
Adding an ATWINC15x0 Wi-Fi® Network Controller to a Microcontroller Application

Introduction

This document demonstrates the process of porting and adding the ATWINC1500 driver for a desired microcontroller. The ATWINC1500 driver is available in the Microchip Libraries for Applications (MLA).

This document is also useful for the following application users:

- Third party non-Microchip microcontroller
- Microchip (PIC®/AVR®/SAM) microcontroller without MPLAB Harmony Software Framework or Advanced Software Framework (ASF)

Note: Harmony and ASF have their own versions of the ATWINC1500 driver, which can be included using the configuration tools.

The references to the ATWINC15x0 module include the following:

- ATWINC1500
- ATWINC1510

Table of Contents

Introduction.....	1
1. Getting Started.....	3
1.1. Prerequisites.....	3
1.2. Reference Documents.....	3
1.3. Glossary.....	3
2. Locating the ATWINC1500 MLA Driver Files.....	4
3. Overview of ATWINC1500 MLA Driver.....	6
3.1. Features.....	6
4. Supporting Blocks of ATWINC1500 Application.....	7
5. Porting ATWINC1500 MLA Driver.....	8
5.1. Importing MLA Driver Files.....	8
5.2. Microcontroller Specific Settings.....	13
5.3. Reviewing and Updating Necessary Port files.....	21
6. Appendix.....	27
6.1. Pin Mapping on the Explorer 16/32 Board between PIC24F Plug-in-Module and WINC1500 Click Module.....	27
7. Document Revision History.....	28
The Microchip Website.....	29
Product Change Notification Service.....	29
Customer Support.....	29
Microchip Devices Code Protection Feature.....	29
Legal Notice.....	29
Trademarks.....	30
Quality Management System.....	30
Worldwide Sales and Service.....	31

1. Getting Started

1.1 Prerequisites

The following tools are used to verify the ported project.

Note: If the specified tools are not available, study the process of porting the driver section and use it with various combinations of development tools.

1. Software:
 - 1.1. MPLAB® X IDE v4.15
 - 1.2. MPLAB XC16 Compiler v1.33
 - 1.3. MPLAB Code Configurator plug-in v3.55.1
 - 1.4. TeraTerm
2. Hardware:
 - 2.1. Explorer 16/32 Development Board
 - 2.2. 16-bit microcontroller PIC24FJ128GA310 plug-in-module
 - 2.3. WiFi 7 click board™ from MikroElektronika (features the ATWINC1510)
 - 2.4. MPLAB ICD4 programmer/debugger (or ICD3/PICKit3, and so on)

1.2 Reference Documents

1. ATWINC1500 MLA User's Guide
2. WiFi 7 Click Board Schematic
3. Explorer 16/32 Development Board User's Guide
4. PIC24FJ128GA310 Plug-in Module Manual

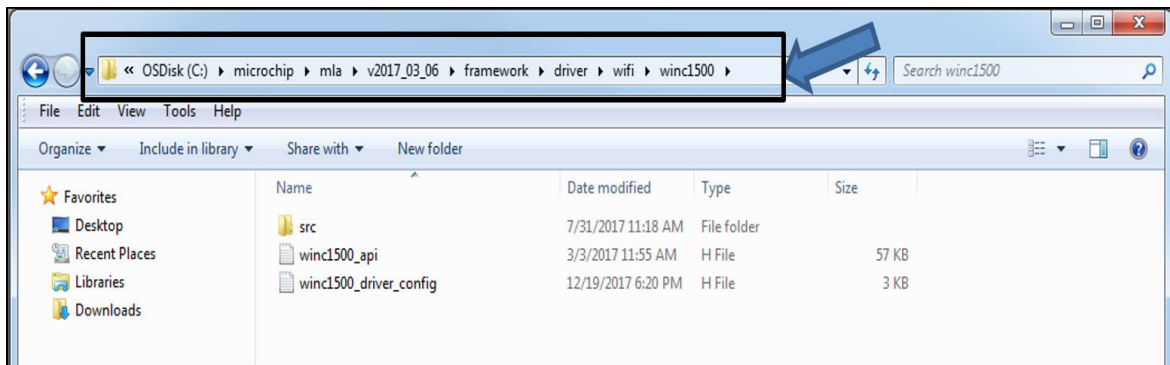
1.3 Glossary

- ASF - Advanced Software Framework
- Explorer - 16/32 Development board is used to run this demonstration. It is sometimes referred as Explorer in this document
- GPIO - General Purpose Input/Output (pins)
- Harmony - Shorthand for MPLAB Harmony Software Framework
- INT1 - Interrupt Module 1
- IRQ - Interrupt Request
- ISR - Interrupt Service Routine
- MCC - MPLAB Code Configurator
- MCU - Microcontroller Unit
- MLA - Microchip Libraries for Applications
- PIM - Plug-in Module
- WINC - Wi-Fi Network Controller

