emo

OBD Tool Development and User Guide

August 2023

The following document summarises the initiation and development of a general On-Board diagnostic tool over the course of Feb 2023 to August 2023 for all the battery packs and energy storage systems developed by EMO energy Pvt. ltd.

INTRODUCTION

The On-Board diagnostic tool is the mediator between the graphical user interface (GUI) and the battery management system (BMS) allowing data to be communicated seamlessly by converting data from BMS readable format to human readable format and vice versa.

The primary functions of the OBD tool are as follows

1. Communicate via controller area network (CAN) protocol to the battery and request the BMS to send necessary data.
2. Communicate via the Universal serial asynchronous RX-TX protocol to the GUI and deliver the data in human readable from.

At present the following functions have been implemented:

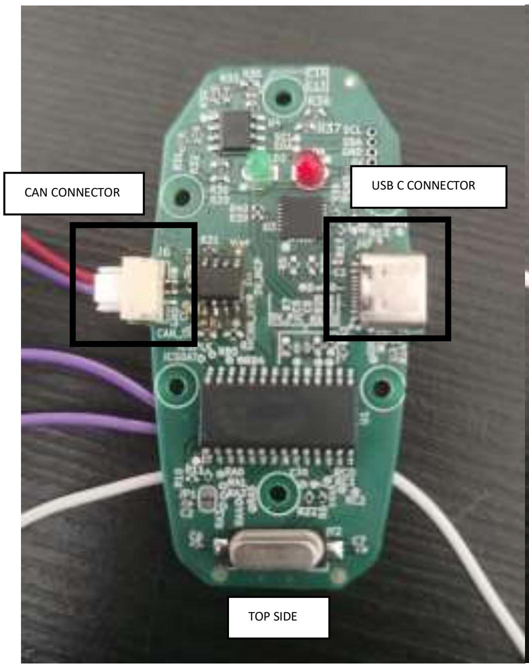
1. Request real time data containing cell voltages, pack voltage, pack current, temperatures, error and faults from the BMS and transmit to the USB via UART in the form of strings appended with certain identifiers.
2. Request the configuration parameters from the BMS that was already uploaded upon getting a command request from the GUI
3. Push the new configuration parameters to the BMS on command request from the GUI.
4. Request SD card data to be downloaded on request from the GUI.
5. Change baud rates of UART and CAN upon command request from GUI
6. Flow control between OBD and BMS whenever there is an event of exchanging data.

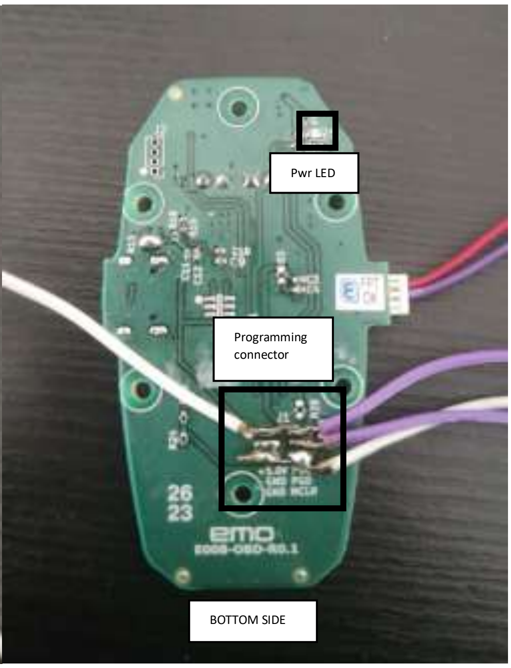
CHAPTER 1

HARDWARE DESIGN

1. The PCB for the OBD tool is designed in KICAD
2. PIC18F27Q84 is used as the MCU on the PCB (Note: The available curiosity nano development board has PIC18F57Q84 as its MCU)
3. CP2102-GMR QFN29 is used for UART to USB converter, MCP2551 is used as CAN controller.
4. USB C is used for connecting it to the device running GUI, and a 3 pin JST connector is used for connecting to the BMS.
5. A 10MHz crystal is used for as external clock for the PIC.

Further schematics and layout can be found in the KICAD project.





CHAPTER 2

FIRMWARE DEVELOPMENT

The OBD is made sure to be useable only when the GUI is opened on the host computer as a result a simple state machine consisting of two states is implemented.

- "GUI_connected" state
- "GUI_disconnected" state

1. GUI_disconnected state

This state is when the OBD is powered and the GUI application is not opened on the host computer.

- The OBD sends command "**AT+TEST\r\n"** continuously on the UART until it receives command "**AT+TEST+OK\r\n"**
- During disconnected state any normal operation or sending commands through foreign serial terminal is prohibited, unless it is a "**AT+TEST=OK\r\n"** command which is a breach of security.
- Upon reading "**AT+TEST=OK\r\n**" the OBD switches to the next state.

2. GUI_connected state

This state is when the OBD is powered and the GUI application is running on the host computer

- There are 10 events that the OBD might go into during this state
- *Default_event*: To continuously request BMS the data containing pack voltage, pack current, SOC and error codes followed by 20 cell voltages and 6 thermistor values. The data is appended to three strings. First containing error codes, pack voltage, pack current and SOC with "VCS" as identifiers, Second containing the 20 cell voltages with "#*" as identifiers, third containing 6 thermistor values with "^C" as identifier. All three strings are sent over UART continuously as comma separated values.
  CAN ID for request: **0x1806F4F4**
  CAN DATA for request: **{0x04, 0x0D, 0x04, 0x63}**
- *BMS_baud_rate_change_event & GUI_baud_rate_change_event:* Request to change the baud rates/ bit rates of the CAN for the BMS or the UART for the GUI.
  The request to change the baud rate is initiated by the GUI through the following commands
  For UART:
  1. "**AT+BAUD=1\r\n"**: Changes UART baud rate to 9600 (default) when OBD is powered or reset.
  2. **"AT+BAUD=2\r\n":** Changes UART baud rate to 57600
  3. **"AT+BAUD=3":** Changes UART baud rate to 115200

  For CAN:

  1. **"AT+CANB=1\r\n":** Changes CAN bit rate to 125Kbps
  2. **"AT+CANB=2\r\n":** Changes CAN bit rate to 250Kbps
  3. **"AT+CANB=3\r\n":** Changes CAN bit rate to 500Kbps

Note: The default CAN bit rate when OBD is powered will go to 250Kbps, however when changed by the GUI the OBD will register the new bit rate to its EEPROM. Upon changing the bit rate of the OBD a message is also sent to the BMS on the older baud rate to change its bit rate to the new bit rate that OBD has been asked to set to.

CAN ID for 125Kbps: **0x1806F4F4; CAN DATA: {0x04, 0x0D, 0x04, 0x66}**

CAN ID for 250Kbps: **0x1806F4F4; CAN DATA: {0x04, 0x0D, 0x04, 0x65}**

CAN ID for 500Kbps: **0x1806F4F4; CAN DATA: {0x04, 0x0D, 0x04, 0x64}**

- *sd_data_download_event:* Request to download the SD card data from given set of from date and to date. The request to download the SD card data from the BMS is initiated by the GUI with the command: **"AT+DATA=1,DD,MM,YY,DD,MM,YY\r\n"** (from date followed by To date) or **"AT+DATA=2\r\n"** if the entire SD card data has to be downloaded.
  On receiving the command, the dates are segregated and stored in an 8byte array, the first two bytes will contain the character 'S' and 'D' the remaining 6 bytes will contain the DD MM YY DD MM YY.
  CAN ID for DATE download: **0x1806F4F4;** CAN DATA: **{'S', 'D', DD,MM,YY,DD,MM,YY}**
  CAN ID for FULL download: **0x1806F4F4**; CAN DATA: **{'S', 'D', 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00}**

  Flow control for exchanging SD card data:

  The flow control is written in such a way that after exchanging 48bytes (1 packet) between the OBD and the BMS, it is sent to the GUI with "SD" as identifiers with every byte as comma separated values.

  1 packet (48bytes) consists of 7 CAN frames/messages with ID: **0x0121FFF4**
  Frame 1: {0x10,D1,D2,D3,D4,D5,D6,D7}
  D1: contains the number of bytes in a packet, for examples if first packet has 48bytes D1 would be 48, if last pack has just 46bytes then D1 would be 46 and so on.
  Frame 2: {0x20,D1,D2,D3,D4,D5,D6,D7}
  Frame 3: {0x21,D1,D2,D3,D4,D5,D6,D7}
  Frame 4: {0x22,D1,D2,D3,D4,D5,D6,D7}
  Frame 5: {0x23,D1,D2,D3,D4,D5,D6,D7}
  Frame 6: {0x24,D1,D2,D3,D4,D5,D6,D7}
  Frame 7: {0x25,D1,D2,D3,D4,D5,D6,D7}

  Upon sending the request for downloading SD card data to the BMS, the BMS responds with the Frame 1, then OBD responds back with an acknowledgement message with ID **0x1806F4F6** and data **{0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00}** it then starts its timer and waits for the next frame, if the BMS does not respond within 6sec (set by the timer) OBD transmits a timeout message to both BMS and GUI and goes back to the default event.

