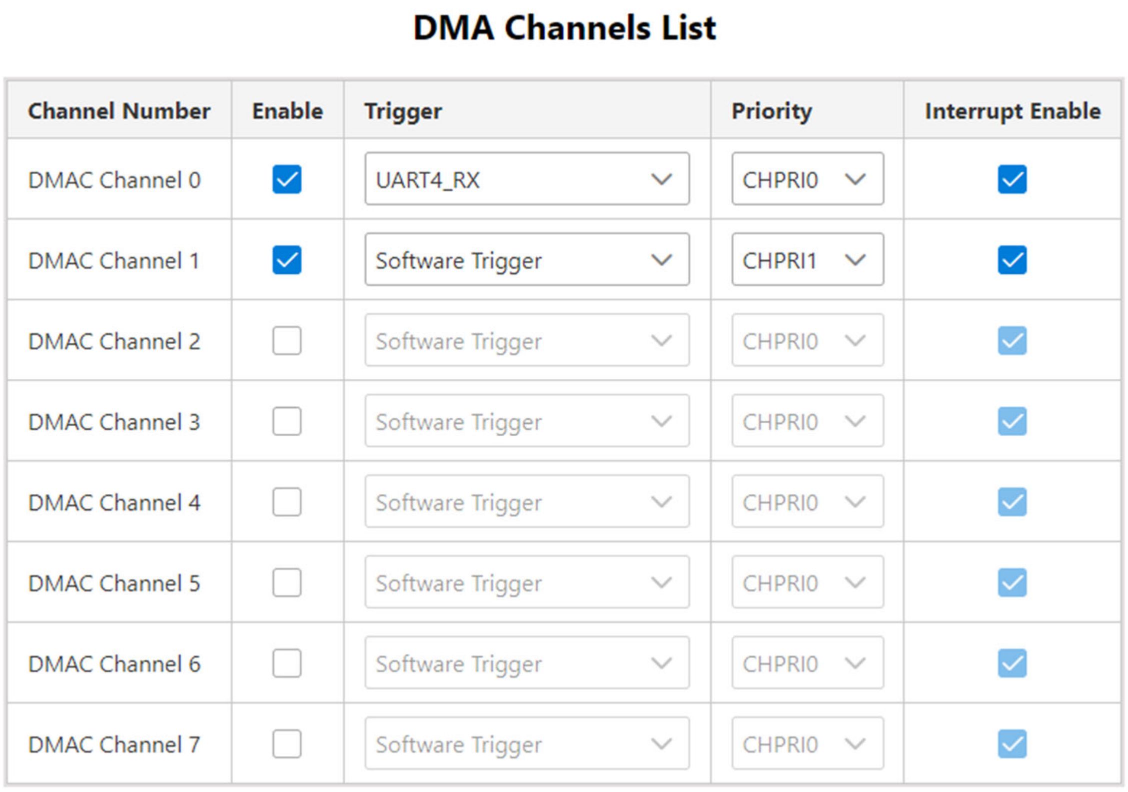
2.8.3 ADC Configuration

This utility in MPLAB Harmony v3 is listed under the *Project Graph > Plugins > ADC Configuration* menu option of MPLAB X IDE (or the Tools menu of the MPLAB Code Configurator window in Standalone mode), only after adding the ADC PLIB on the project graph. The ADC Configuration utility is same in MPLAB Harmony v3 and MPLAB Harmony v2.

2.8.4 DMA Configuration

The DMA Configuration utility of MPLAB Harmony v3 can be used to allocate DMA channels for different transactions as shown in the following figure.

Figure 2-16. DMA Configuration



DMA Channels List

Channel Number	Enable	Trigger	Priority	Interrupt Enable
DMAC Channel 0	✓	UART4_RX	CHPRI0	✓
DMAC Channel 1	✓	Software Trigger	CHPRI1	✓
DMAC Channel 2	☐	Software Trigger	CHPRI0	✓
DMAC Channel 3	☐	Software Trigger	CHPRI0	✓
DMAC Channel 4	☐	Software Trigger	CHPRI0	✓
DMAC Channel 5	☐	Software Trigger	CHPRI0	✓
DMAC Channel 6	☐	Software Trigger	CHPRI0	✓
DMAC Channel 7	☐	Software Trigger	CHPRI0	✓

When the DMA is intended to be used with a PLIB module, such as the UART4 PLIB, then the DMA channels must be configured for the UART4 TX and RX in the DMA Configurator utility and the code can be generated. In the application, the DMA (PLIB or System Service) transfer APIs can be called to do the transfers.

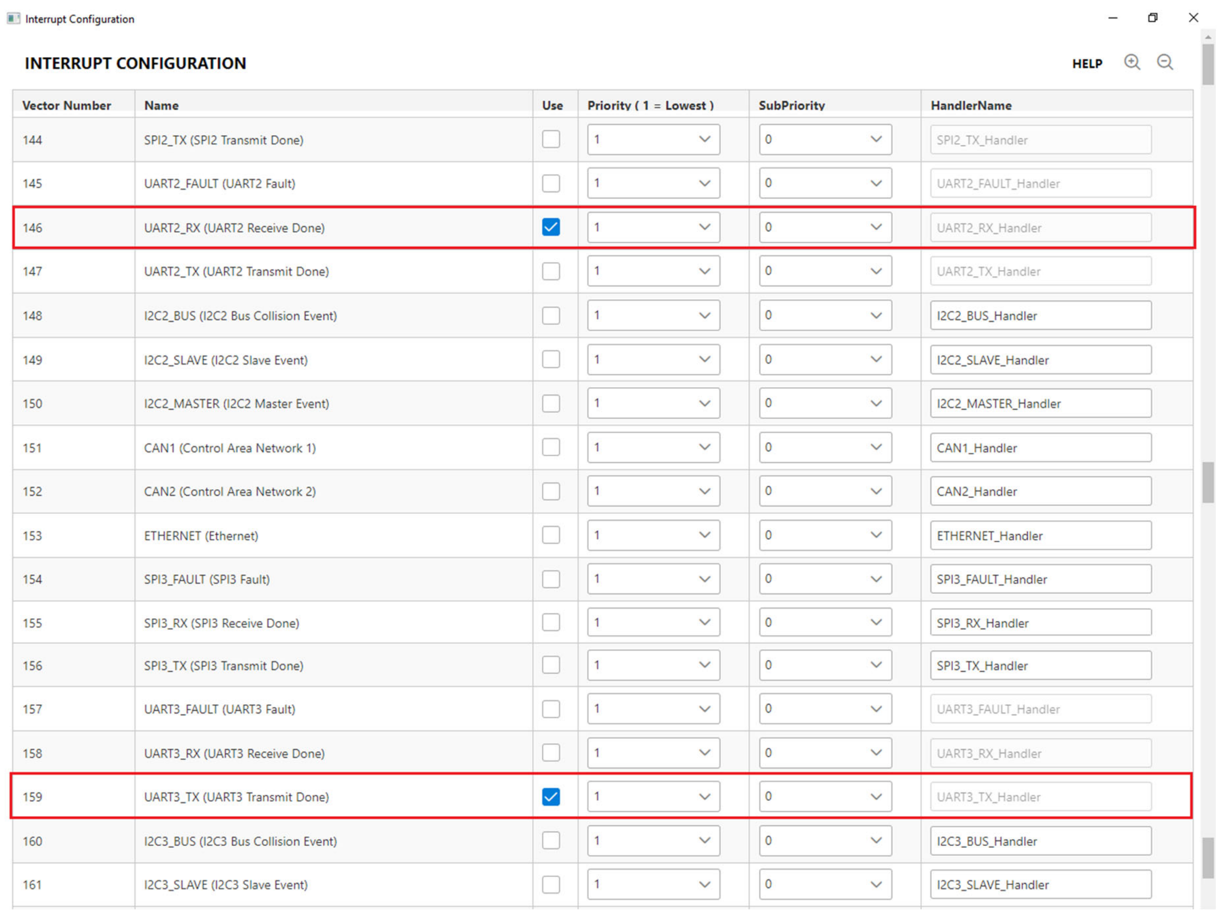
When the DMA is intended to be used with the driver module, DMA mode must be selected in the corresponding driver configuration, then the driver itself configures the required DMA channels. The user does not need to configure the DMA channels, or call any DMA APIs.

2.8.5 Enhanced Vectored Interrupt Controller (EVIC) Configuration

The MPLAB Harmony v3 utility provides the following configuration options:

- **Priorities and Sub-Priorities**: When multiple PLIB modules are used with an interrupt, it is important to configure the priorities and sub-priorities of these interrupts. It must be done using this window and the corresponding code is generated in the `EVIC_Initialize` function of the `plib_evic.c`.
- **ISR Generation**: The ‘Use’ column of the EVIC Configuration utility is used to generate the ISR for corresponding interrupts. By default, the *Use* option for all the interrupts are deselected. When a PLIB module is used in interrupt mode, the MCC automatically selects the corresponding *Use* option in the EVIC Configurator and the ISR is generated in the `interrupts.c` file. If the user wants to generate the ISR of a peripheral even without using its PLIB module, then the *Use* option can be manually selected, and the code can be generated. The ISR name can be configured using the ‘Handler Name’ column. The following figure shows an interrupt configured for the UART2_RX and UART_TX:

Figure 2-17. Interrupt Configuration



Vector Number	Name	Use	Priority (1 = Lowest)	SubPriority	HandlerName
144	SPI2_TX (SPI2 Transmit Done)	☐	1	0	SPI2_TX_Handler
145	UART2_FAULT (UART2 Fault)	☐	1	0	UART2_FAULT_Handler
146	UART2_RX (UART2 Receive Done)	✓	1	0	UART2_RX_Handler
147	UART2_TX (UART2 Transmit Done)	☐	1	0	UART2_TX_Handler
148	I2C2_BUS (I2C2 Bus Collision Event)	☐	1	0	I2C2_BUS_Handler
149	I2C2_SLAVE (I2C2 Slave Event)	☐	1	0	I2C2_SLAVE_Handler
150	I2C2_MASTER (I2C2 Master Event)	☐	1	0	I2C2_MASTER_Handler
151	CAN1 (Control Area Network 1)	☐	1	0	CAN1_Handler
152	CAN2 (Control Area Network 2)	☐	1	0	CAN2_Handler
153	ETHERNET (Ethernet)	☐	1	0	ETHERNET_Handler
154	SPI3_FAULT (SPI3 Fault)	☐	1	0	SPI3_FAULT_Handler
155	SPI3_RX (SPI3 Receive Done)	☐	1	0	SPI3_RX_Handler
156	SPI3_TX (SPI3 Transmit Done)	☐	1	0	SPI3_TX_Handler
157	UART3_FAULT (UART3 Fault)	☐	1	0	UART3_FAULT_Handler
158	UART3_RX (UART3 Receive Done)	☐	1	0	UART3_RX_Handler
159	UART3_TX (UART3 Transmit Done)	✓	1	0	UART3_TX_Handler
160	I2C3_BUS (I2C3 Bus Collision Event)	☐	1	0	I2C3_BUS_Handler
161	I2C3_SLAVE (I2C3 Slave Event)	☐	1	0	I2C3_SLAVE_Handler