2. Locating the ATWINC1500 MLA Driver Files

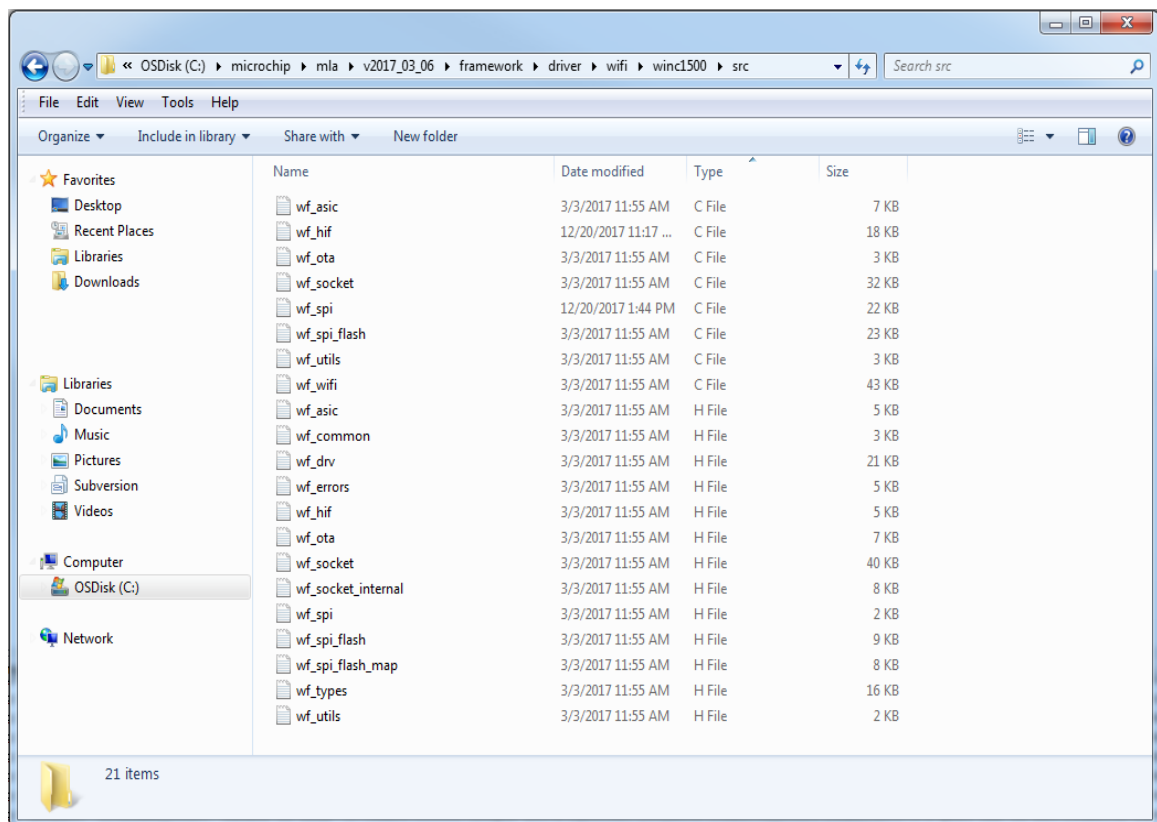
1. Install the MLA framework. See this [link](#) for the latest MLA.
2. Locate the ATWINC1500 driver at: {install path}\microchip\mla\{MLA version}\framework\driver\wifi\winc1500
3. The directory structure for the ATWINC1500 driver with default installation of MLA v2017_03_06 is shown in the following figure.

Figure 2-1. WINC Driver Location in MLA



4. The file `winc1500_api.h` contains declarations for all core driver and support stub functions. Any source file that needs to call the winc1500 driver functions should include this header file.
5. The file `winc1500_driver_config.h` contains configuration macros to customize the driver.
Note: The user refers to this file while porting the driver into their project.

