TB3282

Getting Started with UART Using EUSART on PIC18

Introduction

Author: Alexandru Niculae, Microchip Technology Inc.

The UART-capable peripherals come in different variants on microcontrollers. Sometimes, the peripheral is named UART or USART and sometimes it is called EUSART to emphasize enhanced functionalities. The data sheet of each device shows the type of UART peripheral it has.

The purpose of this document is to describe how to configure the Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART) on PIC18 devices to demonstrate its usage for some common use cases.

Notes:

- For each use case, there are three different implementations, which have the same functionalities: one generated with MPLAB® Code Configurator (MCC), one generated using the Foundation Services MCC Library and one bare metal.
- The MCC generated code offers hardware abstraction layers that ease the use of the code across different devices of the same family. The Foundation Services generated code offers a driver-independent Application Programming Interface (API) and facilitates the portability of code across different platforms. The bare metal code is easier to follow and allows a fast ramp-up on the use case associated code.

While EUSART is a complex peripheral and can work in various modes, this document will use it in Asynchronous mode and describes the following use cases:

- **Send 'Hello World':**
  This example shows how to send a string to the PC and see it in the MPLAB® Data Visualizer Terminal.
- **Send Formatted Messages Using `printf`:**
  This example shows how to enhance the first use case with the ability to use the `printf` function to send messages over EUSART. In this example, messages are Data Stream protocol frames, and the MPLAB Data Visualizer can be used to display them as plots.
- **Receive Control Commands:**
  This example shows how to implement a command line interface. This way, the microcontroller can receive control commands via the EUSART. In this use case, an LED is controlled using commands sent from the MPLAB Data Visualizer.

The examples in this technical brief have been developed using the PIC18F47Q10 Curiosity Nano development board, which is equipped with the QFN40 package.

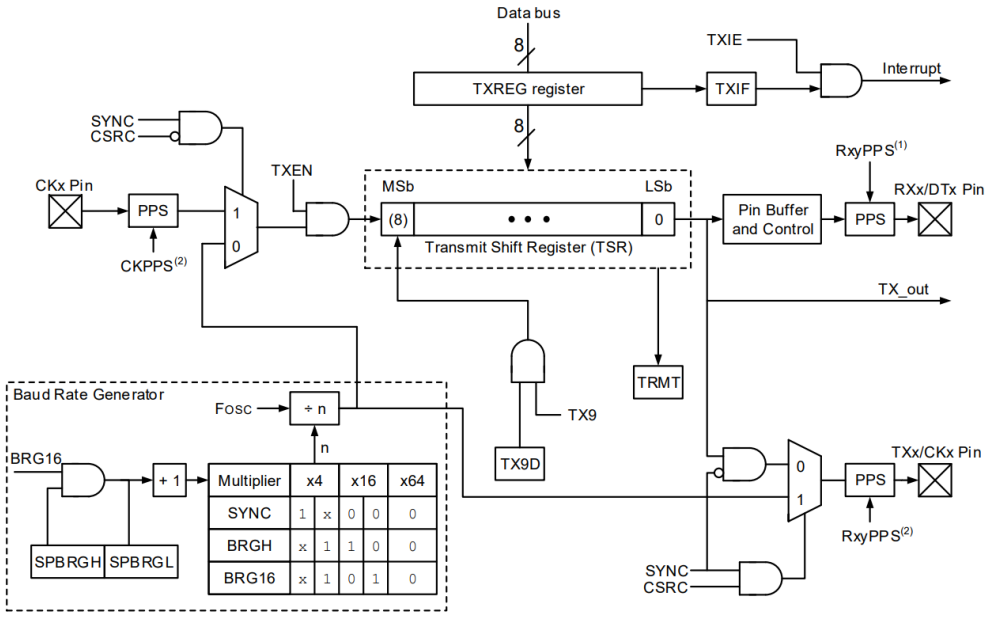
Table of Contents

1. Peripheral Overview

The EUSART module is a serial Input/Output (I/O) communications peripheral. It contains all the clock generators, shift registers and data buffers necessary to perform an input or output serial data transfer, independent of device program execution. The EUSART, also known as a Serial Communications Interface (SCI), can be configured as a full-duplex asynchronous system or half-duplex synchronous system. The Full-Duplex mode is useful for communications with peripheral systems, such as CRT terminals and personal computers. The Half-Duplex Synchronous mode is intended for communications with peripheral devices, such as A/D or D/A integrated circuits, serial EEPROMs or other microcontrollers. These devices typically do not have internal clocks for baud rate generation and require the external clock signal provided by a master synchronous device.

The RXx/DTx and TXx/CKx input pins are selected with the RXxPPS and TXxPPS registers, respectively. TXx, CKx and DTx output pins are selected with each of the pin's RxyPPS register. Since the RX input is coupled with the DT output in Synchronous mode, it is the user's responsibility to select the same pin for both of these functions when operating in Synchronous mode. The EUSART module will control the data direction drivers automatically.

Figure 1-1. EUSART Block Diagram



Note 1: In Synchronous mode, the DT output and RX input PPS selections should enable the same pin.
2: In Master Synchronous mode the TX output and CK input PPS selections should enable the same pin.

2. Send 'Hello World'

This use case demonstrates how to send string messages from the microcontroller to the PC and use MPLAB Data Visualizer to see them. The EUSART will be configured for Asynchronous mode and only the TX pin will be used.

Note:
If a platform without an on-board UART to USB converter is used, there are many options of external UART to USB, such as the MCP2200 and the MCP2221A.

This use case follows the steps:

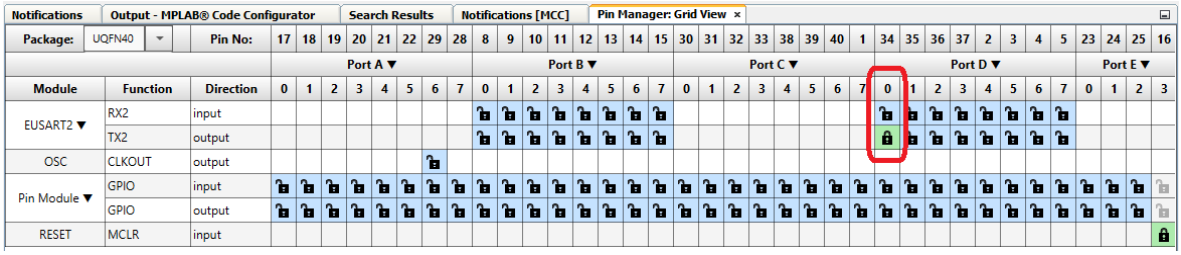
- Configure the system clock
- Configure EUSART2
- Configure the pins

2.1 MCC Generated Code

To generate this project using the MPLAB Code Configurator (MCC), follow these steps:

1. Create a new MPLAB X IDE project .
2. Open the MCC from the toolbar (information on how to install the MCC plug-in can be found on the Microchip website).
3. Go to *Project Resources > System > System Module* and use the following configurations:
   – Oscillator Select: HFINTOSC
   – HF Internal Clock: 4 MHz
   – Clock Divider: 4
   – In the Watchdog Timer Enable field, in the **WWDT** tab, ensure **WDT Disable** is selected
   – In the **Programming** tab, ensure **Low-Voltage Programming Enable** is checked
4. From the Device Resources window, add EUSART2 to the project, then use the following configurations:
   – Mode: Asynchronous
   – Enable EUSART: Checked
   – Enable Transmit: Checked
   – Baud Rate: 9600
   – Transmission-bits: 8 bits
   – Reception-bits: 8 bits
   – Data Polarity: Noninverted
5. Open the *Pin Manager > Grid View* window, select UQFN40 in the MCU package field and select pin RD0 as EUSART TX.