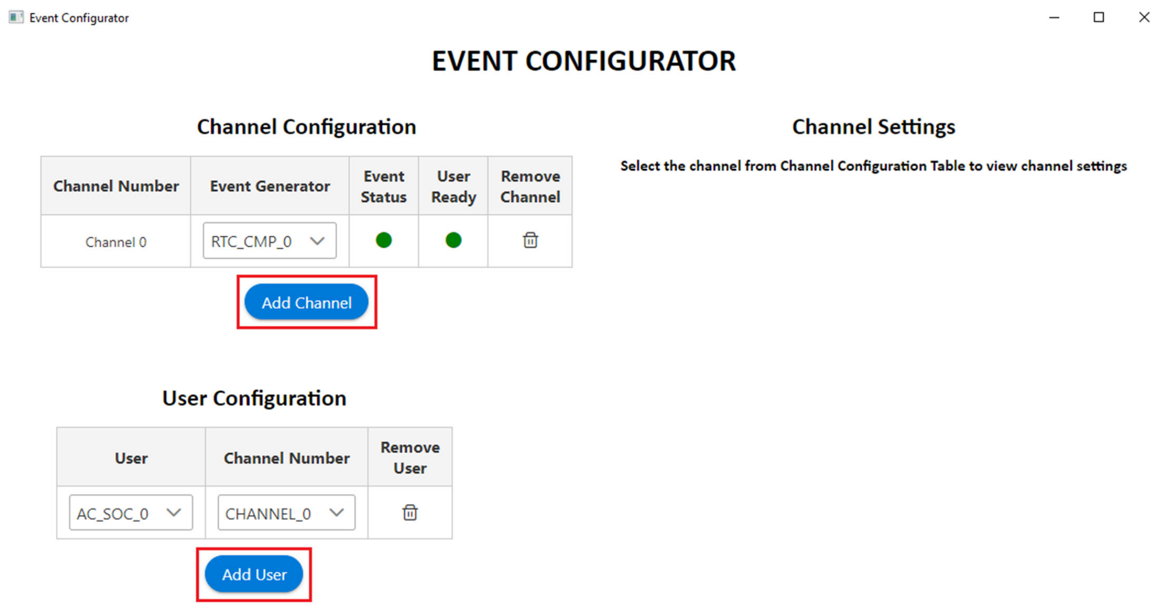
Note: If MCC is selected the 'Use' column of an interrupt, and the user tries to override that setting in the utility, the MCC allows that, but it may cause unpredictable or undesirable behavior.

2.8.6 Event Configurator

The Event Configurator of MPLAB Harmony v3 involves selecting and setting up peripherals to act as event generators and users, then assigning them to specific event channels.

- **Event Generator:** Choose peripherals that will act as event generators. For example, the Real-Time Clock (RTC) can be configured to trigger events.
- **Event User:** Choose peripherals that will act as event users. The Analog Comparator Start of Conversion (AC_SOC) can be set to respond to events generated by the RTC.
- **Event Channel:** Map the RTC (event generator) to the AC_SOC (event user) on a specific event channel, which allows the AC_SOC to start a conversion when the RTC triggers an event.

Figure 2-18. Event Configurator



Event Configurator

EVENT CONFIGURATOR

Channel Configuration

Channel Number	Event Generator	Event Status	User Ready	Remove Channel
Channel 0	RTC_CMP_0	●	●	🗑

Add Channel

Channel Settings

Select the channel from Channel Configuration Table to view channel settings

User Configuration

User	Channel Number	Remove User
AC_SOC_0	CHANNEL_0	🗑

Add User

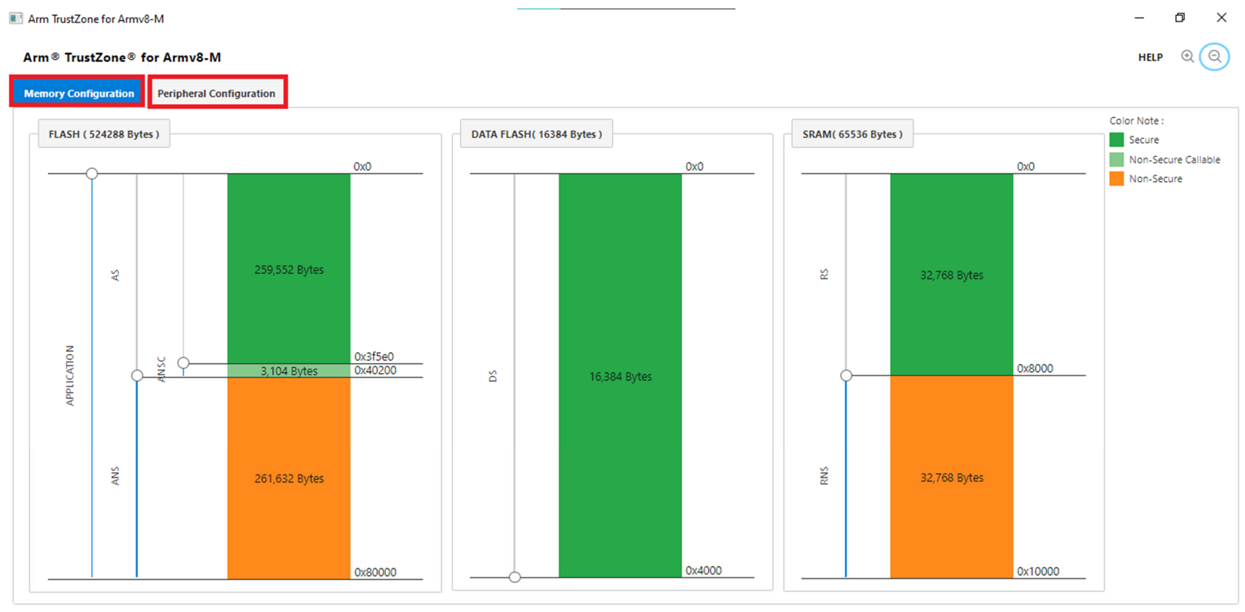
2.8.7 Arm TrustZone for Armv8-M

Arm TrustZone for Cortex-M enables system software to be partitioned into Secure and Non-Secure domains. Secure software can access both Secure and Non-Secure memories and resources, while Non-Secure software can only access Non-Secure memories and resources.

In MPLAB X IDE, the Arm TrustZone for Armv8-M Configuration Window offers two configurations:

1. **Memory Configuration:** This utility allows for the configuration of memory settings.
2. **Peripheral Configuration:** This allows for the designation of peripherals as Secure or Non-Secure.

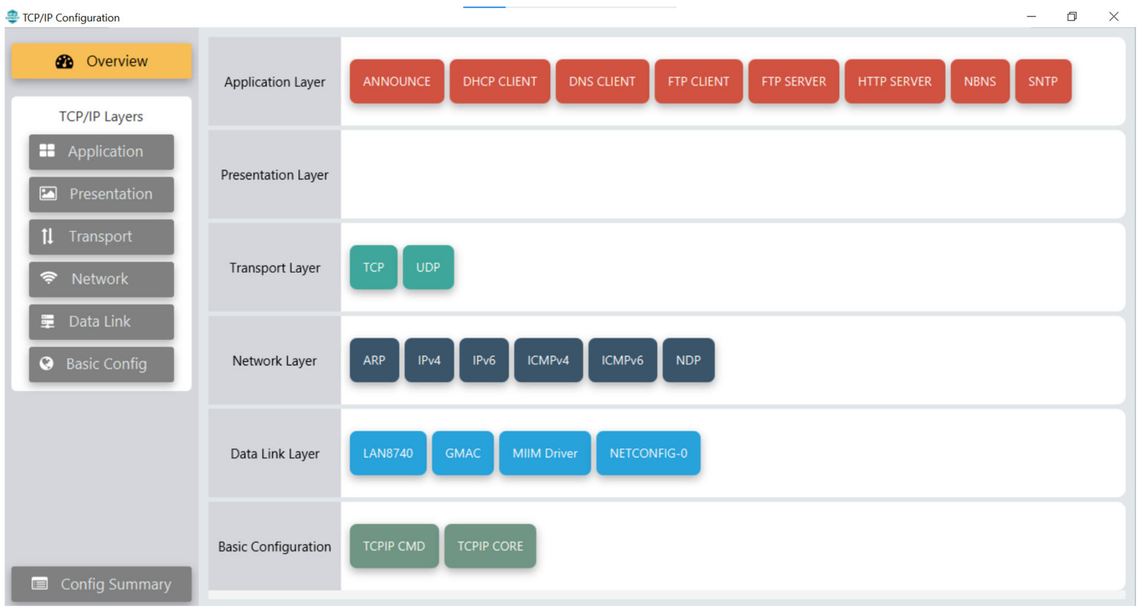
Figure 2-19. Arm TrustZone for Armv8-M



2.8.8 TCP IP Configuration

The TCP/IP Configuration enables a user to graphically add or remove TCP/IP functionality to the MPLAB Harmony v3 project and configure each component per the application demand. The TCP/IP Configurator window initializes with a graphical overview of the different TCP/IP layers. Selecting each layer displays the Device Resources.

Figure 2-20. TCP IP Configuration



3. Peripheral Libraries (PLIBs)

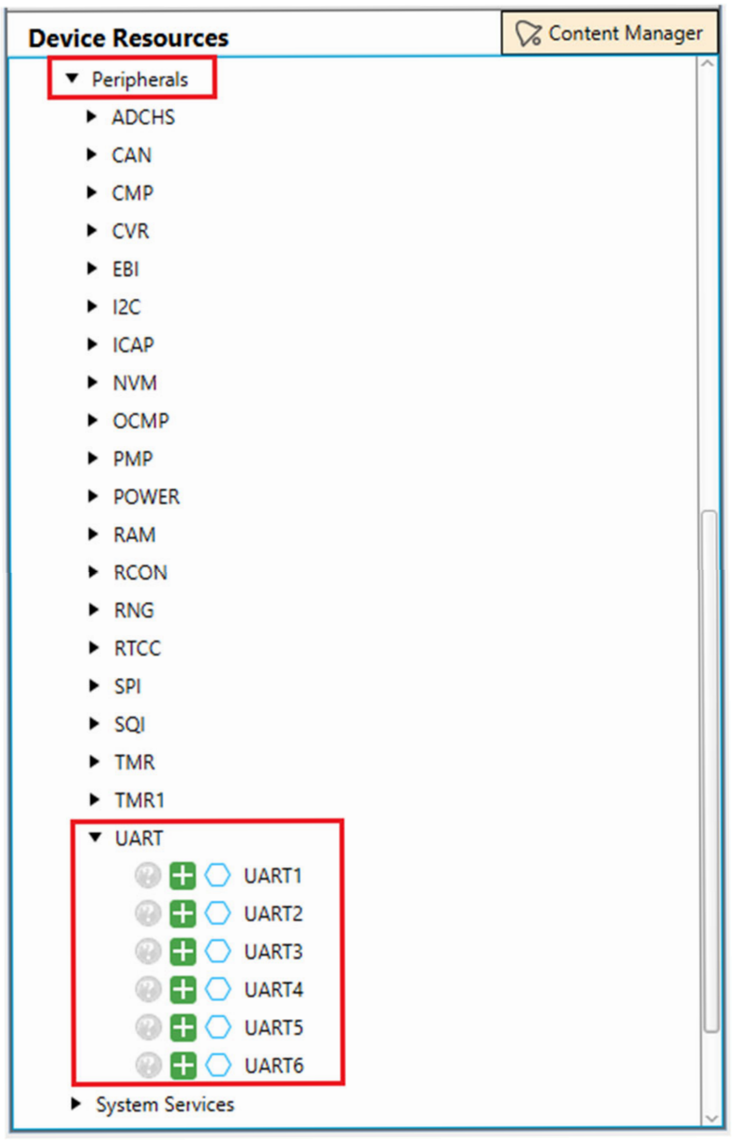
In MPLAB Harmony v2 and MPLAB Harmony v3, the PLIBs interact with Microchip's microcontroller hardware. However, in MPLAB Harmony v3, there are many changes made to the peripheral libraries to make them more user friendly.