BMS timeout message: ID: **0x1806F4F6** data:
{0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00}
GUI timeout message: **"AT+TIME\r\n"**

If the BMS responds within 6 seconds the OBD continues to wait for subsequent frames and after the reception of all the 7 frames (48bytes) (1 packet), the BMS sends a confirmation message on the ID: **0x0110FFF4** with the data **{0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00}.** and the OBD sends the received packet to the GUI as a CSV string with "SD" appended on it. The BMS Upon sending all the packets of the SD card data sends a "complete" message on the ID: **0x0110FFF4** with data: **{0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00}.** And the OBD sends the last packet to the GUI followed by a confirmation message **"AT+DONE\r\n".**

- *BMS_software_rst_event:*

The request for resetting the BMS is initiated by the GUI with the command **"AT+REST=1\r\n"**

Upon receiving the command the OBD forwards the command request to the BMS through the ID:**0x1806F4F5** with the message **{0x00,0xFF,0x01,0x00}**. The BMS if reset sends a confirmation message to the OBD on the ID: **0x0110FFF4** with the message**{0x01,0x00,0x00,0x00,0x00,0x00,0x00}**
The OBD on receiving this command sends a confirmation message to the GUI **"AT+DONE\r\n"** indicating the successful resetting of the BMS.
If the BMS fails to respond with the confirmation message within 6 sec (set by the timer) the OBD sends a timeout message to BMS and GUI and goes back to its default event.
BMS timeout message: ID: **0x1806F4F6** data:
{0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00}
GUI timeout message: **"AT+TIME\r\n"**

- *BMS_upl_config_event:*

The process for uploading the new configuration parameters to the BMS is initiated by the GUI with the command
"AT+CONF=1,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xxxx,xx,xx,xx,xx,xx,xx,xx,xx\r\n"
Where **x** represents digits, currently 15 parameters have been finalised. The parameters that contain 2 bytes of data has to be put in place of 4 digits and the parameters containing 1 byte data has to be put in place of 2 digits.
It is important to note that the parameters have to be sent in the form of 4 or 2 digits as OBD string processing is strictly based on the length of the string.
Since there are 15 parameters in total, 7 parameters with 2 bytes (4 digit) 8 parameters with 1 byte(2digit) there are totally 22 bytes of data to be transmitted to BMS.
The transmission is on the ID: **0x1806F4F5** with frames as follows.

Frame 1: **{0x01, LSB1,MSB1,LSB2,MSB2,LSB3,MSB3,1st two-digit }**

Frame 2: **{0x02, LSB4,MSB4,LSB5,MSB5,LSB6,MSB6,2nd two-digit]}**

Frame 3: **{0x03, LSB7,MSB7, 3rd two-digit,4th two-digit ,5th two-digit ,6th two-digit }**

Frame 4: **{0x04, 7th two-digit, 8th two-digit,0x00,0x00,0x00,0x00,0x00]}**

After each frame transmitted, the OBD waits for a confirmation message from the BMS that comes on the ID: **0x0110FFF4** with the data **{0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00}**, if the confirmation is not received by the OBD within 6 seconds (set by the timer) it simply sends a timeout message to BMS and GUI and goes back to the default event.

BMS timeout message: ID: **0x1806F4F6** data: **{0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00}**

GUI timeout message: **“AT+TIME\r\n”**

After the last frame has been sent and confirmed by the BMS, the OBD sends a confirmation message to the GUI **“AT+DONE\r\n”**

- *BMS_get_config_event:*

The request to get the present configuration parameters from the BMS is initiated by the GUI with the command **“AT+CONF=2\r\n”**

The OBD upon receiving the command sends a request message to the BMS with the ID: **0x1806F4F5** and data: **{0x00,0x01,0x00,0x00}** and waits for the BMS to respond with a CAN message containing the ID:**0x0121FFF5** and the following frames

Frame1: **{0x01,LSB1,MSB1,LSB2,MSB2,LSB3,MSB3,1st two-digit}**

Frame2: **{0x02,LSB4,MSB4,LSB5,MSB5,LSB6,MSB6,2nd two-digit}**

Frame3: **{0x03,LSB7,MSB7,3rd two-digit,4th two-digit,5th two-digit,6th two-digit,7th two-digit}**

Frame4: **{0x04,8th two-digit,0x00,0x00,0x00,0x00,0x00,0x00}**

After every frame is received from the BMS the OBD sends a confirmation on the ID: **0x1806F4F6** with the data: **{0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00}** and waits for the next frames to receive, if the time elapsed in waiting exceeds 6sec the OBD sends a timeout message to BMS and GUI and then goes back to the default event.

After the last frame has been sent by the BMS it sends a “complete” message on the ID: **“0x0110FFF4”** with the data **{0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00}**, After which the OBD sends the parameters to GUI as CSV string with “CON” as identifiers.

- *BMS_bootloader_event:*

Refer the following excel for the detailed CAN messages.

CAN data - Google Sheets

CHAPTER 3

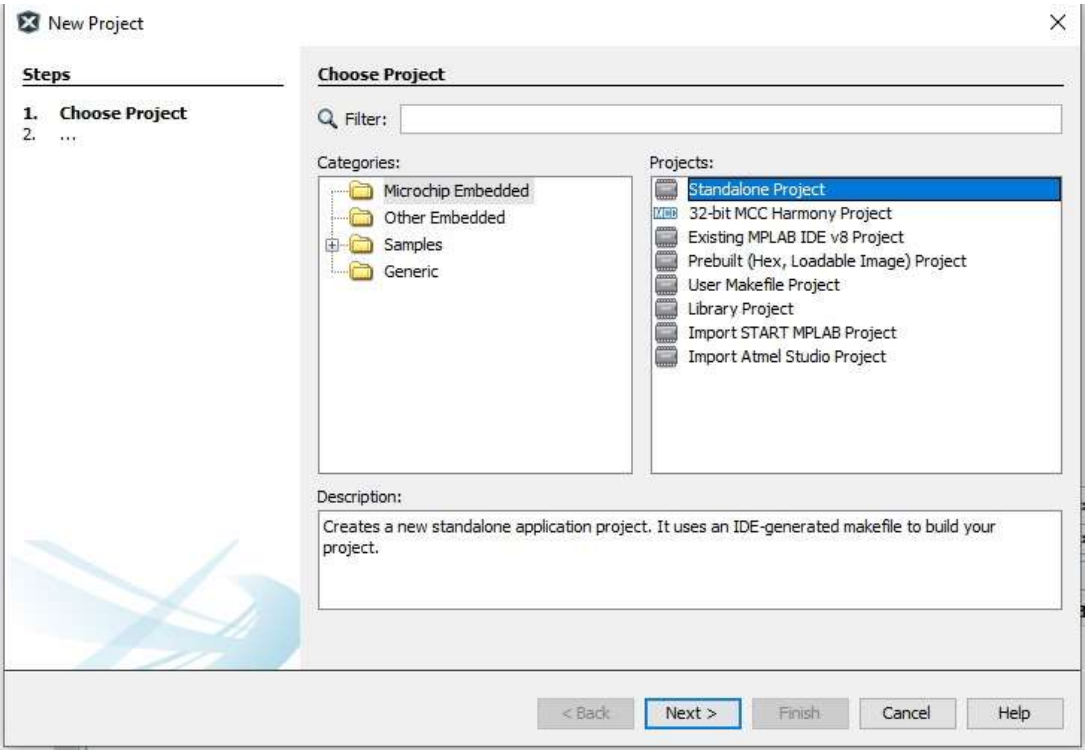
MPLAB CODE CONFIGURATION

The entire firmware is developed in MPLAB X IDE v6.10.

The projects titled **OBD_CAN_UART** is written for curiosity nano board carrying PIC18F57Q84 MCU, and the project titled **OBD_TOO_PCB_FINAL** is written for the E008-PCB carrying PIC18F27Q84 MCU.

The steps to configure projects for the development is as shown:

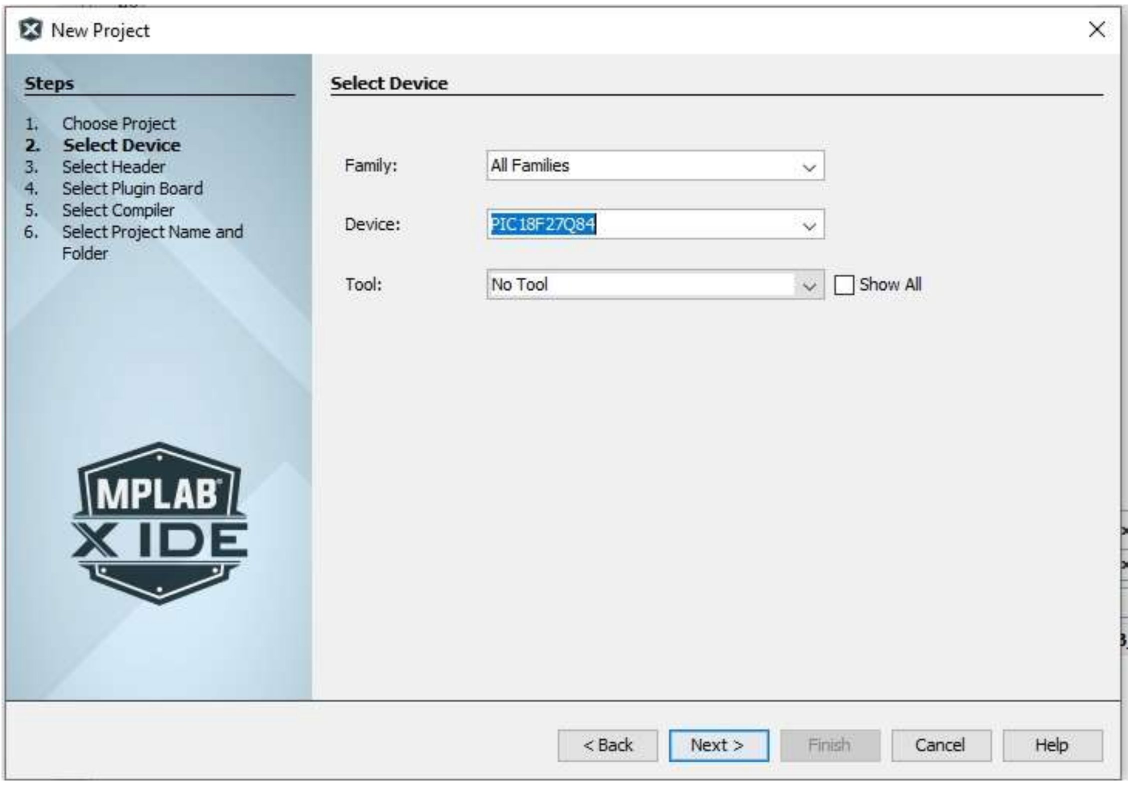
1. Create a new project from the file section, select standalone project and click next, choose the device as PIC18F27Q84 and click next, select xc8 compiler and give the project name and click finish.