Figure 2-1. Pin Mapping



6. Click Generate in the **Project Resources** tab.
7. In the `main.c` file, which has been generated by MCC:
   – Add the following code in the main function (replacing the existing `while(1)` loop):

```
char msg[] = "Hello World\r\n";
while (1)
{
    for(uint8_t i = 0; i < strlen(msg); i++)
    {
        EUSART2_Write(msg[i]);
    }
}
```

8. Use MPLAB X Data Visualizer as described in the appendix, How to Receive Data in MPLAB X Data Visualizer.

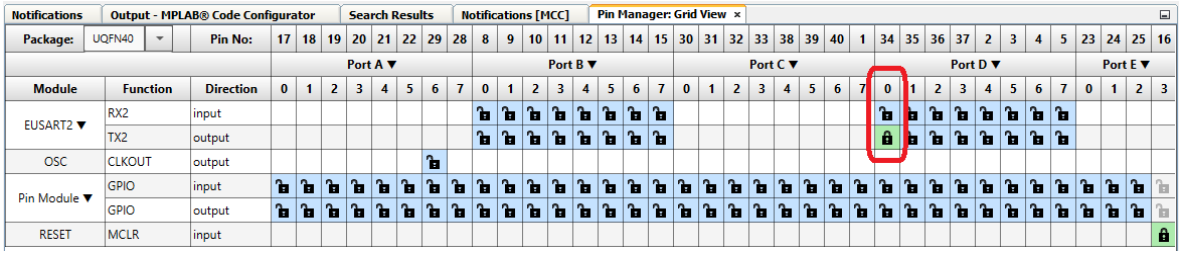
View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

2.2 Foundation Services Generated Code

To generate this project using the Foundation Services library, follow these steps:

1. Create a new MPLAB X project.
2. Open the MCC from the toolbar (information on how to install the MCC plug-in can be found on the Microchip website).
3. Go to *Project Resources > System > System Module* and use the following configurations:
   - Oscillator Select: HFINTOSC
   - HF Internal Clock: 4 MHz
   - Clock Divider: 4
   - In the WDT Enable field, in the **WWDT** tab, ensure **WDT Disable** is selected
   - In the **Programming** tab, ensure **Low-Voltage Programming Enable** is checked
4. In Device Resources under Libraries, find Foundation Services, expand it and double-click **UART** to add it to the project.
5. Click the **+** button at the bottom to add a new Foundation Services federated UART.
6. Select **EUSART2** from the UART drop down in Foundation Services.
7. Open the *Pin Manager > Grid View* window, select **UQFN40** in the MCU package field and select **pin RD0** EUSART TX.

Figure 2-2. Pin Mapping

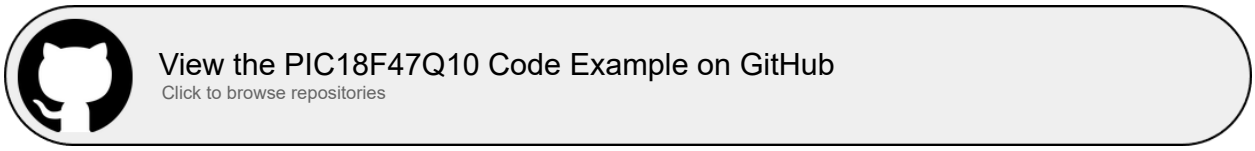


8. Click **Generate** in the **Project Resources** tab.
9. In the `main.c` file, which was generated by MCC:
   - Uncomment the line that enables global interrupts and the one that enables peripheral interrupts
   - Add the following code in the main function (replacing the existing `while(1)` loop):

```
char msg[] = "Hello world\r\n";
while (1)
{
    for(uint8_t i = 0; i < strlen(msg); i++)
    {
```

```
            uart[UART0].Write(msg[i]);
        }
    }
```

10. Use MPLAB X Data Visualizer as described in the appendix, How to Receive Data in MPLAB Data Visualizer.



2.3 Bare Metal Code

The first step is to configure the microcontroller to disable the Watchdog Timer (WDT) and to enable the Low-Voltage Programming (LVP).

```
/* WDT operating mode → WDT Disabled */
#pragma config WDTE = OFF
/* Low-voltage programming enabled, RE3 pin is MCLR */
#pragma config LVP = ON
```

2.3.1 How to Configure the System Clock

To use the High-Frequency Internal Oscillator (HFINTOSC) at 1 MHz, two settings must be made in the registers. First, select HFINTOSC as the oscillator source in the OSCCON1 register by writing to the NOSC bit field:

```
/* Set HFINTOSC as new oscillator source. */
OSCCON1bits.NOSC = 0b110;
```

Then, select the nominal frequency of 1 MHz in the OSCFRQ register:

```
/* Set HFFRQ to 1 MHz. */
OSCFRQbits.HFFRQ = 0;
```

2.3.2 How to Configure EUSART2

EUSART2 will be configured for 9600 baud rate and the standard 8-N-1 (eight data bits, no parity bit and one Stop bit) frame format.

Given the configured frequency and the desired baud rate of 9600, the 16-bit Baud Rate Generator (BRG16) must be used, and High Baud Rate (BRGH) must be enabled. The SPEN bit enables the Serial Port (EUSART) as a whole, while TXEN enables its transmitter.

```
/* Transmit Enable */
TX2STAbits.TXEN = 1;
/* High Baud Rate Select */
TX2STAbits.BRGH = 1;
/* 16-bit Baud Rate Generator is used */
BAUD2CONbits.BRG16 = 1;
/* Serial Port Enable */
RC2STAbits.SPEN = 1;
```

The value to be written in SP2BRG is found in the table below, taken from the device data sheet.