3.1 How to Start Using Harmony v3 PLIBs

The MPLAB Harmony v3 PLIBs are part of a Chip Support Package (CSP). They can be used by downloading (or cloning) the csp repository from the MPLAB Harmony GitHub page as described in the Tools and Installation section. The user needs to follow the steps below to configure the MPLAB Harmony v3 PLIBs:

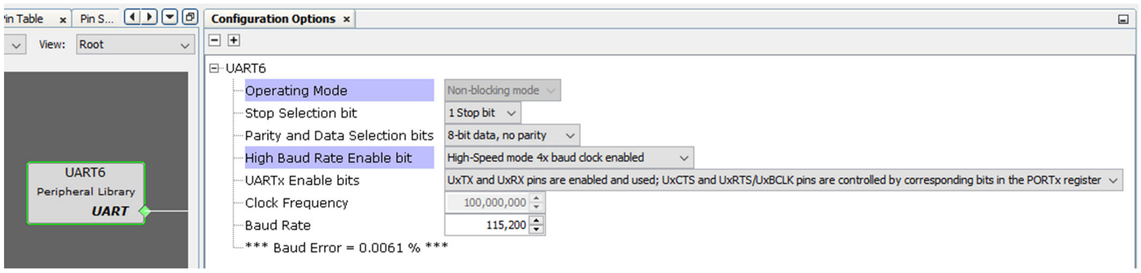
- As described in the MCC section above, all the MPLAB Harmony v3 resources, including PLIBs, are listed in the Device Resources window as shown in the following figure. The PLIB resources may have multiple instances, which denote corresponding multiple hardware instances of that peripheral. For example, a device for which a MPLAB Harmony v3 project is being created, may have six UART peripherals, and these UART peripheral instances are listed inside the UART as shown in the following figure. Double-click on the peripheral instance for which PLIB to be used. This will place the corresponding PLIB component on the project graph.

Figure 3-1. Device Resources - Peripherals



- After the PLIB component is added on the project graph, select it by clicking on it. This will make its configuration options appear on the configuration options towards the right as shown in the following figure. Configure the PLIB as required by the application.

Figure 3-2. Peripheral Configuration



UART6

Operating Mode	Non-blocking mode
Stop Selection bit	1 Stop bit
Parity and Data Selection bits	8-bit data, no parity
High Baud Rate Enable bit	High-Speed mode 4x baud clock enabled
UARTx Enable bits	UxTX and UxRX pins are enabled and used; UxCTS and UxRTS/UxBCLK pins are controlled by corresponding bits in the PORTx register
Clock Frequency	100,000,000
Baud Rate	115,200

*** Baud Error = 0.0061 % ***

- Once PLIB configuration is complete, generate the code and start using the PLIB APIs to develop the application. Code for the finished PLIB configuration is generated inside the `<plib name>_Initilize` function and this function is automatically called from the `SYS_Initialize()` function of the `initilization.c` file.

3.2 Understanding MPLAB Harmony v3 PLIB Generated Code

The following files and folders are generated in the MPLAB Harmony v3 project:

- **peripheral**: Source and header files of all the peripherals used in the project are listed inside the *peripheral* folder. Usually there will be one `.c` and one `.h` PLIB file per peripheral instance. Some of the peripherals and their files are always added by default in the project, such as `gpio`, `clock`, `evic`.
- **definitions.h**: This file is similar to the `system_definitions.h` file of MPLAB Harmony v2. It Includes all the header files required for the application.
- **toolchain_specifics.h**: This file has a few macros defined, which are specific to toolchain.
- **device.h**: This file includes the headers, which have definitions specific to the selected device, such as `xc.h`.
- **exceptions.c**: This file is same as the `system_exceptions.c` file of MPLAB Harmony v2. It defines the exception handler for the application.
- **initialization.c**: This file is same as the `system_init.c` file of MPLAB Harmony v2. It initializes all the MPLAB Harmony modules used in the application.
- **interrupts.c**: This file is same as the `system_interrupt.c` file of MPLAB Harmony v2. It contains definitions of all the interrupt service routines used in the application.
- **stdio/xc32_monitor.c**: A new feature is added in MPLAB Harmony v3 to support the `printf` and `scanf` functions of the standard C library. When the STDIO module (listed in Tools in the Device Resources window) is used, the function definitions inside this file get updated by the MCC to make the `printf` and `scanf` functions work.
- **main.c**: This file is same as in MPLAB Harmony v2. However, in MPLAB Harmony v2 it is recommended to develop the application in `app.c`. In a MPLAB Harmony v3 PLIB only project, it is recommended to do it in `main.c`. MPLAB Harmony v3 driver-based applications are still recommended to be developed in `app.c`.

3.3 MPLAB Harmony v2 and MPLAB Harmony v3 PLIBs Differences

The major differences between MPLAB Harmony v2 and MPLAB Harmony v3 PLIBs are as follows:

- MPLAB Harmony v2 PLIBs do not have any MHC configuration options. The PLIB initialization code must be manually written. However, MPLAB Harmony v3 provides the MCC configuration options for the PLIBs, and based on the configuration it generates the PLIB initialization code.
- MPLAB Harmony v2 PLIBs have APIs to read and write almost every register field of a peripheral module, which requires many APIs per module. Though that gives flexibility, it expects the user to call multiple APIs, sometimes even in a sequence, to perform a task. On the contrary, MPLAB Harmony v3 abstracts the peripheral features and has direct APIs to perform meaningful tasks. It takes care of manipulating multiple register fields and sequencing in the implementation, thereby making the usage simple and quick.
- MPLAB Harmony v2 PLIBs have the same API for a different instance of a peripheral. It provides an argument to distinguish between instances, thereby keeping the API signature the same. However, MPLAB Harmony v3 provides dedicated APIs for different instances.
- MPLAB Harmony v2 provides device-specific pre-compiled library files (in form of `.a`) and header files to use its PLIBs. Based on optimization level, either library file implementation or header files inline implementation gets used. However, MPLAB Harmony v3 generates and adds a `.c` file for every PLIB instance, which has the definitions of the provided interface.

3.4 Application Example Using MPLAB Harmony v2 PLIBs

There are significant differences between MPLAB Harmony v2 and MPLAB Harmony v3 PLIBs, therefore porting from an MPLAB Harmony v2 PLIB-based application to an MPLAB Harmony v3 PLIB-based application is not straight forward. The porting steps will vary based on the MPLAB Harmony modules used by the application.

A simple example is printing a *Hello World* message on the computer console. This is shown to compare the application development steps using the MPLAB Harmony v2 PLIB APIs against the MPLAB Harmony v3 PLIB APIs.

To create an application using the MPLAB Harmony v2 PLIBs, follow these steps:

1. Create the MPLAB Harmony v2 project.
2. Configure using the MHC.
   a. Configure the UART pin: Go to the Pin Setting window of the MHC and configure the UART Transmit pin.
3. Generate the code using the MHC.
4. Update `app.c`.
   a. The following code shows the initialization code required to develop the application. It initializes the UART peripheral.

```
UART_Initialize ()
{
    uint32_t clockSource;

    /* Disable the USART module to configure it*/
    PLIB_USART_Disable (USART_ID_6);

    /* Set the line control mode */
    PLIB_USART_LineControlModeSelect(USART_ID_6, USART_8N1);

    /* We set the receive interrupt mode to receive an interrupt whenever FIFO
       is not empty */

PLIB_USART_InitializeOperation(USART_ID_6,USART_RECEIVE_FIFO_ONE_CHAR,USART_TRANSMIT_FI
FO_IDLE,USART_ENABLE_TX_RX_USED);

    /* Get the USART clock source value*/
    clockSource = SYS_CLK_PeripheralFrequencyGet ( CLK_BUS_PERIPHERAL_1 );

    /* Set the baud rate and enable the USART */
    PLIB_USART_BaudSetAndEnable(USART_ID_6, clockSource, 9600);  /*Desired Baud rate
value*/
}

// *****************************************************************************
// *****************************************************************************
// Section: Application Initialization and State Machine Functions
// *****************************************************************************
// *****************************************************************************

/*******************************************************************************
  Function:
    void APP_Initialize ( void )

  Remarks:
    See prototype in app.h.
 */

void APP_Initialize ( void )
{
    UART_Initialize ();
}
```

Note: In MPLAB Harmony v2, the PLIB for the UART peripheral is named as USART, not UART. All APIs have a prefix of PLIB_USART and not PLIB_UART.

b. The following code has the application logic to show the message on the console.

```
uint8_t count = 0;
uint8_t consoleMsg[] = "Hello World\n\r";

void APP_Tasks ( void )
{
    if (count < sizeof(consoleMsg))
    {
        /* Wait till TX buffer is available */
        while(PLIB_USART_TransmitterBufferIsFull(USART_ID_6));
        /* Send one byte */
        PLIB_USART_TransmitterByteSend(USART_ID_6, consoleMsg[count]);
        count++;
    }
}
```

3.5 Application Example Using MPLAB Harmony v3 PLIBs

The same application used to show a message on the console can be created using MPLAB Harmony v3 PLIBs in the following two possible ways.

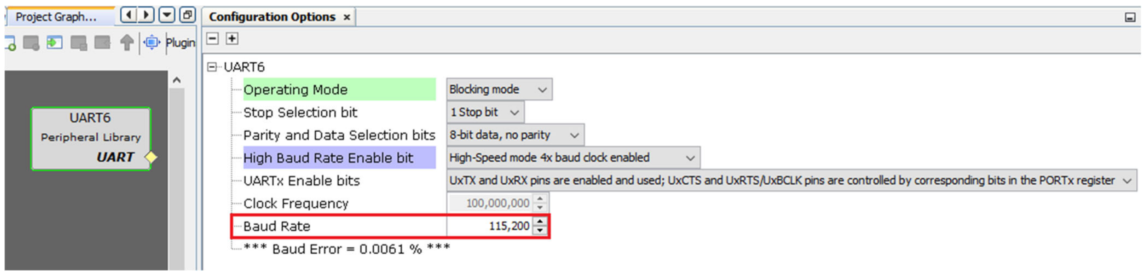
3.5.1 Application Example Using MPLAB Harmony v3 PLIB in Blocking Mode

In blocking mode, the interrupt for the PLIB is disabled. The APIs for the transfer request operate in blocking mode. Because the APIs block until completion, there is no need of any mechanism to check the transfer status.