Categories: Microchip Embedded

Projects: Standalone Project

Description: Creates a new standalone application project. It uses an IDE-generated makefile to build your project.



New Project

Steps

1. Choose Project
2. **Select Device**
3. Select Header
4. Select Plugin Board
5. Select Compiler
6. Select Project Name and Folder

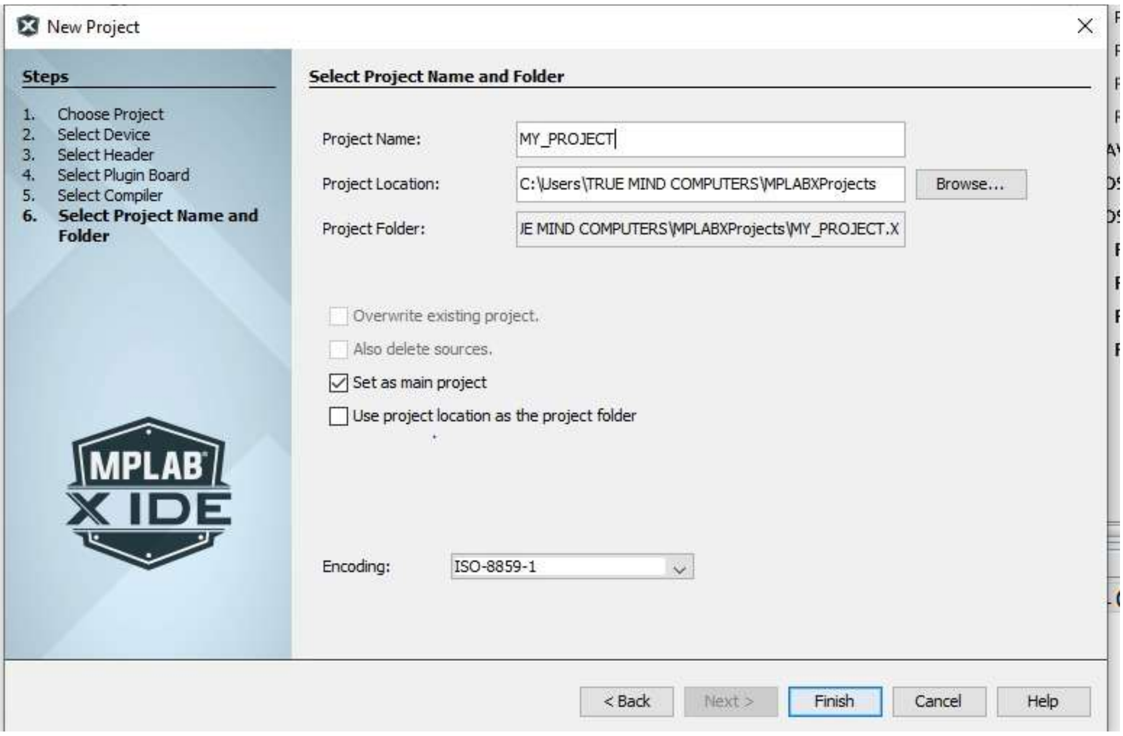
Select Device

Family: All Families

Device: PIC18F27Q84

Tool: No Tool

□ Show All



New Project

Steps

1. Choose Project
2. Select Device
3. Select Header
4. Select Plugin Board
5. Select Compiler
6. **Select Project Name and Folder**

Select Project Name and Folder

Project Name: MY_PROJECT

Project Location: C:\Users\TRUE MIND COMPUTERS\MPLABXProjects

Project Folder: JE MIND COMPUTERS\MPLABXProjects\MY_PROJECT.X

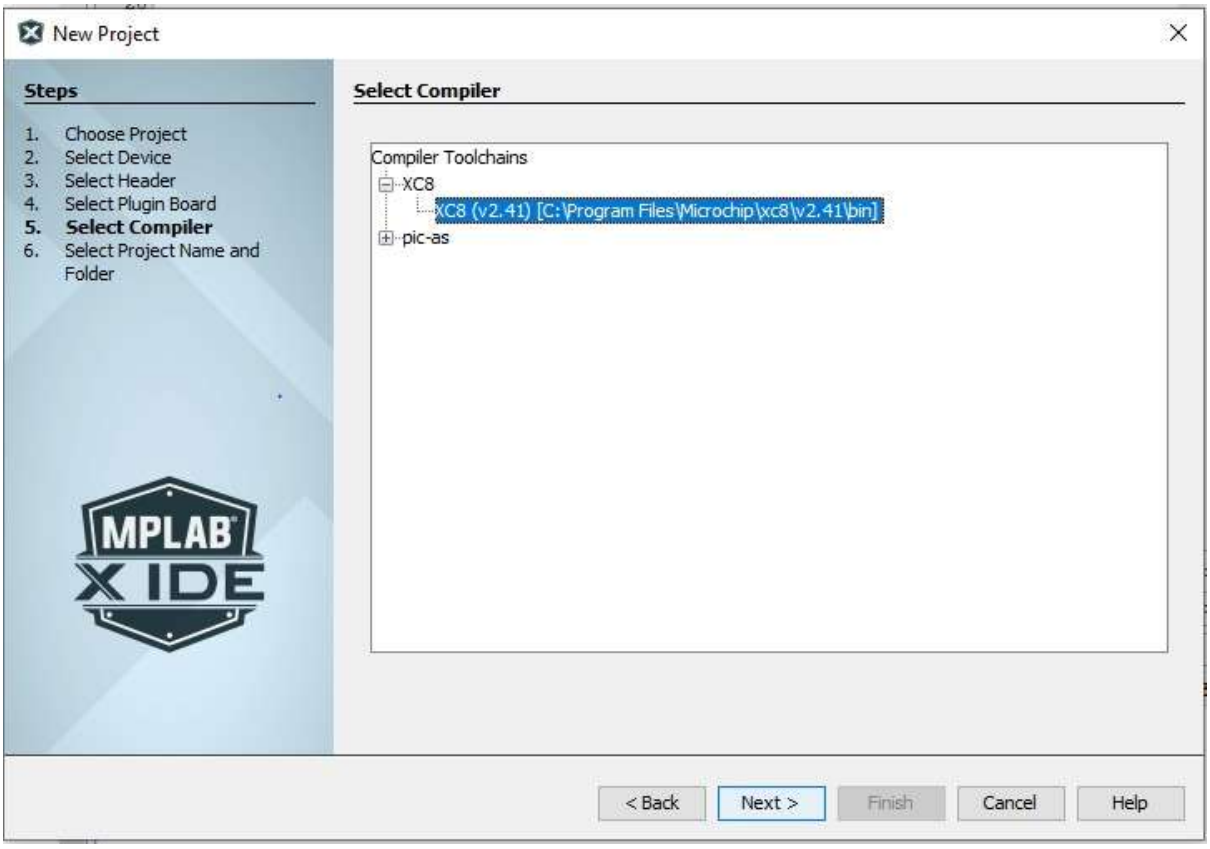
□ Overwrite existing project.

□ Also delete sources.

☑ Set as main project

□ Use project location as the project folder

Encoding: ISO-8859-1



New Project

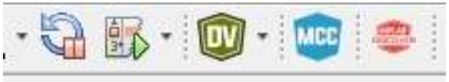
Steps

1. Choose Project
2. Select Device
3. Select Header
4. Select Plugin Board
5. **Select Compiler**
6. Select Project Name and Folder