Figure 2-2. WINC Core Driver SRC Files



6. The 'src' folder contains the core driver files as shown in the above figure and should not be modified.

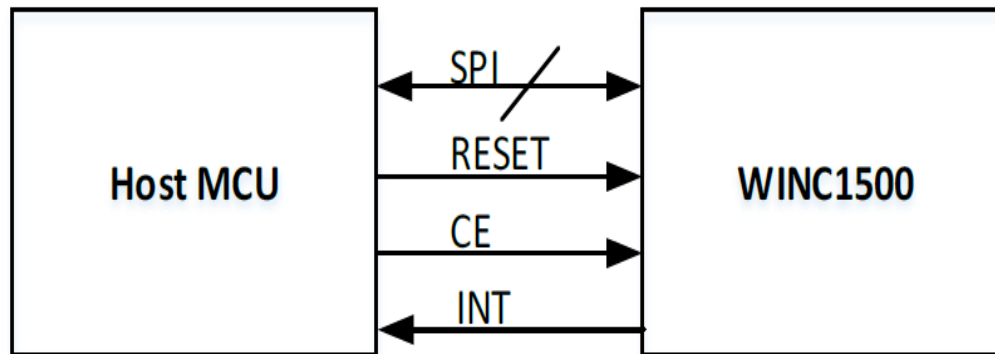
Note: The user can choose either of the following while creating projects for the applications:

1. Copy the winc1500 driver folder to every project, or,
2. The user may point all projects to this common location.

3. Overview of ATWINC1500 MLA Driver

The ATWINC1500 MLA driver helps the user to create 802.11 wireless applications using the ATWINC1500 module. The MLA driver runs on a host MCU that connects to the ATWINC1500, as shown in the following figure.

Figure 3-1. Module Host Interface



3.1 Features

Following are the features of the ATWINC1500 MLA Driver:

- MCU-agnostic; requires only an SPI interface, a timer, 2 GPIO's, and an interrupt line
- Customizable via compiler switches to save memory
- Written in portable 'C' with all source code provided

4. Supporting Blocks of ATWINC1500 Application

Table 4-1. Major Blocks in an ATWINC1500 Application

Block	Written / Edited by User?	Description
Applications software	Yes	This is the application code. The user must also write callback functions for the ATWINC1500 event handlers (For example: Wi-Fi event, Socket event, and so on) that the driver calls, and the application must process.
ATWINC1500 MLA driver software	No	This is the core driver code supplied by Microchip. It should not need to be modified.
Stub software	Yes	The MLA driver calls these functions, but they must be coded by the user. The stub functions control MCU-specific hardware and event handling: <ul style="list-style-type: none"> • SPI Interface • GPIO control • 1ms Timer • Interrupt from ATWINC1500 • Wi-Fi, Socket, Error and OTA event handling

Note: It is highly recommended to read first five sections of the ATWINC1500 MLA user's guide located at C:\microchip\mla\v2017_03_06\apps\tcpip\wifi_winc1500_demo\docs\WINC1500 MLA User's Guide.pdf.

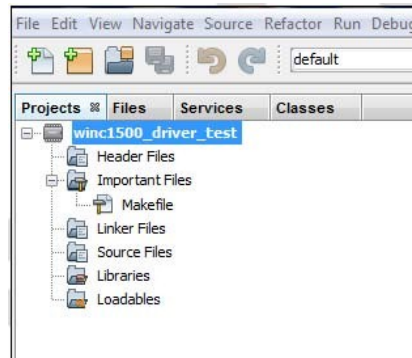
5. Porting ATWINC1500 MLA Driver

Start the process of porting and integrating the ATWINC1500 MLA driver to the project. Before the ATWINC1500 Driver API can be used, there are some MCU-specific stub functions that must be coded. The ATWINC1500 driver calls these functions during run-time. The examples shown here use MPLAB X IDE. If you are using a non-Microchip device and a different IDE, review the steps outlined below and follow the IDE's process to achieve a similar result.