In this mode, the application can be created using the following steps:

1. Create the MPLAB Harmony v3 project.
2. Follow these steps to configure using the MCC:
   a. Configure the UART pin: Launch the pin manager, go to the Pin Setting window of the MCC and configure the UART Transmit pin.
   b. Follow these steps to configure the UART PLIB:
      i. Add the UART6 Peripheral Library from the Device Resources list.
      ii. Click on the added UART6 PLIB box on the project graph, the configuration options will appear on the right.
      iii. Clear Interrupt mode and change the baud rate to 115,200 as shown in the following figure.

Figure 3-3. Configuring UART PLIB 1



UART6
- Operating Mode: Blocking mode
- Stop Selection bit: 1 Stop bit
- Parity and Data Selection bits: 8-bit data, no parity
- High Baud Rate Enable bit: High-Speed mode 4x baud clock enabled
- UARTx Enable bits: UxTX and UxRX pins are enabled and used; UxCTS and UxRTS/UxBCLK pins are controlled by corresponding bits in the PORTx register
- Clock Frequency: 100,000,000
- Baud Rate: 115,200
- *** Baud Error = 0.0061 % ***

3. Generate the code using the MCC.
4. Update the `main.c` file.
   a. The UART initialization code is already generated by the MCC based on the configuration done in step 2.2 above, hence code is not written to initialize the UART peripheral.

b. The following code shows the two lines of code (shown in bold) used to implement an application to display a message on the console.

```
uint8_t consoleMsg[] = "Hello World\n\r";

int main ( void )
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

    UART6_Write(&consoleMsg[0], sizeof(consoleMsg));

    while ( true )
    {
        /* Maintain state machines of all polled MPLAB Harmony modules. */
        SYS_Tasks ( );
    }

    /* Execution should not come here during normal operation */

    return ( EXIT_FAILURE );
}
```

3.5.2 Application Example Using MPLAB Harmony v3 PLIB in Non-Blocking (interrupt) Mode

In Non-Blocking mode, the interrupt for the PLIB is enabled and the API that submits the transfer request does not block until the transfer completes. Instead, the transfer request API is called, the API triggers the transfer process, and it immediately returns. The CPU continues to run the following instructions while the transfer happens in the background. However, because the interrupts are enabled in this mode, CPU execution is interrupted (to execute Interrupt service routines) after every transfer completion until the whole transfer request is completed. In the Interrupt mode of implementation, the transfer status can be checked in the following two ways. The user can choose one of the methods to check the transfer status as required.

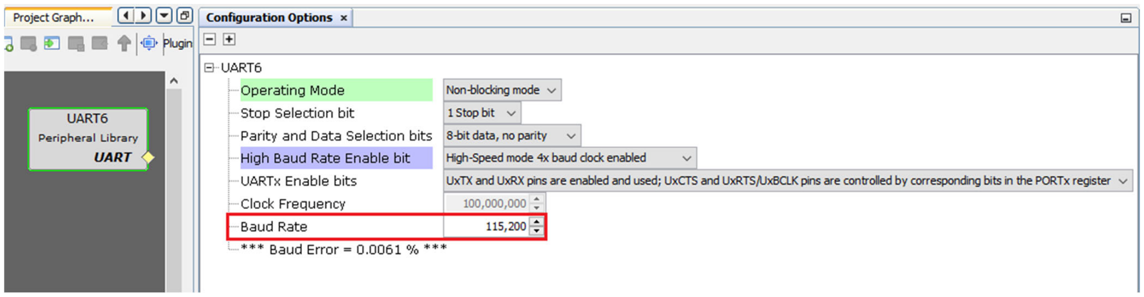
- Status Polling: The MPLAB Harmony v3 PLIBs provide an `IsBusy` API to poll the status of the transfer.
- Callback: The MPLAB Harmony v3 PLIBs provide a callback register API to register the callback. If the callback is registered, the registered callback function is called by the PLIB upon the transfer completion.

3.5.2.1 Application Example Using Status Polling

Follow these steps to create the example application to display a message on the console using the MPLAB Harmony v3 PLIBs in Non-Blocking (interrupt) mode using status polling:

1. Create the MPLAB Harmony v3 project.
2. Configure using the MCC:
   a. Configure the UART Pin: Launch the pin manager, go to the Pin Setting window of the MCC and configure the UART Transmit pin.
   b. Configure the UART PLIB:
      i. Add the UART6 Peripheral Library from the Device Resources list.
      ii. Click on the added UART6 PLIB box on the project graph to see its configuration options in the window to the right.
      iii. Keep the interrupt mode enabled and change the baud rate to 115,200 as shown in the following figure:

Figure 3-4. Configuring UART PLIB 2



UART6
- Operating Mode: Non-blocking mode
- Stop Selection bit: 1 Stop bit
- Parity and Data Selection bits: 8-bit data, no parity
- High Baud Rate Enable bit: High-Speed mode 4x baud clock enabled
- UARTx Enable bits: UxTX and UxRX pins are enabled and used; UxCTS and UxRTS/UxBCLK pins are controlled by corresponding bits in the PORTx register
- Clock Frequency: 100,000,000
- Baud Rate: 115,200
- *** Baud Error = 0.0061 % ***

3. Generate the code using the MCC.
4. Update the `main.c` file.
   a. The UART initialization code is already generated by the MCC based on the configuration done in step 2.2 above, hence code is not required to initialize the UART peripheral.
   b. Update the `main.c` file with the following code to complete the application implementation:

```
uint8_t consoleMsg[] = "Hello World\n\r";

int main ( void )
{
    /* Initialize all modules */
    SYS_Initialize ( NULL );

    UART6_Write(&consoleMsg[0], sizeof(consoleMsg));
    while (UART6_WriteIsBusy());

    while ( true )
    {
        /* Maintain state machines of all polled MPLAB Harmony modules. */
        SYS_Tasks ( );
    }

    /* Execution should not come here during normal operation */

    return ( EXIT_FAILURE );
}
```

Note: The application code in non-blocking (interrupt) mode is similar to that of blocking mode. non-blocking mode has one extra instruction (as shown in bold above) to poll the status of the transfer.

3.5.2.2 Application Using Callback

In the non-blocking method, instead of status polling, the callback mechanism can also be used to check the transfer status. Follow these steps to use the callback mechanism:

1. Register a callback function to the PLIB using a dedicated API given by the PLIB.
2. Define the function which is registered.
3. Make the transfer request.
4. Do other application tasks. Whenever a transfer completes, the registered callback function which is defined, will be called by the PLIB.

Because callbacks are called from the interrupt context, the following guidelines must be followed while defining the callback functions in the application:

- Must be treated like an ISR.
- Must be short.
- Must not call application functions that are not interrupt safe.
- Use volatile keywords for the variables which are accessed both inside of callback function and outside of callback function.

The code example using a callback mechanism is not shown as MPLAB Harmony v2 PLIBs did not have this feature, hence nothing to compare. The user can refer to the PLIB demonstrations in the `csp/apps` folder in the `csp` repository for many PLIB callback examples.

3.6 Comparison Between MPLAB Harmony v2 and MPLAB Harmony v3 PLIB Examples

If application development steps in MPLAB Harmony v2 and MPLAB Harmony v3 are compared for the given example, the following differences can be noticed:

- MPLAB Harmony v3 has one MCC configuration step 2.2 extra
- MPLAB Harmony v3 has one coding step 4.1 less
- MPLAB Harmony v3 has application logic step 4.2 simplified

The previous example is of a very simple application. As the application complexity increases, the differences will become more evident and it becomes much simpler and user friendly to develop applications using MPLAB Harmony v3 PLIBs as compared to MPLAB Harmony v2.

4. Drivers

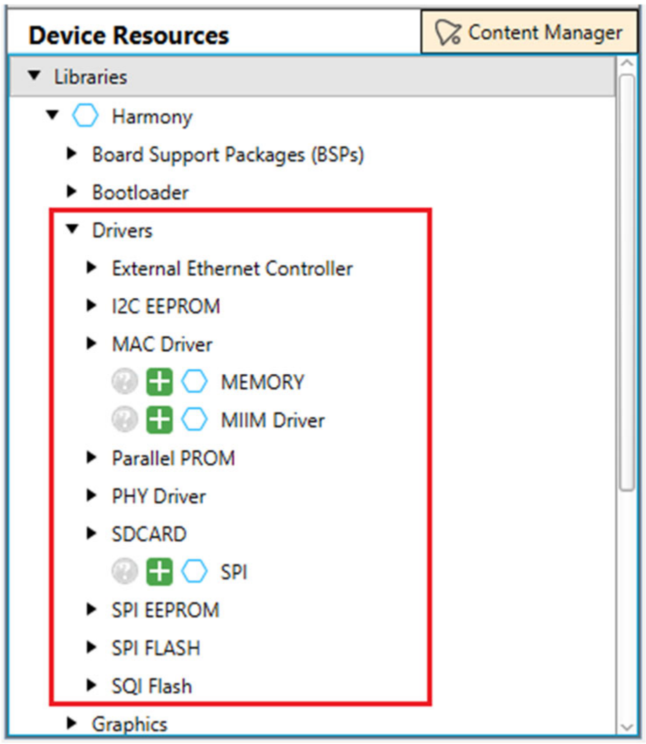
As in MPLAB Harmony v2, MPLAB Harmony v3 drivers also provide a highly abstracted C language interface to peripherals and other resources. A driver's interface allows applications and other client modules (that is, Middleware libraries) to interact with the peripheral it controls.

4.1 Using MPLAB Harmony v3 Drivers

MPLAB Harmony v3 drivers are part of the MPLAB Harmony Core Service and can be used by downloading (cloning) the `core` repository from the MPLAB Harmony GitHub page as described in the Tools and Installation section. Users need to follow these steps to configure and use the MPLAB Harmony v3 drivers:

- As described in the MCC section previously, all MPLAB Harmony v3 resources, including drivers, are listed in the Device Resources section as shown in the following figure. To add a driver component, click on the add component icon.