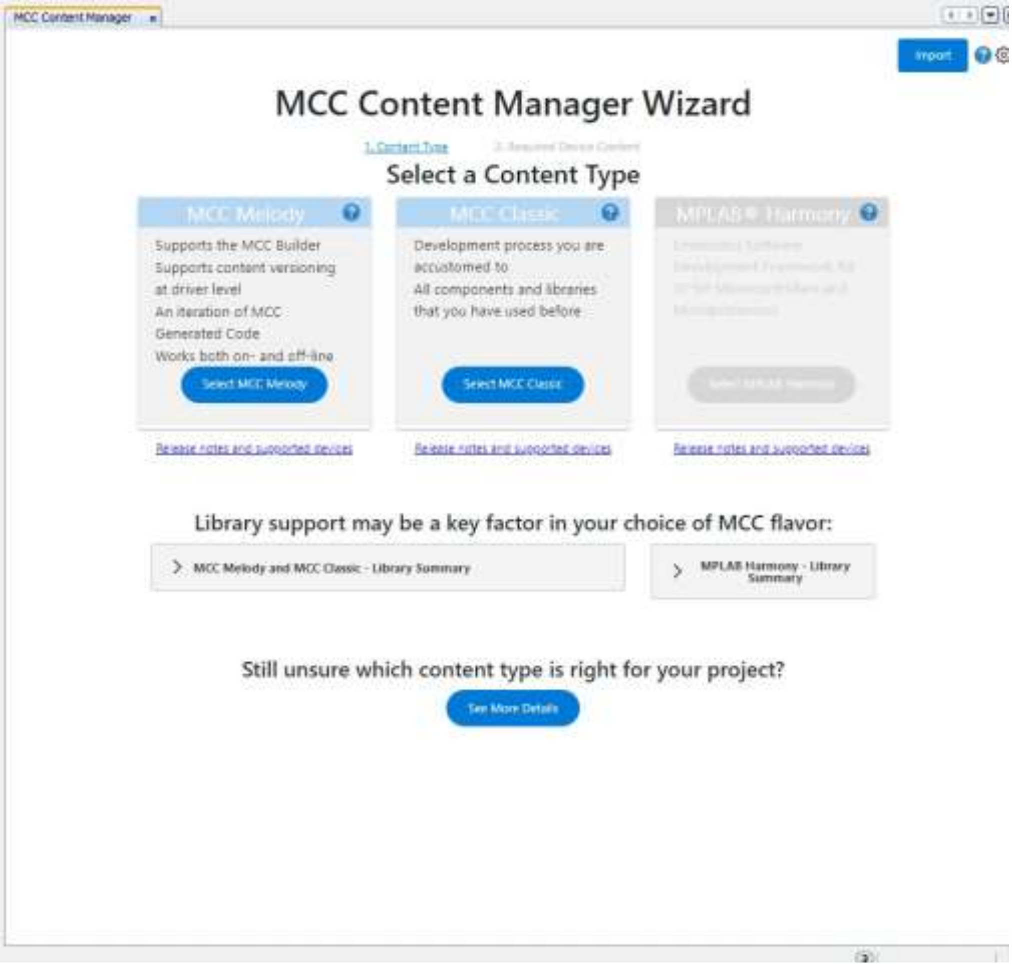
Select Compiler

Compiler Toolchains
- XC8
  - XC8 (v2.41) [C:\Program Files\Microchip\xc8\v2.41\bin]
- pic-as

2. Open the MPLAB code configurator by clicking the blue logo called the MCC from the main page.



3. Select MCC classic.



MCC Content Manager
Import
1. Content Type
2. Required Device Content
Select MCC Melody
Select MCC Classic
Select MPLAB Harmony
Release notes and supported devices
Release notes and supported devices
Release notes and supported devices
MCC Melody and MCC Classic - Library Summary
MPLAB Harmony - Library Summary
See More Details

MCC Content Manager Wizard

Select a Content Type

MCC Melody

Supports the MCC Builder
Supports content versioning at driver level
An iteration of MCC Generated Code
Works both on- and off-line

MCC Classic

Development process you are accustomed to
All components and libraries that you have used before

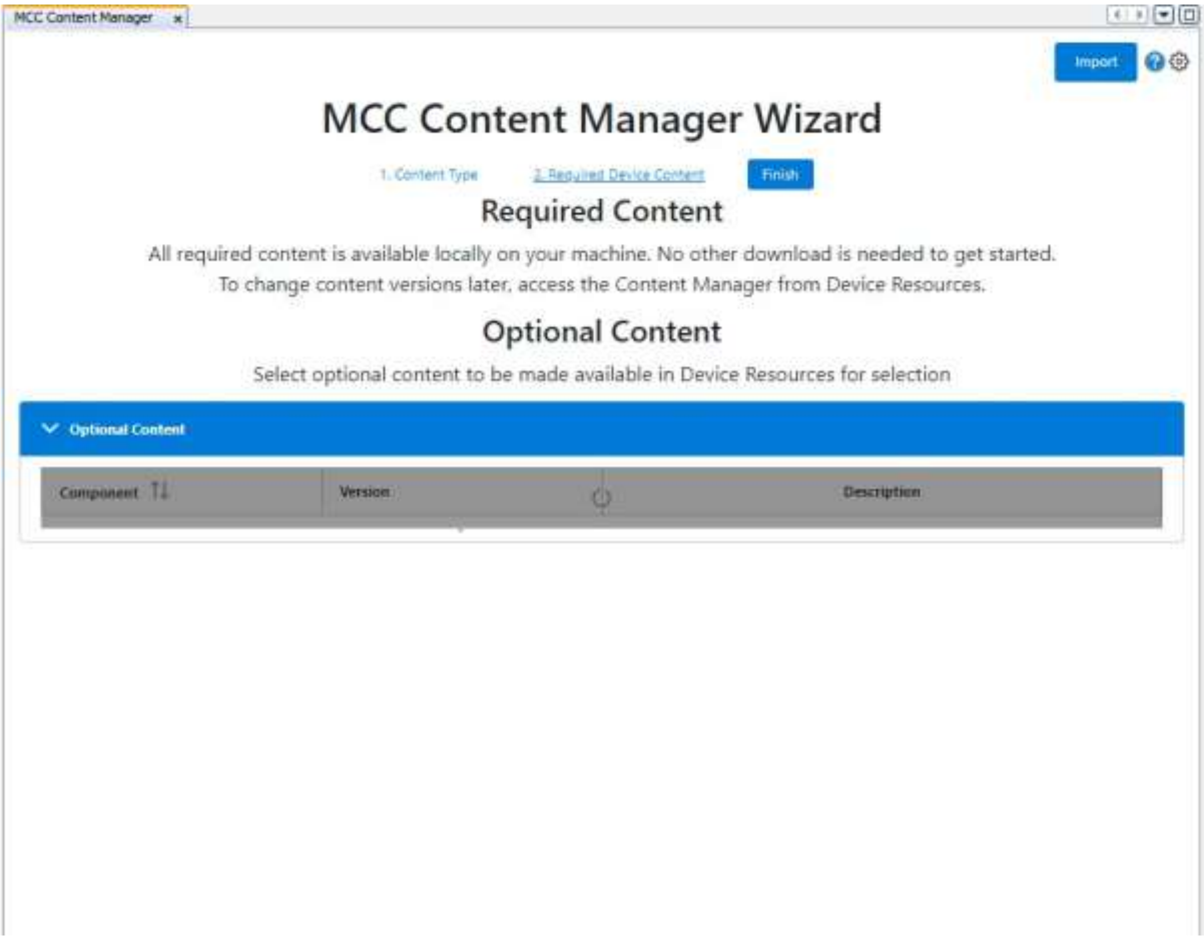
MPLAB® Harmony

Embedded Software Development Framework for 32-bit Microcontrollers and Microprocessors

Library support may be a key factor in your choice of MCC flavor:

Still unsure which content type is right for your project?

4. After downloading necessary modules click on finish



MCC Content Manager Wizard

Required Content

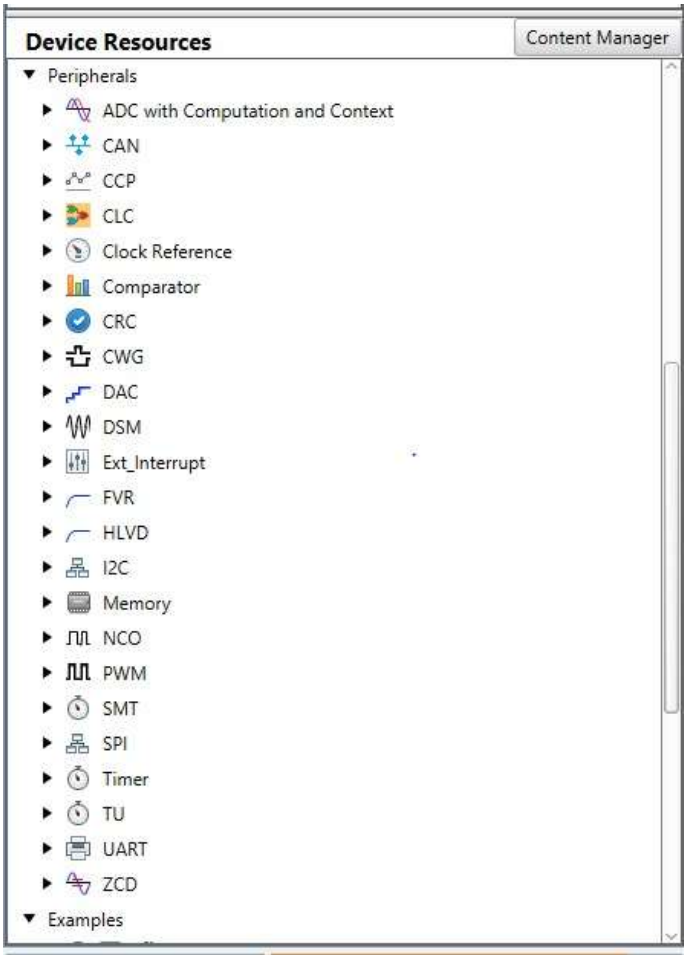
All required content is available locally on your machine. No other download is needed to get started.
To change content versions later, access the Content Manager from Device Resources.