Table 2-1. Sample Baud Rates for Asynchronous Modes

BAUD RATE	SYNC = 0, BRGH = 1, BRG16 = 1 or SYNC = 1, BRG16 = 1											
	F_{OSC} = 8.000 MHz			F_{OSC} = 4.000 MHz			F_{OSC} = 3.6864 MHz			F_{OSC} = 1.000 MHz		
	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)
300	300.0	0.00	6666	300.0	0.01	3332	300.0	0.00	3071	300.1	0.04	832
1200	1200	-0.02	1666	1200	0.04	832	1200	0.00	767	1202	0.16	207
2400	2401	0.04	832	2398	0.08	416	2400	0.00	383	2404	0.16	103
9600	**9615**	**0.16**	**207**	**9615**	**0.16**	**103**	**9600**	**0.00**	**95**	**9615**	**0.16**	**25**
10417	10417	0	191	10417	0.00	95	10473	0.53	87	10417	0.00	23
19.2k	19.23k	0.16	103	19.23k	0.16	51	19.20k	0.00	47	19.23k	0.16	12
57.6k	57.14k	-0.79	34	58.82k	2.12	16	57.60k	0.00	15	—	—	—
115.2k	117.6k	2.12	16	111.1k	-3.55	8	115.2k	0.00	7	—	—	—

```
/* Baud rate 9600 */
SP2BRGL = 25;
SP2BRGH = 0;
```

2.3.3 How to Configure the Pins

Since only the transmission is necessary for this use case, only the TX pin must be configured. The pin used as TX in this example, is the RD0 pin and it must be configured using the Peripheral Pin Select (PPS). To find the value that needs to be written to the RD0PPS register, inspect the Peripheral PPS Output Selection Codes table below, taken from the device data sheet.

Table 2-2. Peripheral PPS Output Selection Codes

RxyPPS	Pin Rxy Output Source	PORT to Which Output can be Directed							
		28-Pin Devices			40-Pin Devices				
0x0C	EUSART2 (DT)	—	B	C	—	B	—	D	—
0x0B	**EUSART2 (TX/CK)**	**—**	**B**	**C**	**—**	**B**	**—**	**D**	**—**
0x0A	EUSART1 (DT)	—	B	C	—	B	C	—	—
0x09	EUSART1 (TX/CK)	—	B	C	—	B	C	—	—

The following line will route the EUSART2 TX to RD0:

```
/* RD0 is TX2 */
RD0PPS = 0x0B;
```

The pin direction is set by default as output, but if it were not, the following line sets it.

```
/* Configure RD0 as output. */
TRISDbits.TRISD0 = 0;
```

Table 2-3. Pin Locations

Function	Pin
EUSART2 TX	RD0

Before sending data, the user needs to check if the previous transmission is complete by checking the PIR3.TXnIF bit field. The following code example waits until the transmit buffer is empty, then writes a character to the TXnREG register:

```
static void EUSART2_write(uint8_t txData)
{
    while(0 == PIR3bits.TX2IF)
    {
        ;
    }

    TX2REG = txData;
}
```

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

Use MPLAB X Data Visualizer as described in the appendix, How to Receive Data in MPLAB X Data Visualizer.

3. Send Formatted Messages Using `printf`

It is a common use case for an application to send a message with variable fields over EUSART, such as when the application reports a sensor reading. Using formatted messages is a very flexible approach and reduces the number of code lines. This can be accomplished by redirecting Standard Input/Output (STDIO) to EUSART.

In this example, a counter value and twice its value are sent in a binary format, using the Data Stream Protocol, over EUSART. The MPLAB X Data Visualizer has a built-in Data Stream Protocol decoder and can display charts of the values in the message in real time.

This use case follows the steps:

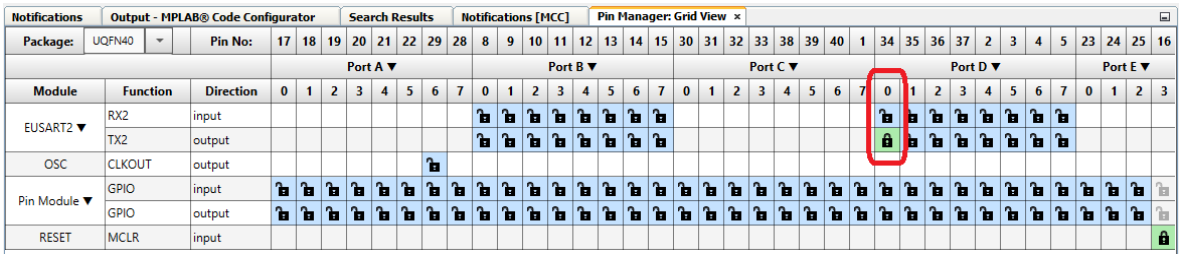
- Configure the system clock
- Configure EUSART2
- Configure the pins

3.1 MCC Generated Code

To generate this project using the MCC, follow these steps:

1. Create a new MPLAB X IDE project.
2. Open the MCC from the toolbar (information on how to install the MCC plug-in can be found on the Microchip website).
3. Go to *Project Resources > System > System Module* and use the following configurations:
   - Oscillator Select: HFINTOSC
   - HF Internal Clock: 4 MHz
   - Clock Divider: 4
   - In the Watchdog Timer Enable field, in the **WWDT** tab, ensure **WDT Disable** is selected
   - In the **Programming** tab, make sure **Low-Voltage Programming Enable** is checked
4. From the Device Resources window, add EUSART2 to the project, then use the following configurations:
   - Mode: Asynchronous
   - Enable EUSART: Checked
   - Enable Transmit: Checked
   - Baud Rate: 9600
   - Transmission-bits: 8 bits
   - Reception-bits: 8 bits
   - Data Polarity: Noninverted
   - Redirect STDIO to USART: Checked
5. Open *Pin Manager > Grid View* window, select UQFN40 in the MCU package field and select pin RD0 as EUSART TX.

Figure 3-1. Pin Mapping



6. Click **Generate** in the **Project Resources** tab.
7. In the `main.c` file, which has been generated by MCC:

– Add the following defines:

```
#define START_DATA_STREAM_PROTOCOL 0x03
#define STOP_DATA_STREAM_PROTOCOL 0xFC
```

– Add the following code in the main function (replacing the existing `while(1)` loop):

```
uint8_t cnt = 0;
while (1)
{
    printf("%c%c%c%c", START_DATA_STREAM_PROTOCOL, cnt, cnt * 2,
STOP_DATA_STREAM_PROTOCOL);
    cnt = cnt + 1;
    __delay_ms(50);
}
```

8. Use MPLAB X Data Visualizer as described in the appendix, How to Configure MPLAB X Data Visualizer to Decode Data Stream Protocol.