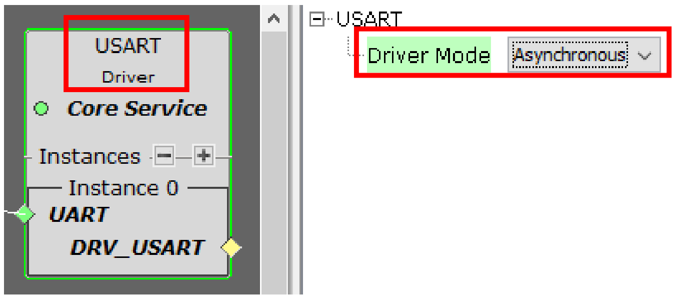
Figure 4-1. Device Resources



- MPLAB Harmony v3 drivers have dependencies that must be satisfied as described in the MCC section above. These dependencies are usually satisfied by the PLIB or System Service components. All the dependencies must be satisfied.
- Configure (in the Configuration Options) all the components which are used to satisfy the dependency. For example, the USART driver will have a dependency on the UART PLIB, hence the UART PLIB must be configured.
  Note: In MPLAB Harmony (both MPLAB Harmony v2 and MPLAB Harmony v3), the driver corresponding to the UART and USART is named as USART driver.
- Drivers can have multiple instances. There are few configuration options which are common for all the instances, where few are instance specific. The driver configuration steps are as follows:

a. Configure common options in the Configuration Options by clicking on the upper portion of the USART Driver box on the Project Graph as shown in the following figure:

Figure 4-2. Driver Common Configuration

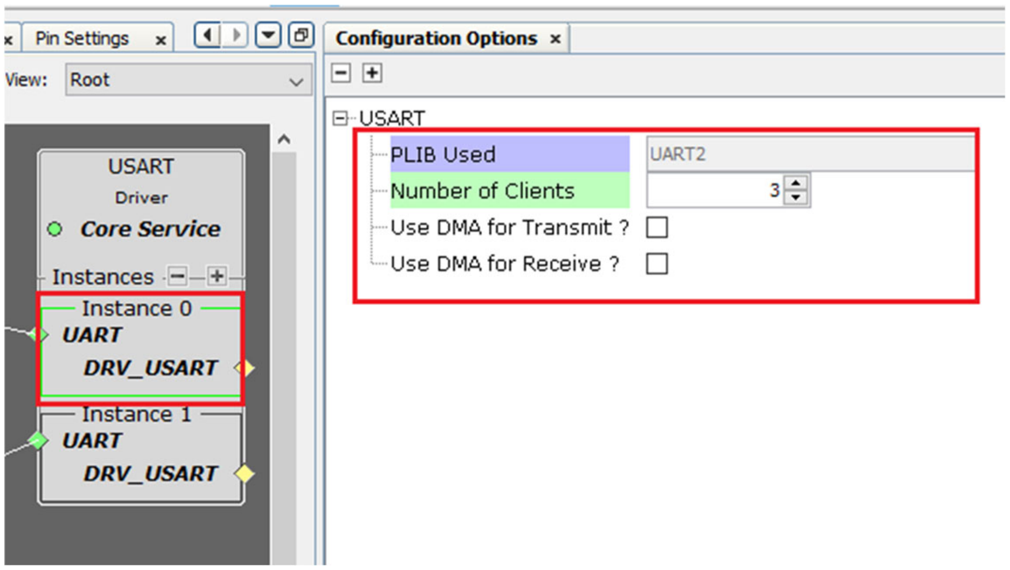


Driver Mode: Asynchronous

b. By default, drivers have one instance (instance number 0). The number of instances can be increased by clicking on the '+' sign and can be reduced by clicking on the '-' sign.

c. Configure instance-specific options by clicking on the respective Instance box on the Project Graph, as shown below:

Figure 4-3. Instance-Specific Configuration



PLIB Used: UART2
Number of Clients: 3
Use DMA for Transmit ? ☐
Use DMA for Receive ? ☐

d. Once the configuration is complete, generate the code and start using the driver APIs to develop the application.

4.2 Understanding MPLAB Harmony v3 Driver Code

When MPLAB Harmony v3 Drivers or System Services are used, the following additional files and folders are generated as compared to a PLIB only project:

- **driver:** Source and header files of all the drivers used in the project are listed inside the driver folder. Usually there will be one `.c` and two `.h` files per driver used.
- **configuration.h:** This file is similar to the `system_config.h` file of MPLAB Harmony v2. It is used to define the configuration macros for drivers, system services and middleware libraries.

- **user.h:** This is a new file in MPLAB Harmony v3, which is by default empty. It should be used to define application specific macros. In MPLAB Harmony v2, those application specific macros are generally defined in the `system_config.h` file.
- **app.h:** This file is same as that in MPLAB Harmony v2, and it is used to develop the application.
- **app.c:** This file is same as that in MPLAB Harmony v2, and it is used to develop the application.

4.3 MPLAB Harmony v2 and MPLAB Harmony v3 Similarities in Drivers

4.3.1 Unique Interface

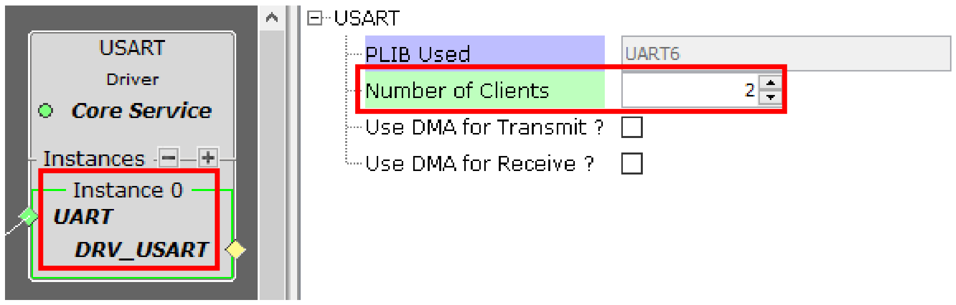
As in MPLAB Harmony v2, MPLAB Harmony v3 drivers also provide unique APIs for peripheral usage across Microchip's 32-bit devices. These unique API interfaces across 32-bit MCU families enable applications developed using the MPLAB Harmony drivers to be ported easily from one device to another. Details of these APIs can be found in the help documentation in the `core` repository.

4.3.2 Multiple Clients Support

As in MPLAB Harmony v2 drivers, MPLAB Harmony v3 drivers also support multiple clients. This allows the application to use one instance of peripheral with different configurations in different contexts without any data interference.

In MPLAB Harmony v3, the maximum number of clients to be used by an application can be configured through the MCC. This can be done under the configuration options for the peripheral-specific driver instance of the MPLAB Harmony driver as shown in the following figure:

Figure 4-4. Multiple Client Configuration



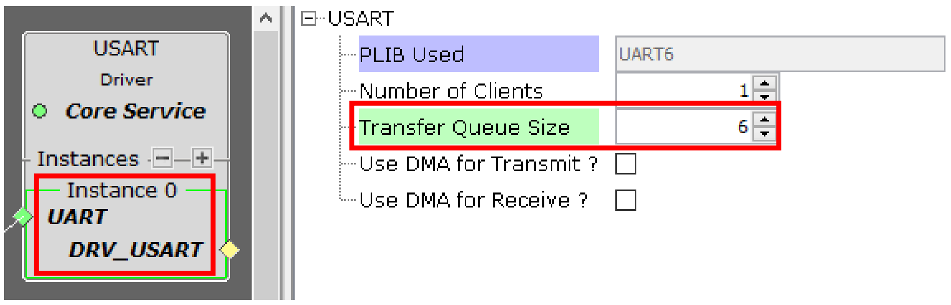
USART
PLIB Used: UART6
Number of Clients: 2
Use DMA for Transmit ? ☐
Use DMA for Receive ? ☐

4.3.3 Buffer Queue Support

As in MPLAB Harmony v2 drivers, MPLAB Harmony v3 drivers also allow buffer queuing. The driver can queue up a client's new request while it is already processing an earlier request. In MPLAB Harmony v3, queuing support is available only for Asynchronous mode of the drivers. Synchronous drivers are blocking in nature, therefore the queuing feature is not applicable (Synchronous and Asynchronous drivers in MPLAB Harmony v3 is discussed in a following section).

In MPLAB Harmony v3, to configure the buffer queue size, first configure the driver in Asynchronous mode through driver's common configurations as shown in Driver's Common Configuration and then the Transfer Queue Size is configured through a peripheral-specific driver instance configuration as shown in the following figure:

Figure 4-5. Configuring Queue Size



USART
PLIB Used: UART6
Number of Clients: 1
Transfer Queue Size: 6
Use DMA for Transmit ? ☐
Use DMA for Receive ? ☐

4.3.4 Real Time Operating System (RTOS) Support

As with MPLAB Harmony v2 drivers, MPLAB Harmony v3 drivers support multiple RTOS.

4.4 MPLAB Harmony v2 and MPLAB Harmony v3 Differences in Drivers

The following are the main differences between MPLAB Harmony v2 and MPLAB Harmony v3 drivers:

4.4.1 API Compatibility

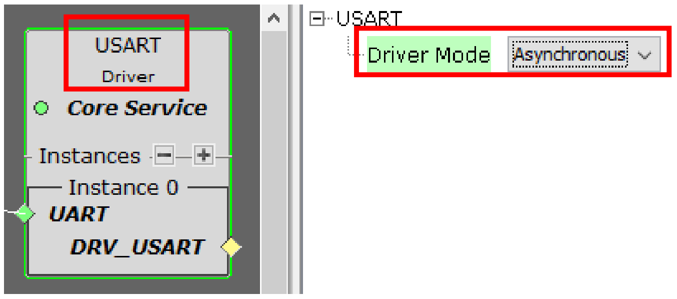
There are few differences in MPLAB Harmony v2 and MPLAB Harmony v3 driver APIs. Because the differences are minimal, an MPLAB Harmony v2 driver-based application can be ported to an MPLAB Harmony v3 application with some changes.

4.4.2 Synchronous and Asynchronous Model

Synchronous drivers are blocking in nature as compared to asynchronous drivers which are non-blocking in nature. Synchronous drivers are recommended to be used with RTOS, and asynchronous drivers are used in a bare-metal (Non-RTOS) environment. Although, Asynchronous drivers are recommended to be used in a Non-RTOS based application environment, they can be used in an RTOS-based application environment where the application must ensure certain tasks yield control to allow the appropriate running of relevant application tasks.

MPLAB Harmony v3 provides synchronous and asynchronous drivers, where MPLAB Harmony v2 provides only an asynchronous model of drivers. In MPLAB Harmony v3, this mode selection can be done in the configuration options by clicking on the upper half of the driver box in the project graph as shown in the following figure:

Figure 4-6. Configuring Driver Mode



USART
Driver Mode: Asynchronous

4.4.3 Interrupt and Polling Mode

In the interrupt mode of a driver, the driver state machine (or task routine) runs from the interrupt service routine. However, in polling mode, the driver state machine runs from a `while(1)` loop. The interrupt mode of drivers provides the best responsiveness where the polling mode is good for debugging purposes.

MPLAB Harmony v2 supports both interrupt and polling modes of drivers. However, MPLAB Harmony v3 supports only interrupt mode (except for a few drivers which are not associated with peripheral interrupts and runs their state machine from the `while(1)` loop, such as the memory driver, and SDSPI driver). Any MPLAB Harmony v2 applications which use the polling mode of a driver, must use the interrupt mode of the driver in MPLAB Harmony v3. However, changing from polling mode to interrupt mode, does not demand any change in the application. Whatever changes are needed in the system and interrupt files, are done by the MCC code generation.

4.4.4 Static and Dynamic Model

MPLAB Harmony v2 static drivers are implemented for a single peripheral instance to reduce the memory footprint of the driver. Sometimes other features of a driver, such as multi-client, buffer queuing, and RTOS support are also removed from static drivers to make them simple and concise. Dynamic drivers are full fledged drivers which support multiple peripheral instances, multi-client, and RTOS.