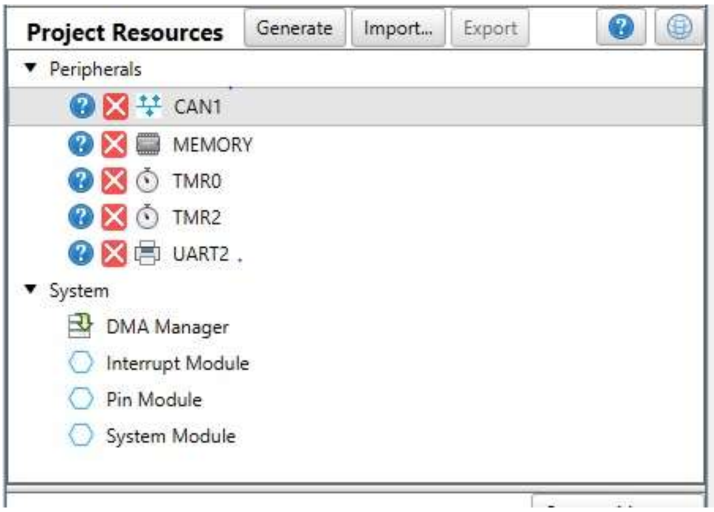
Optional Content

Select optional content to be made available in Device Resources for selection

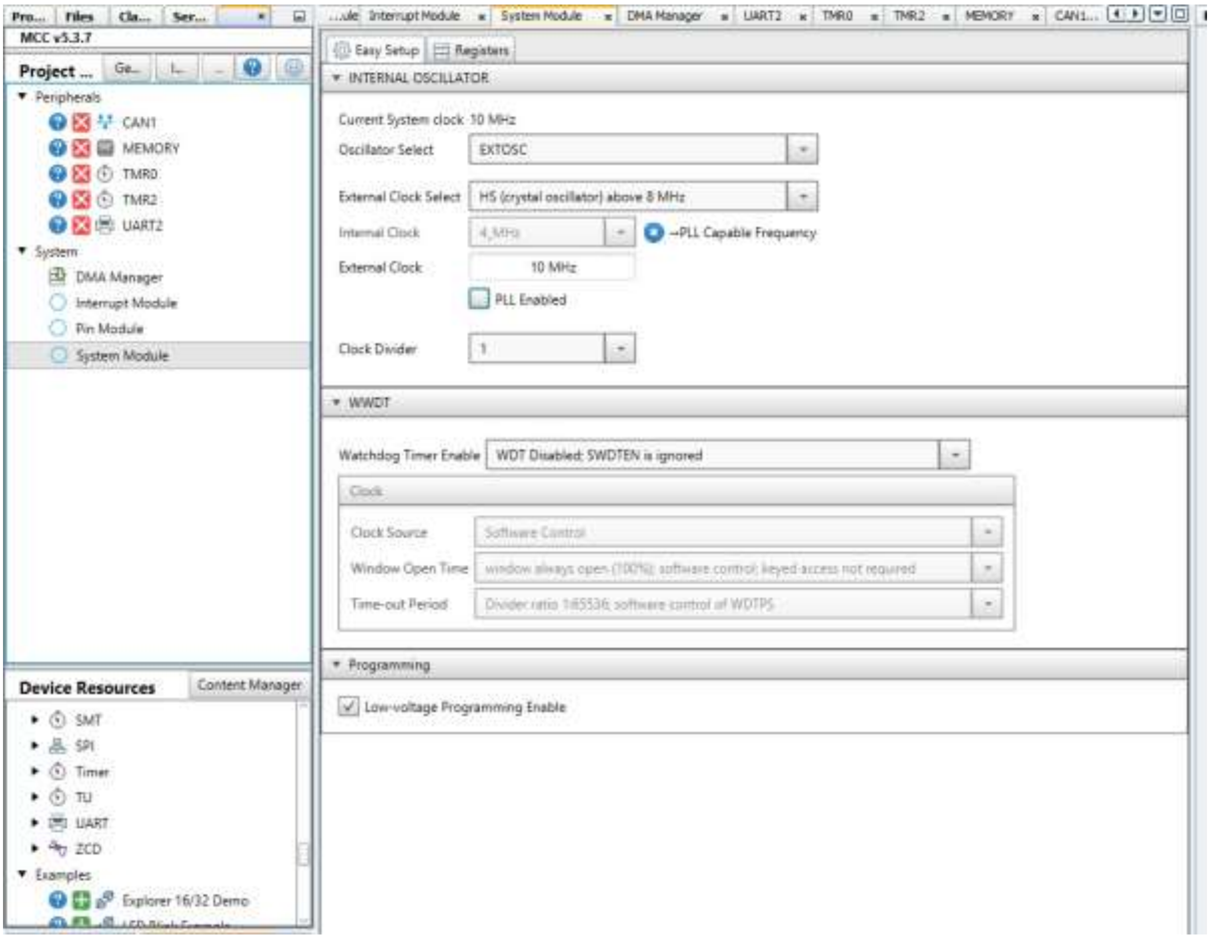
Component	Version	Description

5. In the peripherals section click on plus to add the following peripherals UART2, CAN1, TIMER0, TIMER2, MEMORY





6. In the system module tab configure the following



INTERNAL OSCILLATOR

Current System clock 10 MHz

Oscillator Select: EXTOSC

External Clock Select: HS (crystal oscillator) above 8 MHz

Internal Clock: 4_MHz ● →PLL Capable Frequency

External Clock: 10 MHz

☐ PLL Enabled

Clock Divider: 1

WWDT

Watchdog Timer Enable: WDT Disabled; SWDTEN is ignored

Clock

Clock Source: Software Control

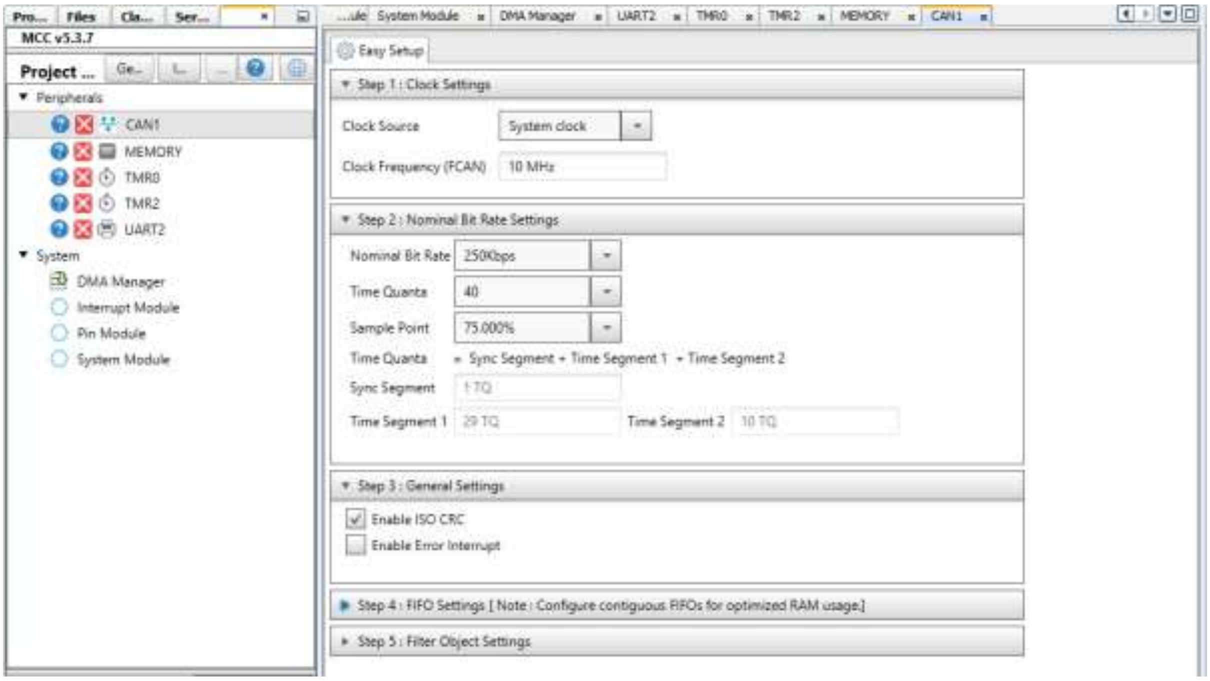
Window Open Time: window always open (100%); software control; keyed access not required

Time-out Period: Divider ratio 1:65536; software control of WDTPS

Programming

☑ Low-voltage Programming Enable

7. In the CAN1 peripheral configure the following



Step 1 : Clock Settings

Clock Source: System clock

Clock Frequency (FCAN): 10 MHz

Step 2 : Nominal Bit Rate Settings

Nominal Bit Rate: 250Kbps

Time Quanta: 40

Sample Point: 75.000%

Time Quanta = Sync Segment + Time Segment 1 + Time Segment 2

Sync Segment: 1 TQ

Time Segment 1: 29 TQ Time Segment 2: 10 TQ

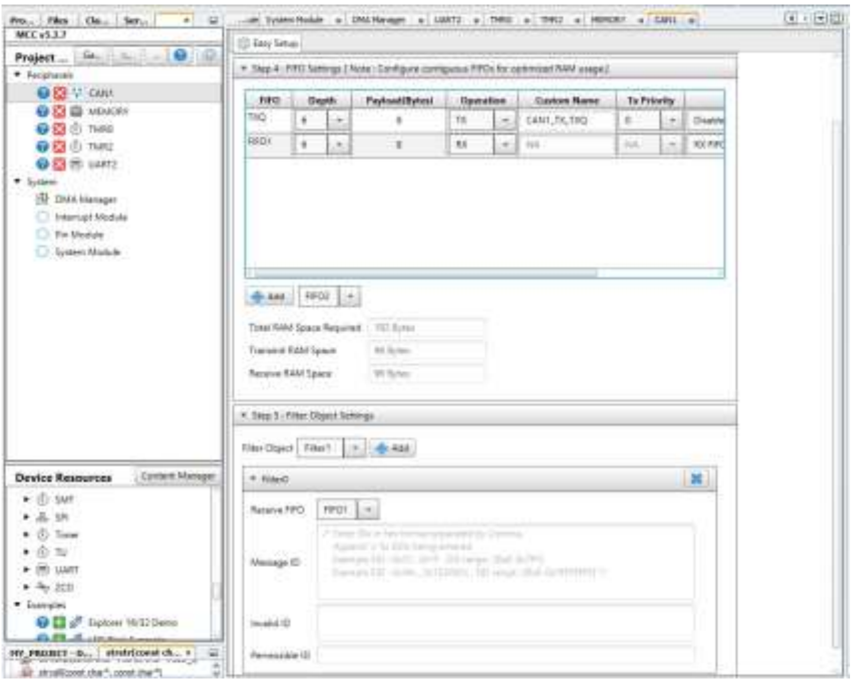
Step 3 : General Settings

☑ Enable ISO CRC

☐ Enable Error Interrupt

Step 4 : FIFO Settings [Note : Configure contiguous FIFOs for optimized RAM usage.]