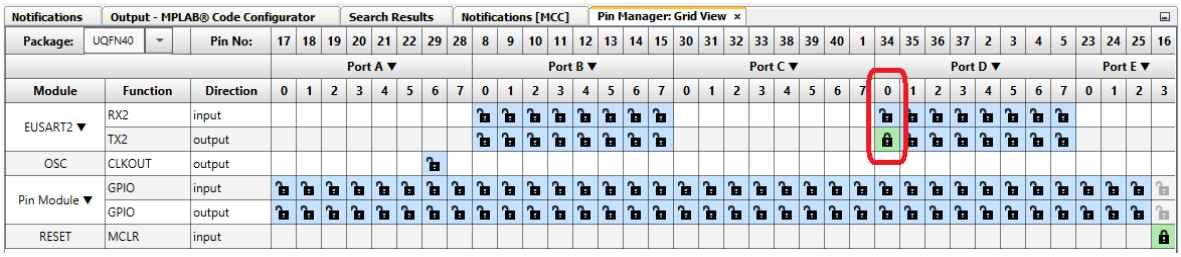
View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

3.2 Foundation Services Generated Code

To generate this project using the Foundation Services library, follow these steps:

1. Create a new MPLAB X IDE project for PIC18F47Q10.
2. Open the MCC from the toolbar (information on how to install the MCC plug-in can be found on the Microchip website).
3. Go to *Project Resources > System > System Module* and use the following configurations:
   – Oscillator Select: HFINTOSC
   – HF Internal Clock: 4 MHz
   – Clock Divider: 4
   – In the WDT Enable field, in the **WWDT** tab, ensure **WDT Disable** is selected
   – In the **Programming** tab, make sure Low-Voltage Programming Enable is checked
4. In Device Resource, under Libraries, find Foundation Services, expand it and double-click **UART** to add it to the project.
5. Click the **+** button at the bottom to add a new Foundation Services federated UART.
6. Select **EUSART2** from the UART drop down in Foundation Services.
7. Open the *Pin Manager > Grid View* window, select **UQFN40** in the MCU package field and select **pin RD0** EUSART TX.

Figure 3-2. Pin Mapping



8. In Project Resources, under Peripherals, select **EUSART2** and use the following configuration:
   – Redirect STDIO to USART: Checked

9. Click Generate in the **Project Resources** tab.
10. In the `main.c` file, which was generated by MCC:
    - Uncomment the line that enables global interrupts and the one that enables peripheral interrupts
    - Add the following defines:

```
#define START_DATA_STREAM_PROTOCOL 0x03
#define STOP_DATA_STREAM_PROTOCOL 0xFC
```

    - Add the following code in the main function (replacing the existing `while(1)` loop):

```
uint8_t cnt = 0;
while (1)
{
    printf("%c%c%c%c", START_DATA_STREAM_PROTOCOL, cnt, cnt * 2,
STOP_DATA_STREAM_PROTOCOL);
    cnt = cnt + 1;
    __delay_ms(50);
}
```

11. Use MPLAB X Data Visualizer as described in the appendix, How to configure MPLAB X Data Visualizer to Decode Data Stream Protocol.

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

3.3 Bare Metal Code

The first step will be to configure the microcontroller to disable the WDT and to enable the LVP.

```
/* WDT operating mode → WDT Disabled */
#pragma config WDTE = OFF
/* Low-voltage programming enabled, RE3 pin is MCLR */
#pragma config LVP = ON
```

3.3.1 How to Configure the System Clock

To use the HFINTOSC at 1 MHz, two settings must be made in the registers. First, select HFINTOSC as the oscillator source in the OSCCON1 register by writing to the NOSC bit field:

```
/* Set HFINTOSC as new oscillator source. */
OSCCON1bits.NOSC = 0b110;
```

Then, select the nominal frequency of 1 MHz in the OSCFRQ register:

```
/* Set HFFRQ to 1 MHz. */
OSCFRQbits.HFFRQ = 0;
```

3.3.2 How to Configure EUSART2

EUSART2 will be configured for 9600 baud rate and the standard 8-N-1 (eight data bits, no parity bit and one Stop bit) frame format.

Given the configured frequency and the desired baud rate of 9600, the 16-bit Baud Rate Generator (BRG16) must be used and High Baud Rate (BRGH) must be enabled. The SPEN bit enables the Serial Port (EUSART) as a whole, while TXEN enables its transmitter.

```
/* Transmit Enable */
TX2STAbits.TXEN = 1;
/* High Baud Rate Select */
TX2STAbits.BRGH = 1;
/* 16-bit Baud Rate Generator is used */
BAUD2CONbits.BRG16 = 1;
/* Serial Port Enable */
RC2STAbits.SPEN = 1;
```

The value to be written in SP2BRG is found in the table below, taken from the device data sheet.

Table 3-1. Sample Baud Rates for Asynchronous Modes

BAUD RATE	SYNC = 0, BRGH = 1, BRG16 = 1 or SYNC = 1, BRG16 = 1											
	F_{OSC} = 8.000 MHz			F_{OSC} = 4.000 MHz			F_{OSC} = 3.6864 MHz			F_{OSC} = 1.000 MHz		
	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)
300	300.0	0.00	6666	300.0	0.01	3332	300.0	0.00	3071	300.1	0.04	832
1200	1200	-0.02	1666	1200	0.04	832	1200	0.00	767	1202	0.16	207
2400	2401	0.04	832	2398	0.08	416	2400	0.00	383	2404	0.16	103
9600	**9615**	**0.16**	**207**	**9615**	**0.16**	**103**	**9600**	**0.00**	**95**	**9615**	**0.16**	**25**
10417	10417	0	191	10417	0.00	95	10473	0.53	87	10417	0.00	23
19.2k	19.23k	0.16	103	19.23k	0.16	51	19.20k	0.00	47	19.23k	0.16	12
57.6k	57.14k	-0.79	34	58.82k	2.12	16	57.60k	0.00	15	—	—	—
115.2k	117.6k	2.12	16	111.1k	-3.55	8	115.2k	0.00	7	—	—	—

```
/* Baud rate 9600 */
SP2BRGL = 25;
SP2BRGH = 0;
```

3.3.3 How to Configure the Pins

Since only the transmission is necessary for this use case, only the TX pin must be configured. The pin used as TX, in this example, is the RD0 pin and it must be configured using the Peripheral Pin Select (PPS). To find the value that needs to be written to the RD0PPS register, inspect the Peripheral PPS Output Selection Codes table below, taken from the device data sheet.

Table 3-2. Peripheral PPS Output Selection Codes

RxyPPS	Pin Rxy Output Source	PORT to Which Output can be Directed							
		28-Pin Devices			40-Pin Devices				
0x0C	EUSART2 (DT)	—	B	C	—	B	—	D	—
0x0B	**EUSART2 (TX/CK)**	**—**	**B**	**C**	**—**	**B**	**—**	**D**	**—**
0x0A	EUSART1 (DT)	—	B	C	—	B	C	—	—

.....continued									
RxyPPS	Pin Rxy Output Source	PORT to Which Output can be Directed							
		28-Pin Devices			40-Pin Devices				
0x09	EUSART1 (TX/CK)	—	B	C	—	B	C	—	—

The following line will direct the EUSART2 TX to RD0:

```
/* RD0 is TX2 */
RD0PPS = 0x0B;
```

The pin direction is set by default output, but otherwise, the following line sets it.

```
/* Configure RD0 as output. */
TRISDbits.TRISD0 = 0;
```

Table 3-3. EUSART Pin Locations

Function	Pin
EUSART2 TX	RD0

Before sending data, the user needs to check if the previous transmission is complete by checking the PIR3.TXnIF bit field. The following code example waits until the transmit buffer is empty, then writes a character to the TXnREG register:

```
static void EUSART2_write(uint8_t txData)
{
    while(0 == PIR3bits.TX2IF)
    {
        ;
    }

    TX2REG = txData;
}
```

3.3.4 How to Implement the Function that Handles STDIO

The function that handles STDIO is `void puthc(char c)`. Implementing this function will transmit its char argument over EUSART and will cause the `printf` to be redirected to EUSART.

```
void putch(char txData)
{
    EUSART2_write(txData);
}
```

The following line will send messages over EUSART:

```
printf("%c%c%c%c", START_DATA_STREAM_PROTOCOL, cnt, cnt * 2, STOP_DATA_STREAM_PROTOCOL);
```

The Data Stream Protocol frame has start and stop tokens, as inverse/one's complement each other. In between those, the payload may contain any number of values. The protocol must be sent in binary format.