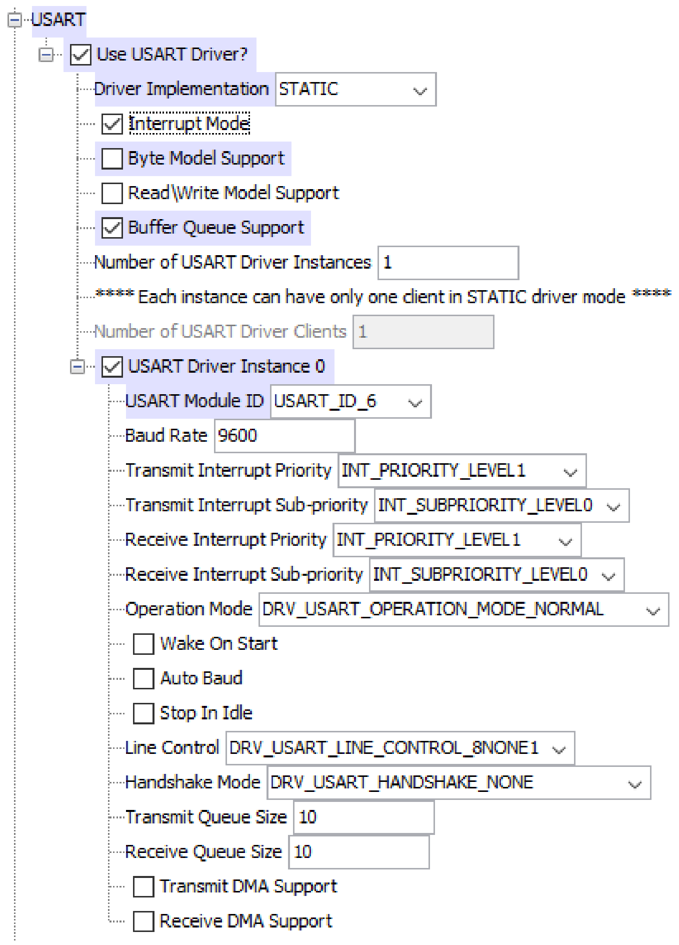
MPLAB Harmony v2 supports both, static and dynamic driver models, where MPLAB Harmony v3 has only dynamic drivers. However, many of the characteristics of MPLAB Harmony v2 static drivers are provided in MPLAB Harmony v3 PLIBs. Applications developed using MPLAB Harmony v2 static drivers can switch to MPLAB Harmony v3 PLIBs or MPLAB Harmony v3 drivers (dynamic drivers) based on their requirements. If an application needs features, such as multiple client, buffer queuing and RTOS, then MPLAB Harmony v3 dynamic drivers can be used; otherwise MPLAB Harmony v3 PLIBs can be considered.

4.5 Application Example Using MPLAB Harmony v2 Driver

To create an application to display a message on a computer console using MPLAB Harmony v2 drivers follow these steps:

1. Create the MPLAB Harmony v2 project.
2. Configure using the MHC.
   a. Configure the UART pin: Go to the Pin Setting window of the MHC and configure the UART Transmit pin.
   b. Configure the USART driver as shown in the figure below (notice the highlighted options are modified):
   Note: In MPLAB Harmony, the driver corresponding to the UART and USART is the USART driver.

Figure 4-7. Configuring MPLAB Harmony v2 USART Driver



```
USART
  ☑ Use USART Driver?
    Driver Implementation  STATIC
    ☑ Interrupt Mode
    ☐ Byte Model Support
    ☐ Read\Write Model Support
    ☑ Buffer Queue Support
    Number of USART Driver Instances  1
    **** Each instance can have only one client in STATIC driver mode ****
    Number of USART Driver Clients  1
    ☑ USART Driver Instance 0
      USART Module ID  USART_ID_6
      Baud Rate  9600
      Transmit Interrupt Priority  INT_PRIORITY_LEVEL1
      Transmit Interrupt Sub-priority  INT_SUBPRIORITY_LEVEL0
      Receive Interrupt Priority  INT_PRIORITY_LEVEL1
      Receive Interrupt Sub-priority  INT_SUBPRIORITY_LEVEL0
      Operation Mode  DRV_USART_OPERATION_MODE_NORMAL
      ☐ Wake On Start
      ☐ Auto Baud
      ☐ Stop In Idle
      Line Control  DRV_USART_LINE_CONTROL_8NONE1
      Handshake Mode  DRV_USART_HANDSHAKE_NONE
      Transmit Queue Size  10
      Receive Queue Size  10
      ☐ Transmit DMA Support
      ☐ Receive DMA Support
```

3. Generate the code using the MHC.
4. Follow these steps to update the application:
   a. The UART initialization code is already generated by the MHC based on the configuration, as described in step 2.2, hence it does not required to be added.
   b. Update the `app.c` file and `app.h` file with application logic. The following figure shows the application logic to be developed in the `app.c` file, which has the following three states:
      i. Open the driver: This state is required for the driver model which supports multiple clients. For example, the dynamic driver. It can be skipped for static drivers as they are single client.
      ii. Queue transfer requests: This state adds the transfer request in the queue. In this application, there is no back-to-back requests to be queued, but a queueing model has been used for representation.
      iii. Check status of the transfer: This state checks the transfer status. The status of the transfer can be checked by these methods:
         1. Polling: Application continuously polls for transfer status using an API. In this example the polling method is used.

2. Callback: Callback can be registered using a dedicated API which can be called once the transfer completes.

```
uint8_t consoleMsg[] = "Hello World\n\r";
void APP_Tasks ( void )
{

    /* Check the application's current state. */
    switch ( appData.state )
    {
        /* Application's initial state. */
        case APP_STATE_OPEN_DRIVER:
        {
            appData.myUSARTHandle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NONBLOCKING);
            if (appData.myUSARTHandle != DRV_HANDLE_INVALID)
            {
                appData.state = APP_STATE_ADD_REQUEST;
            }
            break;
        }
        case APP_STATE_ADD_REQUEST:
        {
            DRV_USART_BufferAddWrite(appData.myUSARTHandle, &appData.bufferHandle,
&consoleMsg[0], sizeof(consoleMsg));
            appData.state = APP_STATE_STATUS_CHECK;
            break;
        }
        case APP_STATE_STATUS_CHECK:
        {
            if (DRV_USART_TRANSFER_STATUS_TRANSMIT_EMPTY &
DRV_USART_TransferStatus(appData.myUSARTHandle))
            {
                // Data has been transmitted, go to next state
                appData.state = APP_STATE_COMPLETE;
            }
            break;
        }
```

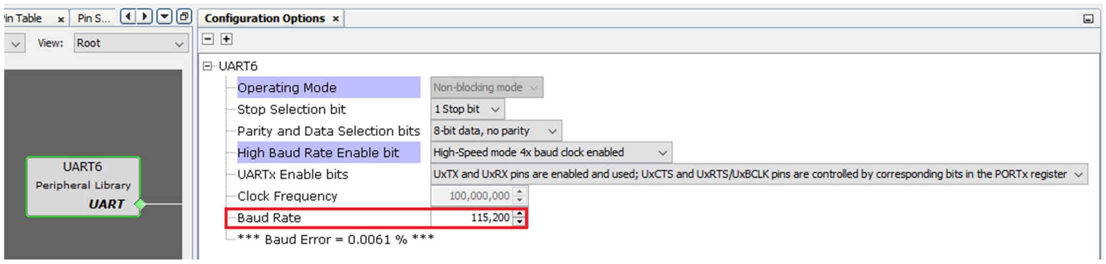
Note: If the user wants to use a dynamic driver or non-interrupt mode of a driver for this application, the application code in the `app.c` file must be the same. The only change is the MCC configuration change, as described in the step 2.2.

4.6 Application Example Using MPLAB Harmony v3 Driver

To create an application to display a message on a computer console using the MPLAB Harmony v3 drivers, follow these steps:

1. Create an MPLAB Harmony v3 project.
2. Configure using the MCC.
   a. Configure the UART pin: Launch the pin manager, go to the Pin Setting window of the MCC and configure the UART Transmit pin.
   b. Follow these steps to configure the UART PLIB:
      i. Add the UART6 Peripheral Library from the Device Resources window.
      ii. Click on the added UART6 PLIB box on the project graph to see its configuration options in the window on the right.
      iii. Keep the Interrupt mode enabled and change the baud rate to 115,200 as shown in the following figure:

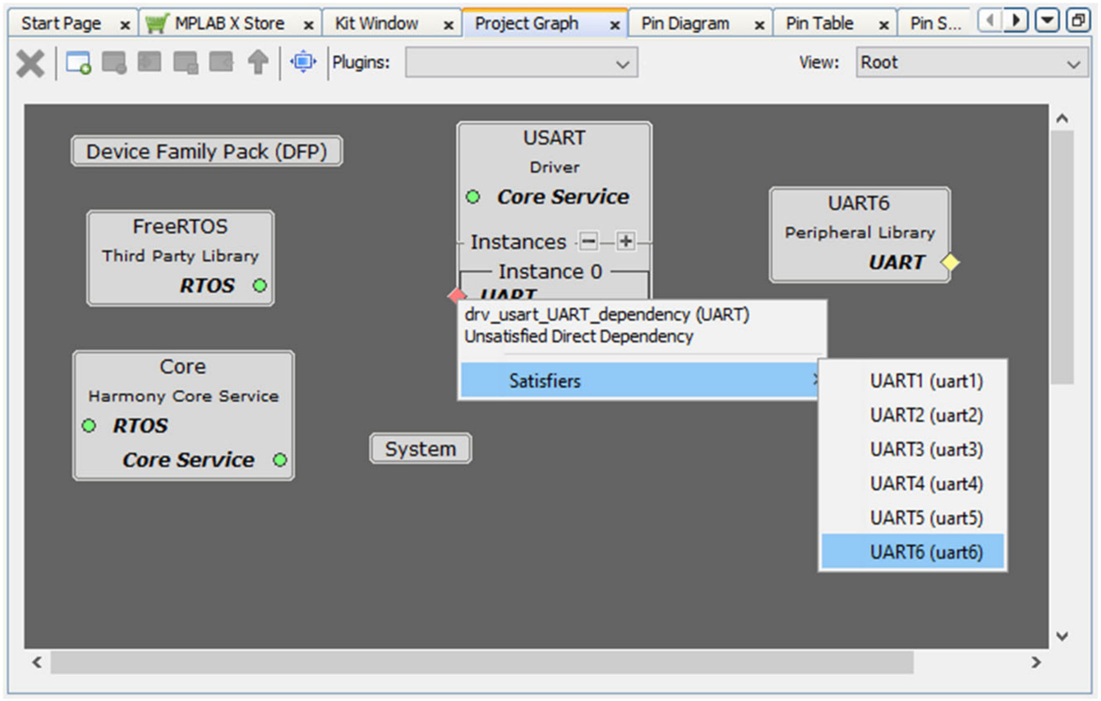
Figure 4-8. Configure UART PLIB



UART6
- Operating Mode: Non-blocking mode
- Stop Selection bit: 1 Stop bit
- Parity and Data Selection bits: 8-bit data, no parity
- High Baud Rate Enable bit: High-Speed mode 4x baud clock enabled
- UARTx Enable bits: UxTX and UxRX pins are enabled and used; UxCTS and UxRTS/UxBCLK pins are controlled by corresponding bits in the PORTx register
- Clock Frequency: 100,000,000
- Baud Rate: 115,200
- *** Baud Error = 0.0061 % ***