Step 5 : Filter Object Settings



▼ Step 4 : FIFO Settings [Note : Configure contiguous FIFOs for optimized RAM usage.]

FIFO	Depth	Payload(Bytes)	Operation	Custom Name	Tx Priority	
TXQ	6	8	TX	CAN1_TX_TXQ	0	Disable
FIFO1	6	8	RX	NA	NA	RX FIFO

Add FIFO2

Total RAM Space Required: 192 Bytes

Transmit RAM Space: 96 Bytes

Receive RAM Space: 96 Bytes

▼ Step 5 : Filter Object Settings

Filter Object: Filter1 Add

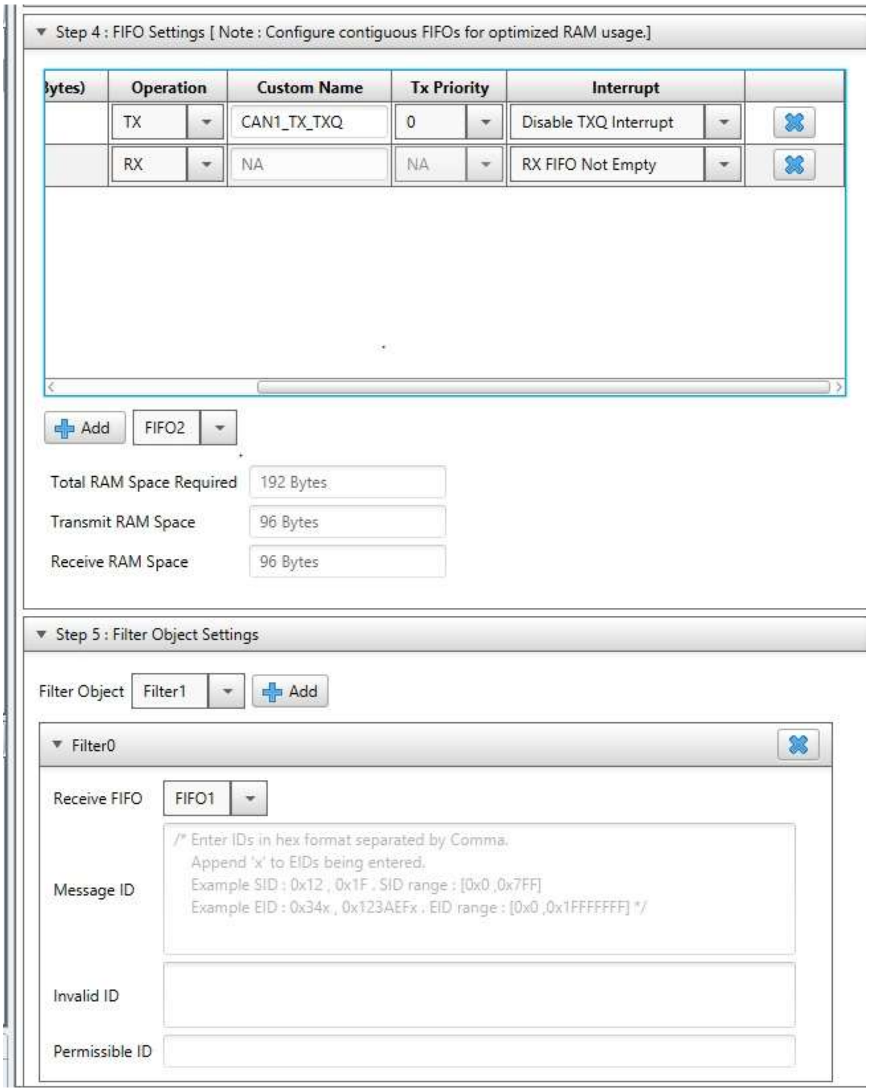
▼ Filter0

Receive FIFO: FIFO1

Message ID

Invalid ID

Permissible ID



▼ Step 4 : FIFO Settings [Note : Configure contiguous FIFOs for optimized RAM usage.]

ytes)	Operation	Custom Name	Tx Priority	Interrupt	
	TX	CAN1_TX_TXQ	0	Disable TXQ Interrupt	✖
	RX	NA	NA	RX FIFO Not Empty	✖

Total RAM Space Required: 192 Bytes

Transmit RAM Space: 96 Bytes

Receive RAM Space: 96 Bytes

▼ Step 5 : Filter Object Settings

Filter Object: Filter1 Add

▼ Filter0

Receive FIFO: FIFO1

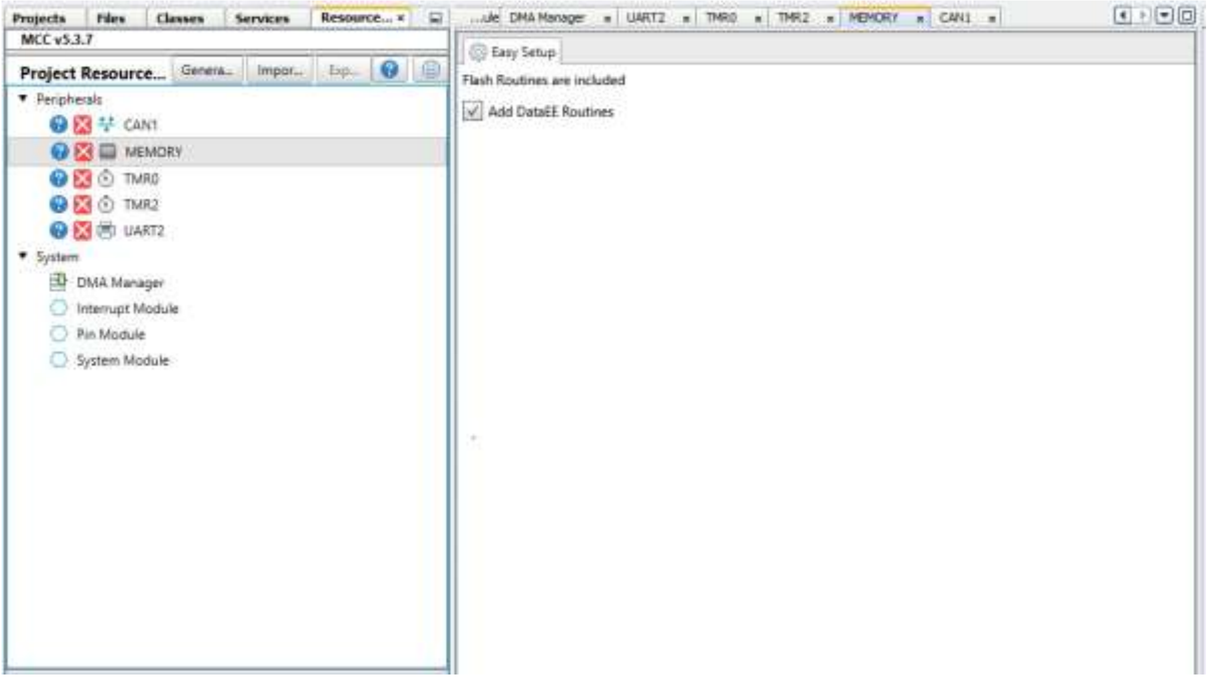
Message ID:

```
/* Enter IDs in hex format separated by Comma.
   Append 'x' to EIDs being entered.
   Example SID : 0x12 , 0x1F . SID range : [0x0 ,0x7FF]
   Example EID : 0x34x , 0x123AEFx . EID range : [0x0 ,0x1FFFFFFF] */
```

Invalid ID:

Permissible ID:

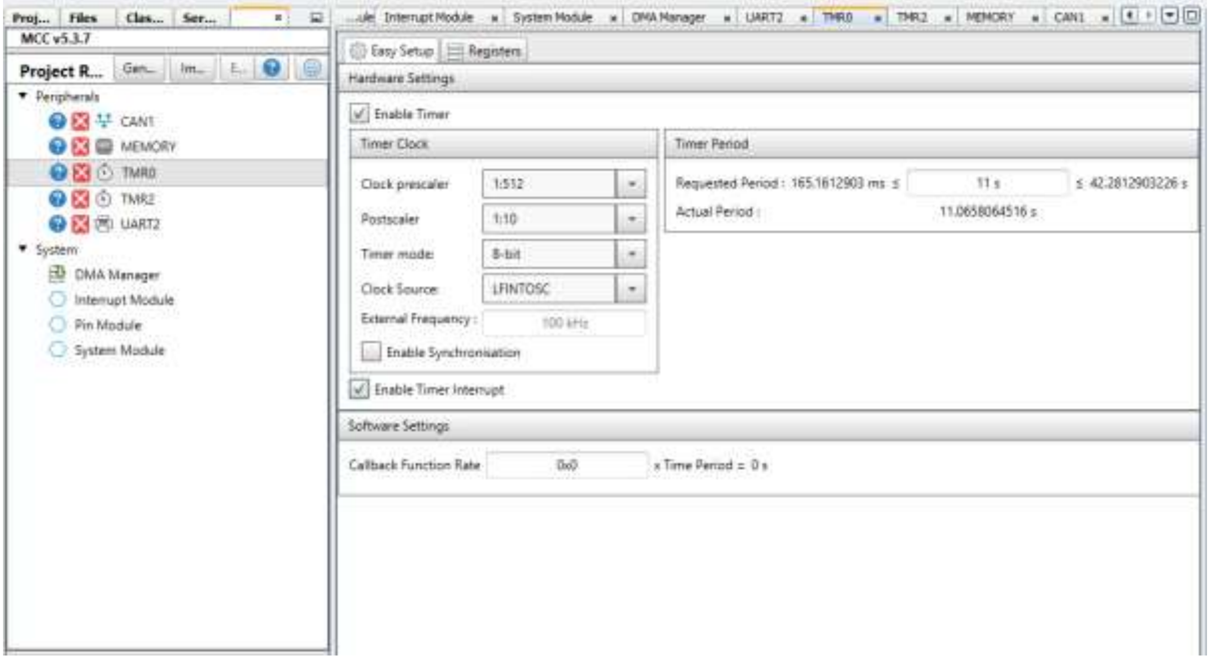
8. In the memory peripheral tick the Add dataEE routines option



Flash Routines are included

☑ Add DataEE Routines

9. In the TMR0 peripheral configure the following option



Hardware Settings

☑ Enable Timer

Timer Clock

Clock prescaler: 1:512

Postscaler: 1:10

Timer mode: 8-bit

Clock Source: LFINTOSC

External Frequency : 100 kHz

□ Enable Synchronisation

☑ Enable Timer Interrupt

Timer Period

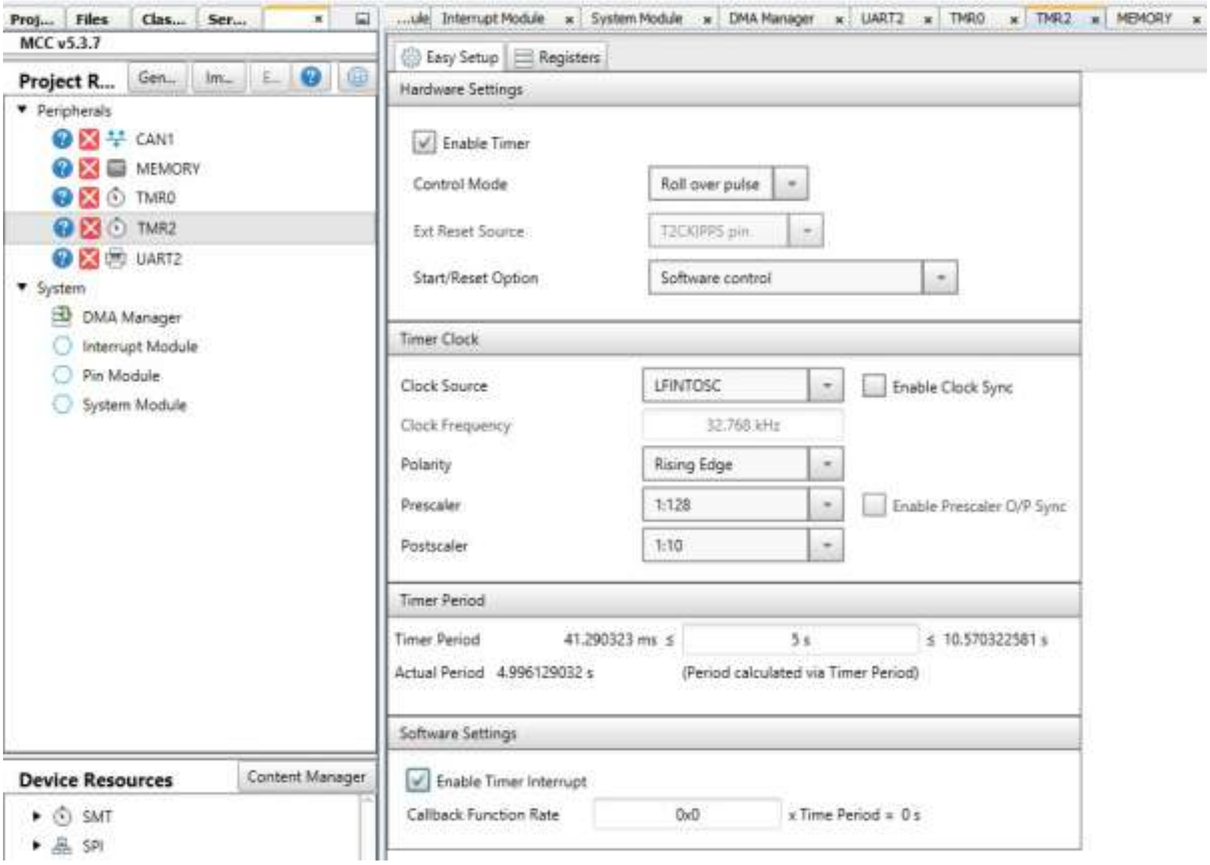
Requested Period : 165.1612903 ms ≤ 11 s ≤ 42.2812903226 s

Actual Period : 11.0658064516 s

Software Settings

Callback Function Rate 0x0 x Time Period = 0 s

10. In the TMR2 peripheral configure the following options



Hardware Settings

☑ Enable Timer

Control Mode: Roll over pulse

Ext Reset Source: T2CKIPPS pin

Start/Reset Option: Software control

Timer Clock

Clock Source: LFINTOSC   □ Enable Clock Sync

Clock Frequency: 32.768 kHz

Polarity: Rising Edge

Prescaler: 1:128   □ Enable Prescaler O/P Sync

Postscaler: 1:10

Timer Period

Timer Period 41.290323 ms ≤ 5 s ≤ 10.570322581 s

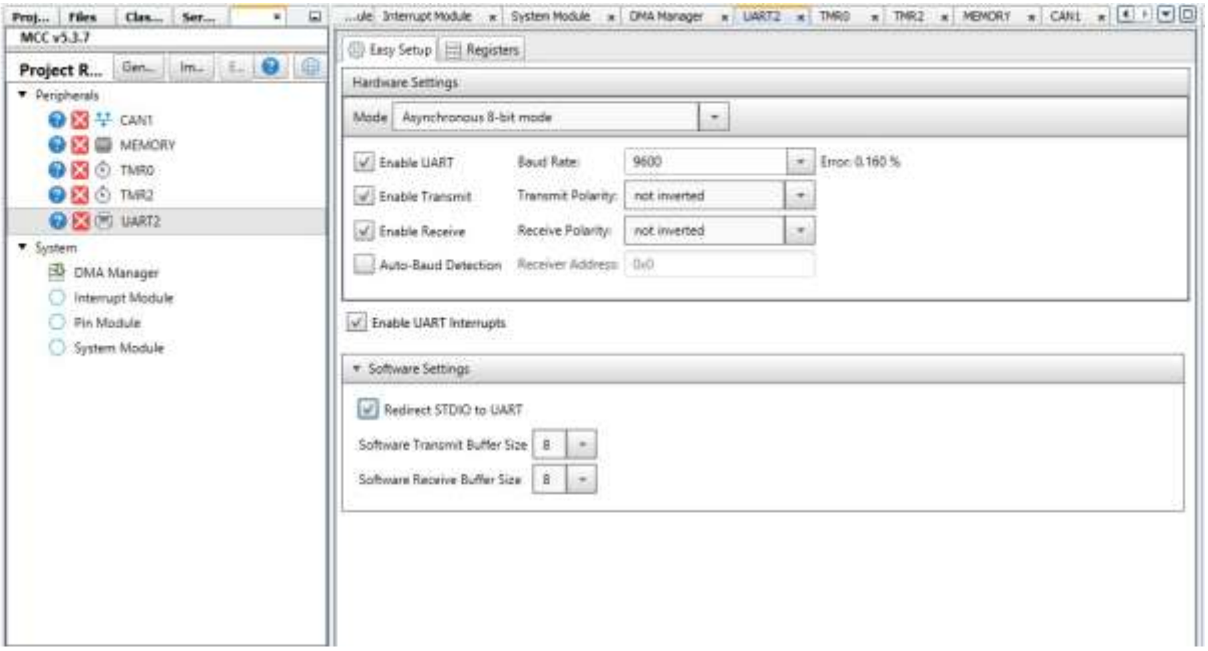
Actual Period 4.996129032 s (Period calculated via Timer Period)