Note: The `printf` function uses placeholder specifiers in the format string to mark where to insert variables. Some of the available placeholders are displayed in the table below.

Table 3-4. `printf` Specifiers

Placeholder	Description
%d	Insert a signed integer
%f	Insert a floating point number
%s	Insert a sequence of characters
%c	Insert a character
%x	Insert integer unsigned in `hex` format

In this example, the %c specifier is used, because the row binary value will be transmitted.

Note: The %x specifier inserts the hex value as ASCII characters, thus, this is not well-suited here.

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

Use MPLAB X Data Visualizer as described in the appendix, How to Configure MPLAB X Data Visualizer to Decode Data Stream Protocol.

4. Receive Control Commands

One important usage of the EUSART represents the implementation of a command line interface. This way, the microcontroller can receive control commands via EUSART. It is convenient to use the line terminator as command delimiter, so for this use case, EUSART will read full lines and then check if the line contains a valid command.

This use case follows the steps:

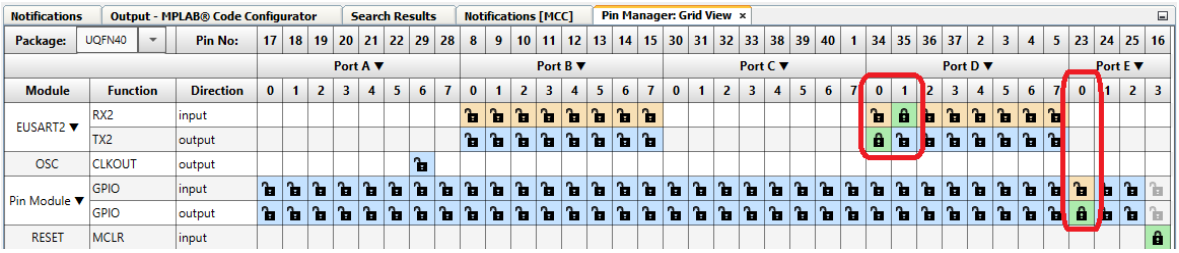
- Configure the system clock
- Configure EUSART2
- Configure the pins
- Implement STDIO receive and send functions
- Read lines and execute valid commands

4.1 MCC Generated Code

To generate this project using MPLAB Code Configurator (MCC), follow these steps:

1. Create a new MPLAB X IDE project.
2. Open the MCC from the toolbar (information on how to install the MCC plug-in can be found on the Microchip website).
3. Go to *Project Resources > System > System Module* and use the following configurations:
   - Oscillator Select: HFINTOSC
   - HF Internal Clock: 4 MHz
   - Clock Divider: 4
   - In the Watchdog Timer Enable field, in the **WWDT** tab, ensure **WDT Disable** is selected
   - In the **Programming** tab, make sure **Low-Voltage Programming Enable** is checked
4. From the Device Resources window, add EUSART2 to the project, then use the following configurations:
   - Mode: Asynchronous
   - Enable EUSART: Checked
   - Enable Transmit: Checked
   - Baud Rate: 9600
   - Transmission-bits: 8 bits
   - Reception-bits: 8 bits
   - Data Polarity: Noninverted
   - Redirect STDIO to USART: Checked
5. Open *Pin Manager > Grid View* window, select UQFN40 in the MCU package field and make the following pin configurations:
   - RD0 – EUSART2 TX
   - RD1 – EUSART RX
   - RE0 – GPIO output

Figure 4-1. Pin Mapping



6. In Project Resource, locate Pin Module and rename RE0 from IO_RE0 to LED0; uncheck the Analog check box.

7. Click **Generate** in the **Project Resources** tab.
8. Replace the `main.c` file with the code provided in the code example.
9. Use MPLAB X Data Visualizer as described in the appendix, Send Commands from MPLAB X Data Visualizer.

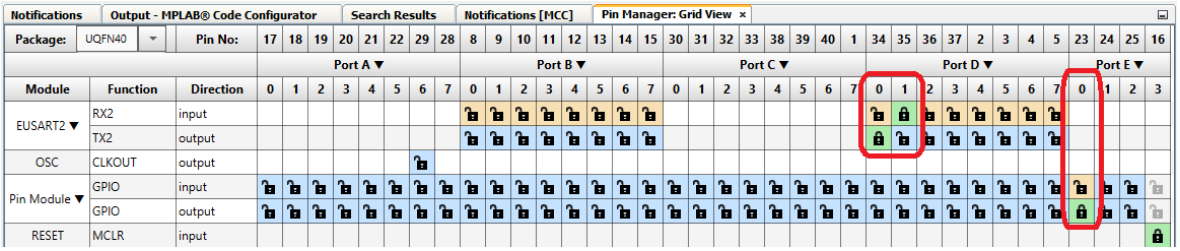
View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

4.2 Foundation Services Generated Code

To generate this project using the Foundation Services library, follow these steps:

1. Create a new MPLAB X IDE project.
2. Open the MCC from the toolbar (information on how to install the MCC plug-in can be found on the Microchip website).
3. Go to *Project Resources > System > System Module* and use the following configurations:
   - Oscillator Select: HFINTOSC
   - HF Internal Clock: 4 MHz
   - Clock Divider: 4
   - In the WDT Enable field, in the **WWDT** tab, ensure **WDT Disable** is selected
   - In the **Programming** tab, make sure **Low-Voltage Programming Enable** is checked
4. In device resource, under Libraries find Foundation Services, expand it and double-click UART it to add it to the project.
5. Click the **+** button at the bottom to add a new Foundation Services federated UART.
6. Select EUSART2 from the UART drop down in Foundation Services.
7. Open the *Pin Manager > Grid View* window, select **UQFN40** in the MCU package field and make the following pin configurations:
   - RD0 – EUSART TX
   - RD1 – EUSART RX
   - RE0 – GPIO output

Figure 4-2. Pin Mapping



8. In Project Resources, under Peripherals, select **EUSART2** and use the following configuration:
   - Redirect STDIO to USART: Checked
9. In Project Resources, locate Pin Module and rename RE0 from IO_RE0 to LED0, and uncheck the Analog check box.
10. Click **Generate** in the **Project Resources** tab.
11. Replace the `main.c` file with the code provided in the code example.
12. Use MPLAB X Data Visualizer as described in the appendix, Send Commands from MPLAB X Data Visualizer.