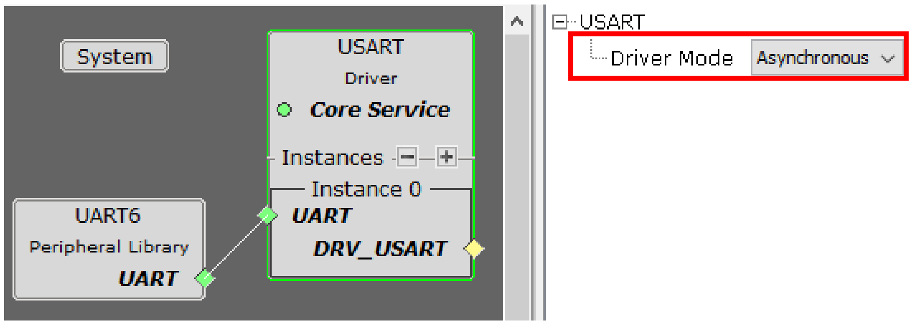
c. Follow these steps to configure the USART Driver:
   i. Add the USART Driver from the Device Resources window.
   ii. Add the Core (HarmonyCore) component when asked in the pop up window.
   iii. Do not add the FreeRTOS component when asked in the pop up window.
   iv. Right-click on the USART driver instance 0 dependency box, and satisfy the dependency with the UART6 PLIB as shown in the following figure:
   Note: In MPLAB Harmony, the driver corresponding to the UART and USART, is the USART driver.

Figure 4-9. Configuring MPLAB Harmony v3 USART Driver 1



   v. Click on the upper side of the USART driver box on the project graph to see the USART driver common configuration options in the window on the right. Let the Driver Mode be Asynchronous as shown in the following figure:

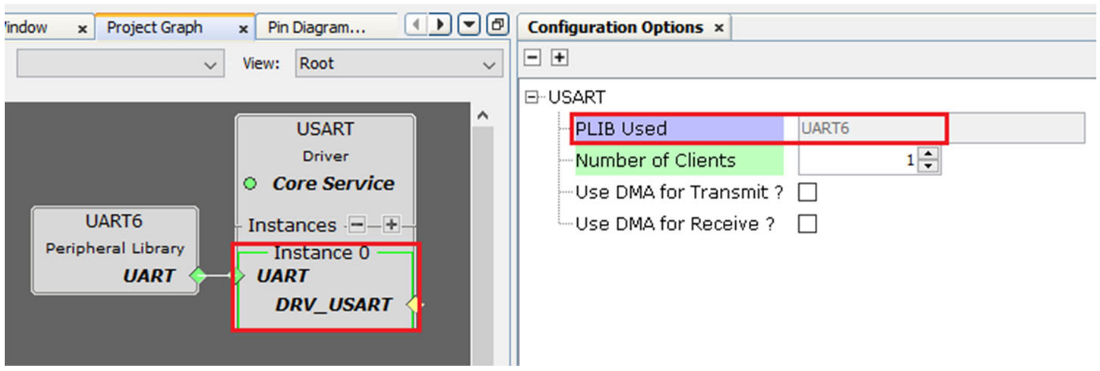
Figure 4-10. Configuring MPLAB Harmony v3 USART Driver 2



USART
Driver Mode: Asynchronous

vi. Click on the lower side of the USART driver box on the project graph to see the USART driver instance 0 configuration options in the window on the right. Leave the configurations set as is. Notice the 'PLIB Used' is automatically configured as UART6. If UART6 is not shown, that means the USART driver is not yet connected with the UART6 PLIB, and this connection needs to be made.

Figure 4-11. Configuring MPLAB Harmony v3 USART Driver 3



USART
PLIB Used: UART6
Number of Clients: 1
Use DMA for Transmit ? ☐
Use DMA for Receive ? ☐

3. Generate the code using the MCC.
4. Update the application:
   a. The UART PLIB and driver initialization code is already generated by the MCC, which is based on the configuration described in the step 2.2 and 2.3.
   b. Update the `app.c` file and the `app.h` file with application logic. The following figure shows the application logic to be developed in the `app.c` file, which has three states:
      i. Open the driver: This state is compulsory for MPLAB Harmony v3 drivers as they are multi-client. Which means even if driver has one client, the application needs to open the driver before using it.
      ii. Queue transfer request: This state adds the transfer request in the queue.
      iii. Check status of the transfer: This state checks the transfer status. As with MPLAB Harmony v2, the status of the transfer can be checked by these methods:
         1. Polling: Application continuously polls for the transfer status using an API. In this example the polling method is used.

2. Callback: Callback can be registered using a dedicated API which will be called once the transfer completes.

```
uint8_t consoleMsg[] = "Hello World\n\r";
void APP_Tasks ( void )
{

    /* Check the application's current state. */
    switch ( appData.state )
    {
        /* Application's initial state. */
        case APP_STATE_OPEN_DRIVER:
        {
            appData.myUSARTHandle = DRV_USART_Open(DRV_USART_INDEX_0, DRV_IO_INTENT_READWRITE|
DRV_IO_INTENT_NONBLOCKING);
            if (appData.myUSARTHandle != DRV_HANDLE_INVALID)
            {
                appData.state = APP_STATE_ADD_REQUEST;
            }
            break;
        }
        case APP_STATE_ADD_REQUEST:
        {
            DRV_USART_WriteBufferAdd(appData.myUSARTHandle, &consoleMsg[0], sizeof(consoleMsg),
&appData.bufferHandle);
            appData.state = APP_STATE_STATUS_CHECK;
            break;
        }
        case APP_STATE_STATUS_CHECK:
        {
            if (DRV_USART_BUFFER_EVENT_COMPLETE & DRV_USART_BufferStatusGet(appData.bufferHandle))
            {
                // Data has been transmitted, go to next state
                appData.state = APP_STATE_COMPLETE;
            }
            break;
        }
```

4.6.1 Comparison Between MPLAB Harmony v2 and MPLAB Harmony v3 Driver Examples

The MCC configuration items are similar in MPLAB Harmony v2 and MPLAB Harmony v3. MPLAB Harmony v3 requires configuration in two parts (as shown in steps 2.2 and 2.3), but the same configuration items are configured in MPLAB Harmony v2 in one step (2.2).

MPLAB Harmony v2 and MPLAB Harmony v3 have the similar driver usage model and application logic except the API changes in MPLAB Harmony v3. The API changes are highlighted in the previous code example.

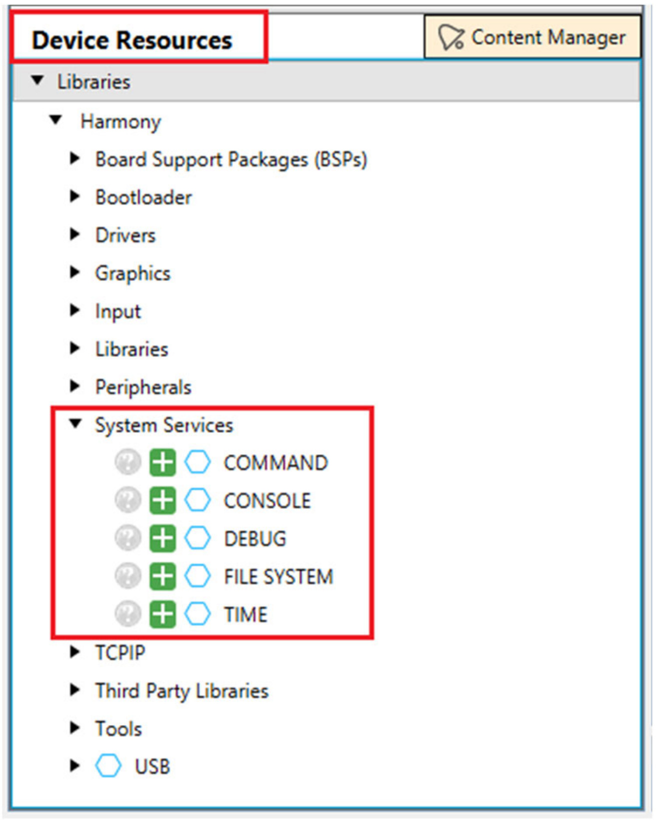
5. System Services

MPLAB Harmony provides System Service libraries to support common functionality and manage resources that are shared by multiple drivers, libraries, and other modules. MPLAB Harmony v3 System Services are similar to MPLAB Harmony v2 System Services.

5.1 How to Start Using MPLAB Harmony v3 System Services

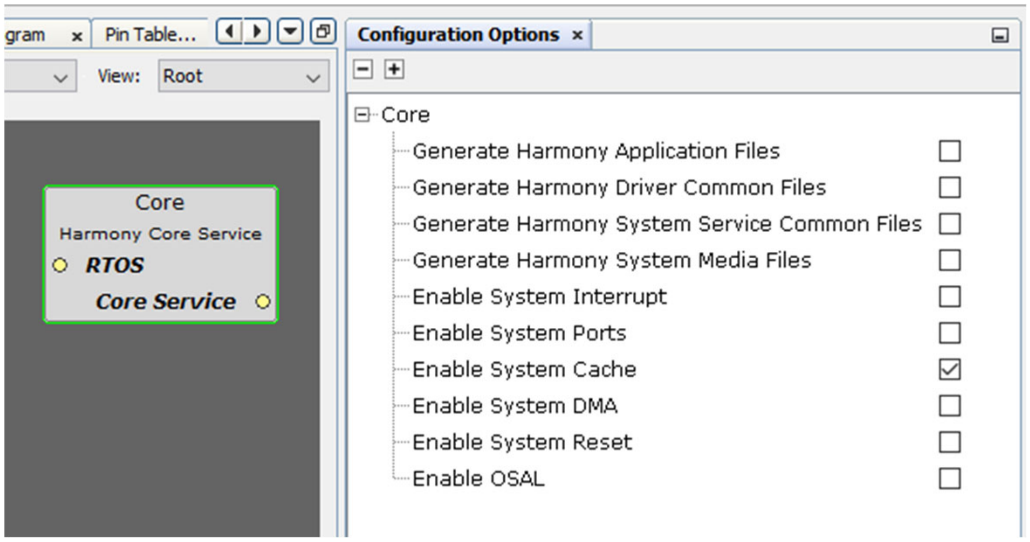
MPLAB Harmony v3 system services are also part of the Harmony Core Service and they can be used by downloading (cloning) the `core` repository from the MPLAB Harmony GitHub as described in the Tools and Installation section. As described in the MCC section above, all the MPLAB Harmony v3 resources, including system services, are listed in the Device Resources window as shown in the following figure. The user can double-click on them to place them on the project graph to configure and use them.

Figure 5-1. Device Resources - System Services



A few System services, commonly used by drivers, middleware libraries and applications (Ports system service, Interrupt system service, and so on) can be configured from the configuration tree of the `core` component (Harmony Core Service) as shown in the following figure. Any MPLAB Harmony v3 resources which need these system services, will select these options automatically. The user needs to select them manually only if they are not selected by default, and the application needs to use them.