Software Settings

☑ Enable Timer Interrupt

Callback Function Rate 0x0 x Time Period = 0 s

11. In the UART2 peripheral configure the below settings



Hardware Settings

Mode: Asynchronous 8-bit mode

☑ Enable UART   Baud Rate: 9600   Error: 0.160 %

☑ Enable Transmit   Transmit Polarity: not inverted

☑ Enable Receive   Receive Polarity: not inverted

□ Auto-Baud Detection   Receiver Address: 0x0

☑ Enable UART Interrupts

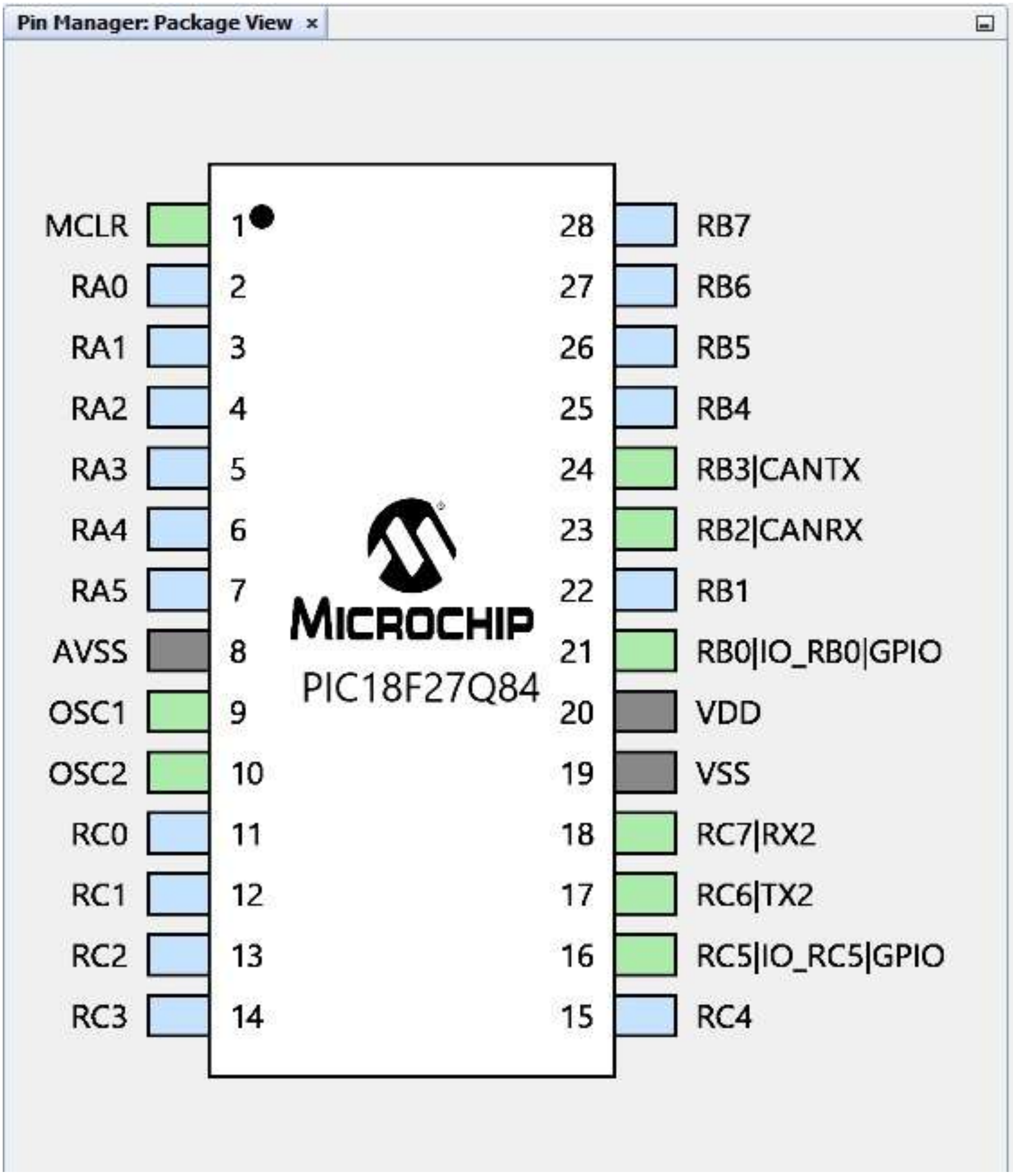
Software Settings

☑ Redirect STDIO to UART

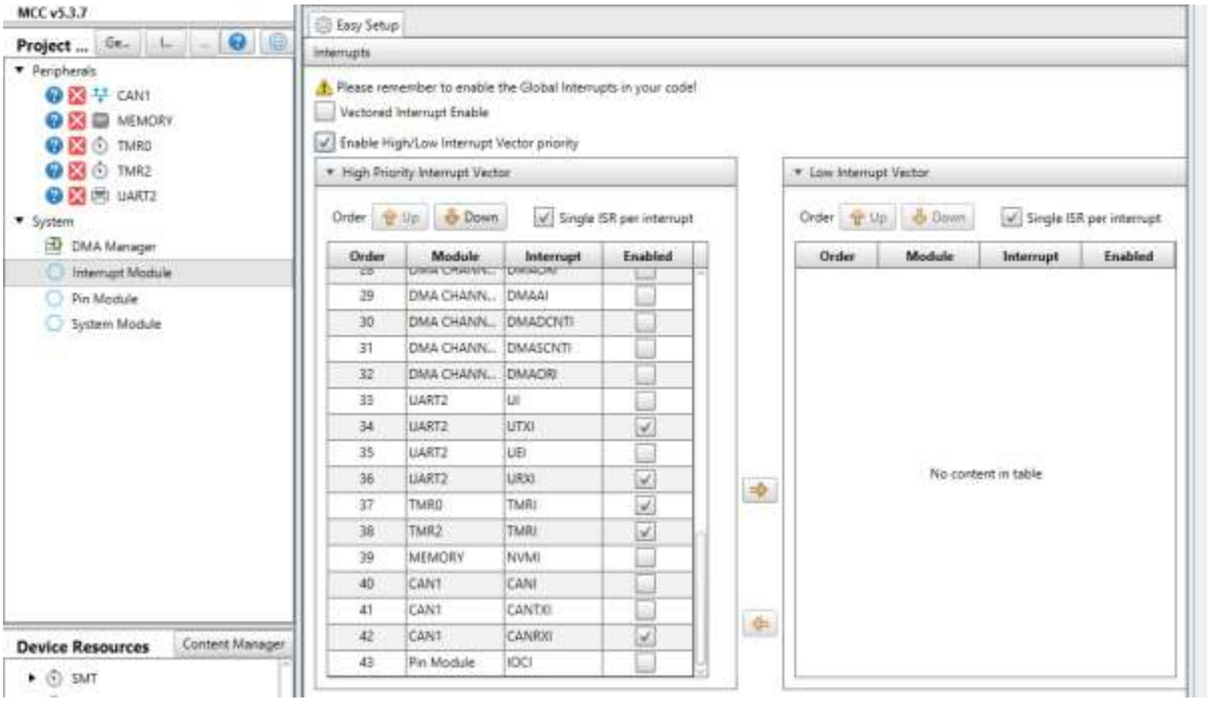
Software Transmit Buffer Size 8

Software Receive Buffer Size 8

12. In the pin manager window configure the below pins and its functionality



13. Make sure the following are enabled in interrupt module window



MCC v5.3.7

Project ... | Ge... | I... | ...

- ▼ Peripherals
  - CAN1
  - MEMORY
  - TMR0
  - TMR2
  - UART2
- ▼ System
  - DMA Manager
  - Interrupt Module
  - Pin Module
  - System Module

Device Resources | Content Manager

▸ SMT

Easy Setup

Interrupts

⚠ Please remember to enable the Global Interrupts in your code!

□ Vectored Interrupt Enable

☑ Enable High/Low Interrupt Vector priority

▼ High Priority Interrupt Vector

Order Up Down ☑ Single ISR per interrupt

Order	Module	Interrupt	Enabled
28	DMA CHANN...	DMAORI	□
29	DMA CHANN...	DMAAI	□
30	DMA CHANN...	DMADCNTI	□
31	DMA CHANN...	DMASCNTI	□
32	DMA CHANN...	DMAORI	□
33	UART2	UI	□
34	UART2	UTXI	☑
35	UART2	UEI	□
36	UART2	URXI	☑
37	TMR0	TMRI	☑
38	TMR2	TMRI	☑
39	MEMORY	NVMI	□
40	CAN1	CANI	□
41	CAN1	CANTXI	□
42	CAN1	CANRXI	☑
43	Pin Module	IOCI	□

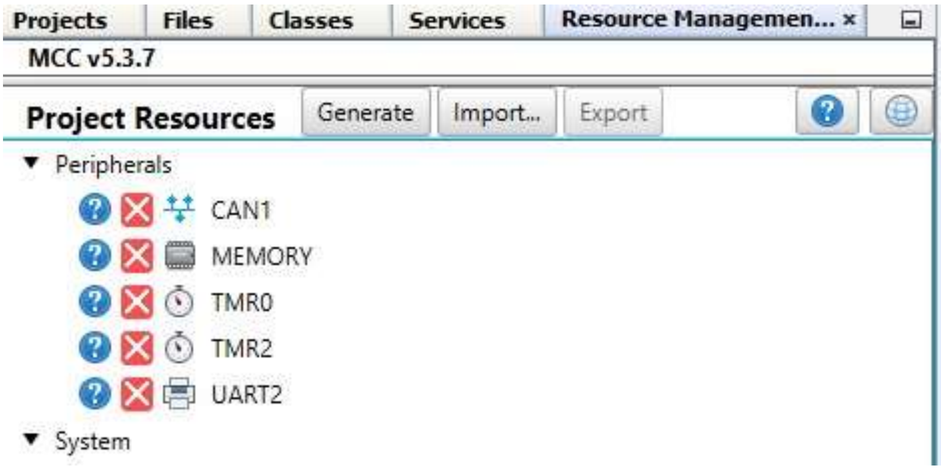
▼ Low Interrupt Vector

Order Up Down ☑ Single ISR per interrupt

Order	Module	Interrupt	Enabled

No content in table

14. Click on the generate button and the libraries and the APIs will be generated in the project folders



MCC v5.3.7

Project Resources | Generate | Import... | Export

- ▼ Peripherals
  - CAN1
  - MEMORY
  - TMR0
  - TMR2
  - UART2
- ▼ System