View the PIC18F47Q10 Code Example on GitHub
Click to browse repositories

4.3 Bare Metal Code

The first step will be to configure the microcontroller to disable the WDT and to enable the LVP.

```
/* WDT operating mode → WDT Disabled */
#pragma config WDTE = OFF
/* Low-voltage programming enabled, RE3 pin is MCLR */
#pragma config LVP = ON
```

4.3.1 How to Configure the System Clock

To use the HFINTOSC at 1 MHz, two settings must be made in the register. First, select HFINTOSC as the oscillator source in the OSCCON1 register by writing to the NOSC bit field:

```
/* Set HFINTOSC as new oscillator source. */
OSCCON1bits.NOSC = 0b110;
```

Then, select the nominal frequency of 1 MHz in the OSCFRQ register:

```
/* Set HFFRQ to 1 MHz. */
OSCFRQbits.HFFRQ = 0;
```

4.3.2 How to Configure EUSART2

EUSART2 will be configured for 9600 baud rate and the standard 8-N-1 (eight data bits, no parity bit and one Stop bit) frame format.

Given the configured frequency and the desired baud rate of 9600, the 16-bit Baud Rate Generator (BRG16) must be used and High Baud Rate (BRGH) must be enabled. The SPEN bit enables the Serial Port (EUSART) as a whole, while TXEN enables its transmitter.

In Asynchronous mode, the EUSART supports two types of receive: Single Receive and Continuous Receive. Since a command can be received at any time, Continuous Receive mode will be used.

```
/* 16-bit Baud Rate Generator is used */
BAUD2CONbits.BRG16 = 1;
/* Transmit Enable */
TX2STAbits.TXEN = 1;
/* High Baud Rate Select */
TX2STAbits.BRGH = 1;
/* Continuous Receive Enable */
RC2STAbits.CREN = 1;
/* Serial Port Enable
```

The value to be written in SP2BRG is found in the table below, taken from the device data sheet.

Table 4-1. Sample Baud Rates for Asynchronous Modes

BAUD RATE	SYNC = 0, BRGH = 1, BRG16 = 1 or SYNC = 1, BRG16 = 1											
	F_{OSC} = 8.000 MHz			F_{OSC} = 4.000 MHz			F_{OSC} = 3.6864 MHz			F_{OSC} = 1.000 MHz		
	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)
300	300.0	0.00	6666	300.0	0.01	3332	300.0	0.00	3071	300.1	0.04	832
1200	1200	-0.02	1666	1200	0.04	832	1200	0.00	767	1202	0.16	207

.....continued

BAUD RATE	SYNC = 0, BRGH = 1, BRG16 = 1 or SYNC = 1, BRG16 = 1											
	F_{OSC} = 8.000 MHz			F_{OSC} = 4.000 MHz			F_{OSC} = 3.6864 MHz			F_{OSC} = 1.000 MHz		
	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)	Actual Rate	% Error	SPBRG value (decimal)
2400	2401	0.04	832	2398	0.08	416	2400	0.00	383	2404	0.16	103
9600	**9615**	**0.16**	**207**	**9615**	**0.16**	**103**	**9600**	**0.00**	**95**	**9615**	**0.16**	**25**
10417	10417	0	191	10417	0.00	95	10473	0.53	87	10417	0.00	23
19.2k	19.23k	0.16	103	19.23k	0.16	51	19.20k	0.00	47	19.23k	0.16	12
57.6k	57.14k	-0.79	34	58.82k	2.12	16	57.60k	0.00	15	—	—	—
115.2k	117.6k	2.12	16	111.1k	-3.55	8	115.2k	0.00	7	—	—	—

```
/* Baud rate 9600 */
SP2BRGL = 25;
SP2BRGH = 0;
```

4.3.3 How to Configure the Pins

In this example, the pin used as TX is RD0 and it must be configured using the Peripheral Pin Select (PPS). To find the value that needs to be written to the RD0PPS register, inspect the Peripheral PPS Output Selection Codes table below, taken from the device data sheet.

Table 4-2. Peripheral PPS Output Selection Codes

RxyPPS	Pin Rxy Output Source	PORT to Which Output can be Directed							
		28-Pin Devices			40-Pin Devices				
0x0C	EUSART2 (DT)	—	B	C	—	B	—	D	—
0x0B	**EUSART2 (TX/CK)**	**—**	**B**	**C**	**—**	**B**	**—**	**D**	**—**
0x0A	EUSART1 (DT)	—	B	C	—	B	C	—	—
0x09	EUSART1 (TX/CK)	—	B	C	—	B	C	—	—

The following line will direct the EUSART2 TX to RD0:

```
/* RD0 is TX2 */
RD0PPS = 0x0B;
```

The pin direction is set by default output, but if it were not, the following line sets it.

```
/* Configure RD0 as output. */
TRISDbits.TRISD0 = 0;
```

The pin used as RX is RD1, so using PPS, EUSART2 RX must be routed to this pin. For this, the RX2PPS register is used. The value to be written is determined based on the xxxPPS register definition:

Figure 4-3. Peripheral xxx Input Selection

17.9.1 Peripheral xxx Input Selection

Name: xxxPPS

Bit	7	6	5	4	3	2	1	0
				PORT[1:0]		PIN[2:0]		
Access				R/W	R/W	R/W	R/W	R/W
Reset				g	g	g	g	g

Bits 4:3 – PORT[1:0] Peripheral xxx Input PORT Selection bits
See the input selection table for a list of available ports and default pin locations.

Value	Description
11	PORTD
10	PORTC
01	PORTB
00	PORTA

Bits 2:0 – PIN[2:0] Peripheral xxx Input Pin Selection bits

Value	Description
111	Peripheral input is from PORTx Pin 7 (Rx7)
110	Peripheral input is from PORTx Pin 6 (Rx6)
101	Peripheral input is from PORTx Pin 5 (Rx5)
100	Peripheral input is from PORTx Pin 4 (Rx4)
011	Peripheral input is from PORTx Pin 3 (Rx3)
010	Peripheral input is from PORTx Pin 2 (Rx2)
001	Peripheral input is from PORTx Pin 1 (Rx1)
000	Peripheral input is from PORTx Pin 0 (Rx0)

Therefore, the following line routes RX2 to pin RD1.

```
RX2PPS = 0b00011001;
```

Since the pin is not a digital input by default, this needs to be configured (enable digital input buffers using the ANSELD register and set as input using the TRISD register).

```
/* Configure RD1 as input. */
TRISDbits.TRISD1 = 1;
/* Enable RD1 digital input buffers.*/
ANSELDbits.ANSELD1 = 0;
```

The LED pin, RE0 in this example, must be configured as a digital output.

```
/* Configure RE0 as output. */
TRISEbits.TRISE0 = 0;
```