Figure 5-2. Common System Services



- Core
  - Generate Harmony Application Files ☐
  - Generate Harmony Driver Common Files ☐
  - Generate Harmony System Service Common Files ☐
  - Generate Harmony System Media Files ☐
  - Enable System Interrupt ☐
  - Enable System Ports ☐
  - Enable System Cache ☑
  - Enable System DMA ☐
  - Enable System Reset ☐
  - Enable OSAL ☐

6. Middleware Libraries

MPLAB Harmony v3 has the same middleware libraries as in MPLAB Harmony v2. The implementation of the middleware is updated to use the latest drivers and system services, but the APIs will remain the same. To get the middleware listed in the MPLAB Harmony v3 Device Resources list, the user must download (clone) the corresponding middleware repository from GitHub as explained in the Tools and Installation section.

7. Real Time Operating System (RTOS) Support

MPLAB Harmony v3 drivers, system services, and middleware, same as in MPLAB Harmony v2, support multiple third party RTOS through the Operating System Abstraction Layer (OSAL). The OSAL provides a consistent interface to allow MPLAB Harmony compliant libraries to take advantage of the operating system constructs when running in an OS environment or when operating without one. The OSAL layer in MPLAB Harmony v3 is same as in MPLAB Harmony v2.

7.1 How to Start Using RTOS in MPLAB Harmony v3

To use any RTOS in MPLAB Harmony v3, two repositories (for each RTOS) are required:

- **Harmony Configuration Repository:** This repository will have the MPLAB Code Configuration files and MPLAB Harmony applications of the RTOS. It is provided by Microchip and can be downloaded from GitHub as explained in the Tools and Installation section.
- **RTOS Source Code Repository:** This repository will have the source code for the RTOS. It must be obtained from a corresponding third-party vendor.

The following table provides the list of currently supported RTOS and the required repositories for use in MPLAB Harmony v3:

Table 7-1. Supported RTOS and Corresponding Repositories

RTOS Name	MPLAB Harmony Configuration Repository	RTOS Source Code Repository
FreeRTOS	core	FreeRTOS
Micrium ucos3	micrium_ucos3	Need to be obtained from the vendor
Thread-X	threadx	Need to be obtained from the vendor

8. Porting MPLAB Harmony v2 Application to MPLAB Harmony v3

An MPLAB Harmony v2 application which needs to be developed on MPLAB Harmony v3, might be using multiple components, such as a PLIB, driver, and middleware. The following is the summary of components for developing an application in MPLAB Harmony v3:

8.1 PLIB

If an application uses the MPLAB Harmony v2 PLIBs, it can be developed using the MPLAB Harmony v3 PLIBs, but it may not be straightforward as the MPLAB Harmony v3 PLIBs are different as compared to the MPLAB Harmony v2 PLIBs. For more information, refer to Application Example Using MPLAB Harmony v2 PLIBS, and Application Example Using MPLAB Harmony v3 PLIBS.

8.2 Static Driver

If an MPLAB Harmony v2 application uses the static driver, it can be developed on MPLAB Harmony v3 in two ways:

1. Instead of the MPLAB Harmony v2 static driver, the MPLAB Harmony v3 PLIB can be used. Application Example Using MPLAB Harmony v3 PLIBS shows an example.
2. Instead of the MPLAB Harmony v2 static driver, the MPLAB Harmony v3 dynamic driver can be used. Application Example Using MPLAB Harmony v3 Driver shows an example.

8.3 Dynamic Driver

If an MPLAB Harmony v2 application uses the dynamic driver, it can be developed on MPLAB Harmony v3 by using the MPLAB Harmony v3 dynamic driver. Application Example Using MPLAB Harmony 3 Driver shows an example.

8.4 System Services

If an MPLAB Harmony v2 application uses the system services, it can be developed on MPLAB Harmony v3 by using the MPLAB Harmony v3 system services. There is no example shown for this as it should be a straightforward change. Refer to System Services for details on the MPLAB Harmony v3 system services.

8.5 Middleware

MPLAB Harmony v3 middleware libraries are same as those in MPLAB Harmony v2. There should not be any API level change needed to port a middleware-based application. However, the MCC configuration options and style has changed in MPLAB Harmony v3, so care must be taken. The following migration documents can be referred to for developing MPLAB Harmony v3 middleware-based applications:

USB:

github.com/Microchip-MPLAB-Harmony/usb

TCP/IP:

github.com/Microchip-MPLAB-Harmony/net

Graphics:

github.com/Microchip-MPLAB-Harmony/gfx

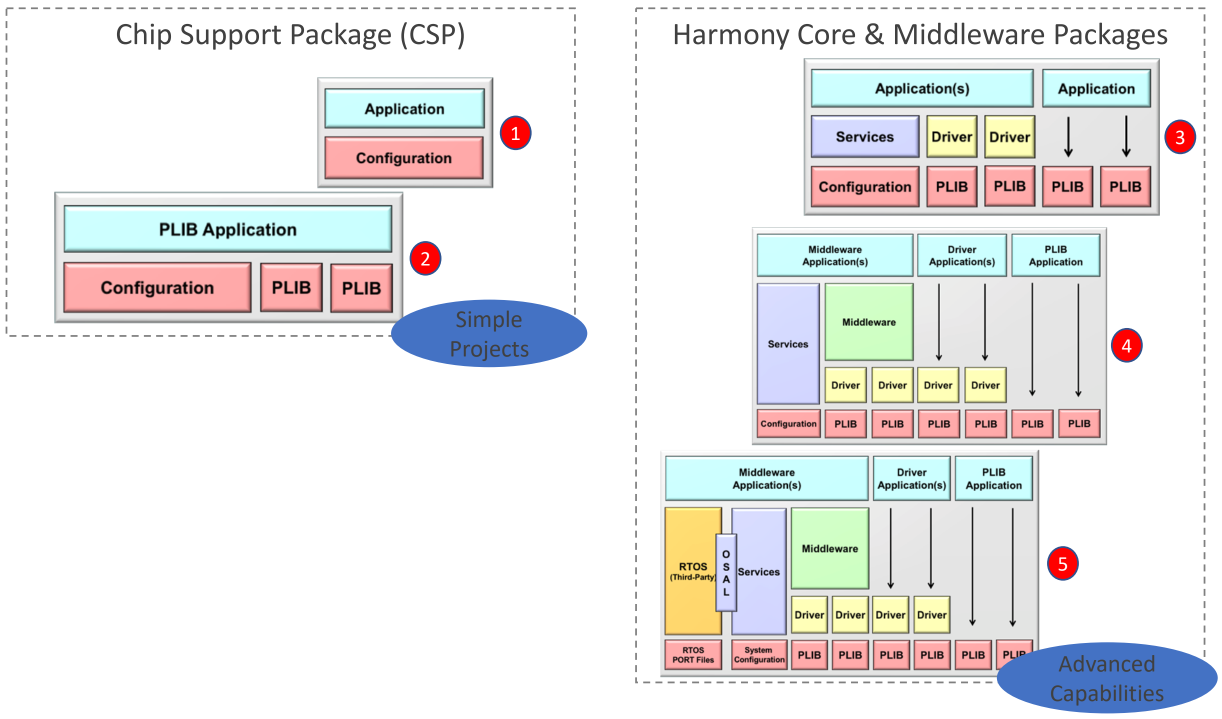
9. MPLAB Harmony Development Models

MPLAB Harmony architecture allows the implementation of a variety of applications from small real-time applications to larger feature rich applications. These applications can be developed using various MPLAB Harmony components (modules) as described in the previous sections. The following development models are derived based on the MPLAB Harmony components used in the application development:

- Simple device configuration and initialization using the MCC
- Peripheral library-based application
- Powerful, conflict-free drivers-based application
- Application requiring MPLAB Harmony middleware
- RTOS-based application for optimum Central Processing Unit (CPU) utilization

The following figure shows the MPLAB Harmony components used in different development models. The first two models are used for simple applications and they only need the `csp` repository to be downloaded (cloned). The last three models are for advanced applications which need `core` and other repositories.

Figure 9-1. MPLAB Harmony v3 Development Models



All five models are supported by both MPLAB Harmony v2 and MPLAB Harmony v3. However, using the 2nd model for application development is easier in MPLAB Harmony v3 compared to MPLAB Harmony v2.

10. Conclusion

MPLAB Harmony v3 provides a modular download and updates through GitHub for better installation and configuration management. It enhances the way MPLAB Harmony modules can be configured using the Project Graph window of the MPLAB Code Configurator (MCC). MPLAB Harmony v3 provides ease of use, and optimized peripheral libraries to develop applications quickly. Applications which use MPLAB Harmony v2 drivers, system services, and middleware libraries can be ported to MPLAB Harmony v3 with few code changes.

11. References

- For additional information on MPLAB Harmony v3, refer to the Microchip web site:
  www.microchip.com/mplab/mplab-harmony
  microchipdeveloper.com/harmony3:start
- How to Setup the MPLAB Harmony v3 Software Development Framework:
  ww1.microchip.com/downloads/en/DeviceDoc/How_to_Setup_MPLAB_%20Harmony_v3_Software_Development_Framework_DS90003232C.pdf
- Detailed documentation on various MPLAB Harmony 3 resources can be found in the documentation folder of the corresponding repository
- For additional information about 32-bit Microcontroller Collaterals and Solutions, refer to:
  DS70005534: 32-bit Microcontroller Collateral and Solutions Reference Guide
- For other relevant information, refer to the Microchip web site
  www.microchip.com/

12. Revision History

Revision B - 12/2024

The following updates were performed for this revision:

- Throughout the document references to the MHC were converted to MCC to reflect a tool upgrade
- The following sections were updated with new links, images and minor content additions to reflect the tool upgrade:
  – Tools and Installation
  – MPLAB Code Configurator GUI
  – Device Resources
  – Project Resources
  – Capability
  – Configuration Options
  – Output
  – Generate
  – MCC Utilities or Plugins
  – Pin Configuration
  – ADC Configuration
  – DMA Configuration
  – Enhanced Vectored Interrupt Controller (EVIC) Configuration
  – How to Start Using Harmony v3 PLIBs
  – Understanding MPLAB Harmony v3 PLIB Generated Code
  – MPLAB Harmony v2 and MPLAB Harmony v3 PLIBs Differences
  – Application Example Using MPLAB Harmony v3 PLIB in Blocking Mode
  – Application Example Using Status Polling
  – How to Start Using MPLAB Harmony v3 Drivers
  – Multiple Clients Support
  – Application Example Using MPLAB Harmony v3 Driver
  – How to Start Using MPLAB Harmony v3 System Services
  – Middleware Libraries
  – How to Start Using RTOS in MPLAB Harmony v3
  – Middleware
  – Conclusion
  – References
- The following new sections were added to this document:
  – Event Configurator
  – Arm TrustZone for Armv8-M
  – TCP IP Configuration

Revision A - 02/2020

This is the initial release of the document.

Microchip Information

Trademarks

ISBN: 979-8-3371-0209-2

Legal Notice

Microchip Devices Code Protection Feature