Table 4-3. EUSART Pin Locations

Function	Pin
EUSART2 TX	RD0
EUSART2 RX	RD1
LED0	RE0

Before sending data, the user needs to check if the previous transmission is complete by checking the PIR3.TXnIF bit field. The following code example waits until the transmit buffer is empty, then writes a character to the TXnREG register:

```
static void EUSART2_write(uint8_t txData)
{
    while(0 == PIR3bits.TX2IF)
    {
        ;
    }
    TX2REG = txData;
}
```

Before reading the data, the user must wait for it to be available by polling the PIR3.RCnIF flag.

```
uint8_t EUSART2_read(void)
{
    while(0 == PIR3bits.RC2IF)
    {
        ;
    }

    return RC2REG;
}
```

4.3.4 Implement STDIO Receive and Send Functions

The function that handles STDIO is void `puthc(char c)`. Implementing this function will transmit its char argument over EUSART and will cause `printf` to be redirected to EUSART.

```
void putch(char txData)
{
    EUSART2_write(txData);
}
```

Similarly, to implementing `putch`, `char getch(void)` must be implemented so that the EUSART incoming data are mapped to STDIO.

```
char getch(void)
{
    return EUSART2_read();
}
```

4.3.5 How to Read Lines and Execute Valid Commands

The following code snippet reads one line of data and stores it in an array. A valid line is shorter than the array length in this example.

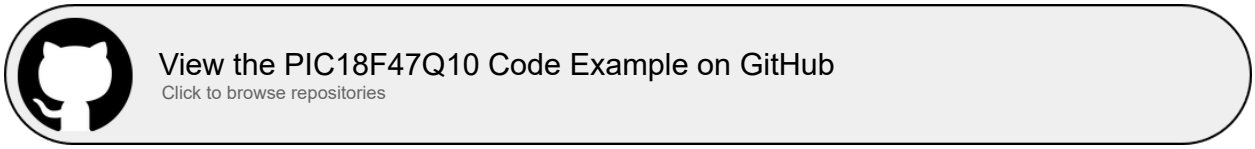
The array index is reset to zero when reaching the array end, to avoid a bugger overflow error in case of longer lines received. The characters, '\n' (line feed) and '\r' (carriage return), are ignored, because they are part of the line terminator. When '\n' is found, the string end (NULL) is added to the command, and the function '`executeCommand`' will interpret it and change the state of the LED .

```
char command[MAX_COMMAND_LEN];
uint8_t index = 0;
char c;
while(1)
{
    c = getch();
    if(c != '\n' && c != '\r')
```

```
    {
        command[index++] = c;
        if(index > MAX_COMMAND_LEN)
        {
            index = 0;
        }
    }

    if(c == '\n')
    {
        command[index] = '\0';
        index = 0;
        executeCommand(command);
    }
}
```

Using the MPLAB X Data Visualizer, ‘LED ON’ and ‘LED OFF’ commands can be sent to the board.



Use the MPLAB X Data Visualizer as described in the appendix, Send Commands from MPLAB X Data Visualizer.

5. References

1. Getting Started with USART
2. MPLAB Code Configurator User's Guide
3. PIC1000: Getting Started with Writing C-Code for PIC16F and PIC18F

6. Appendix

6.1 How to Receive Data in MPLAB® X Data Visualizer

First, ensure the plug-in is installed; if not, install it from within MPLAB® X Integrated Development Environment (IDE) by going to *Tools > Plug-ins > Available Plug-ins*.

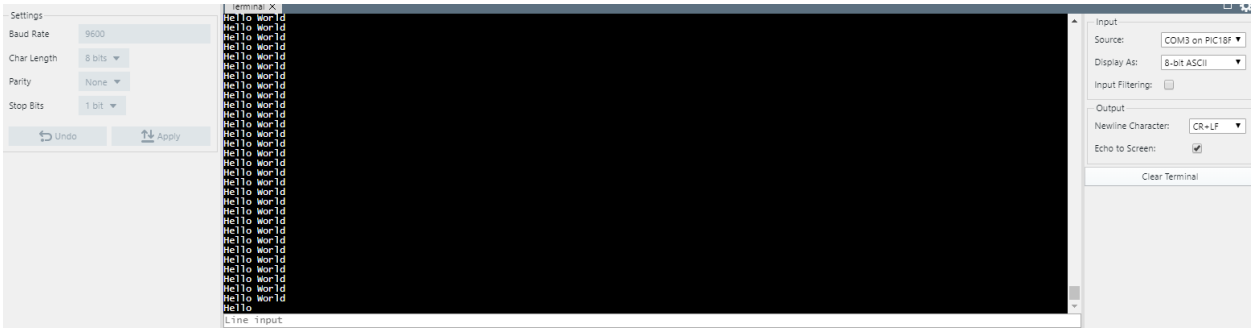
If using a platform with on-board UART to USB, no further hardware setup is needed. If not, connect the TX pin to the RX pin of a UART to USB converter and connect that to the PC.

Open the Data Visualizer by clicking the icon at the top of the MPLAB X IDE.

Figure 6-1. MPLAB® X Data Visualizer Icon

Available COM ports will be listed on the left hand side of the screen. Find the one associated with the connected board and click the **Play** button. In the terminal window settings (right hand side), select **Input Source** to be the COM port associated with the board. It will start receiving messages from the microcontroller.

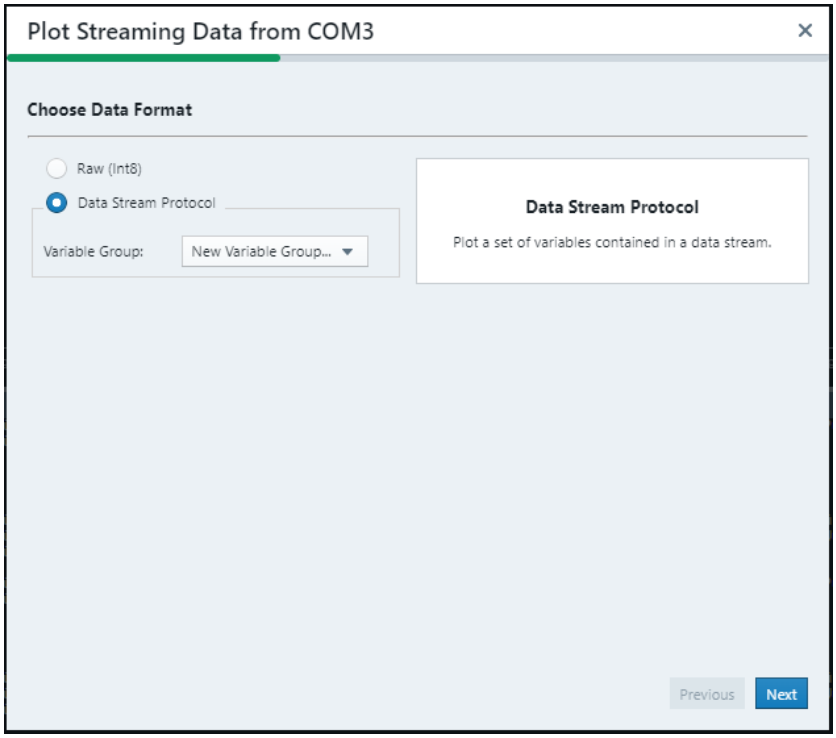
Figure 6-2. MPLAB® X Data Visualizer Receiving Data



6.2 How to Configure MPLAB® X Data Visualizer to Decode Data Stream Protocol

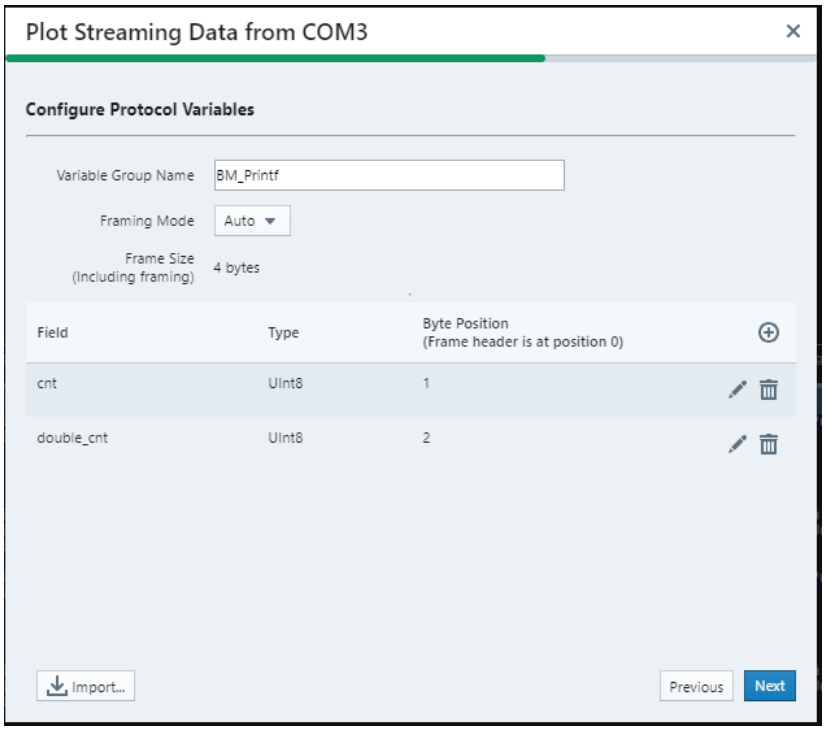
- In Data Visualizer, click the **Plot Data** button and select **Data Stream Protocol**.

Figure 6-3. MPLAB® X Data Visualizer Data Format



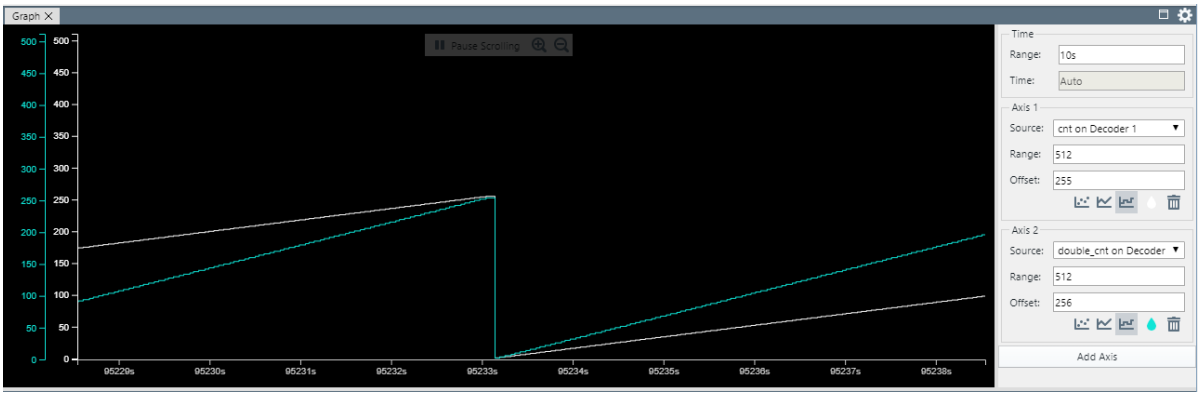
- Click **Next** and in the second screen, configure the frame format for the two fields in the payload.

Figure 6-4. MPLAB® X Data Visualizer Data Stream Protocol Payload Configuration



- Click **Next** and then **Plot**, as the default name for the Decoder works well the way it is generated. As soon as this is done, given the device is correctly programmed with the example code and is connected to the PC, the plots should start showing in real time.

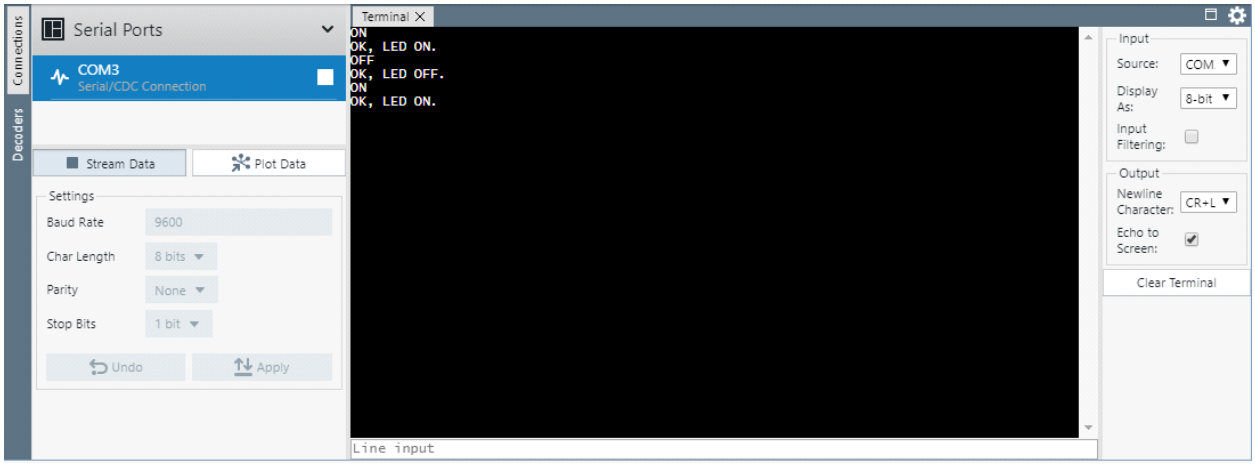
Figure 6-5. MPLAB® X Data Visualizer Displaying Plots



6.3 Send Commands from MPLAB® X Data Visualizer

Program the example code on the device, then open MPLAB® X Data Visualizer. Select the correct COM port as the input source. In the output section, ensure Newline Character is set to ‘CR+LF’. Type one of the commands, ‘ON’ or ‘OFF’, and observe the LED changing its state accordingly.

Figure 6-6. Commands Sent from MPLAB® X Data Visualizer



7. Revision History

Doc Rev.	Date	Comments
A	08/2020	Initial document release

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with

your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

ISBN: 978-1-5224-6528-7

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820