5.1 Importing MLA Driver Files

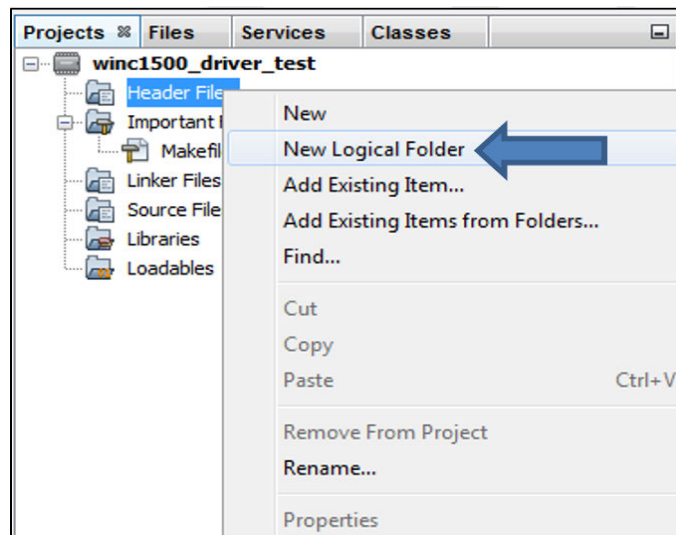
1. Create a project with PIC24FJ128GA310 in MPLAB X IDE, the latest XC16 compiler and programmer tool.

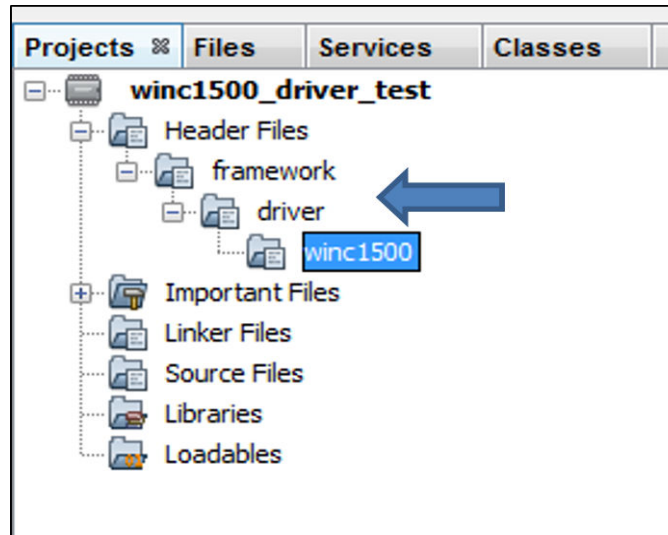
Figure 5-1. Creating the Project



2. Right click on "Header Files" and create a New Logical Folder named "framework". Similarly, create logical folders as shown in the following figure, such as: Header Files -> framework -> driver -> winc1500.

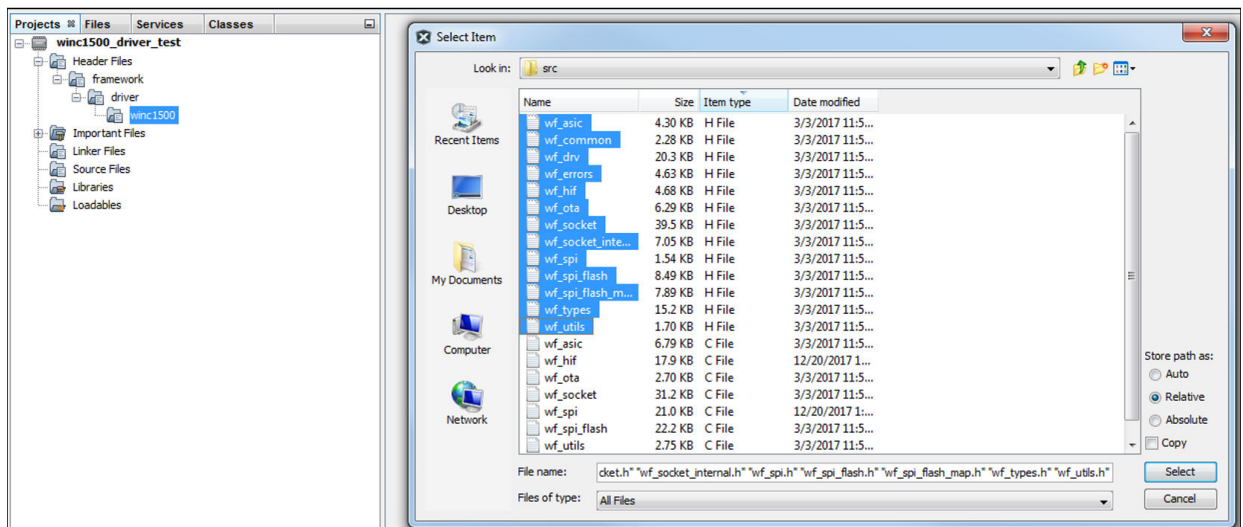
Figure 5-2. New Logical Folder Hierarchy for ATWINC1500 Driver





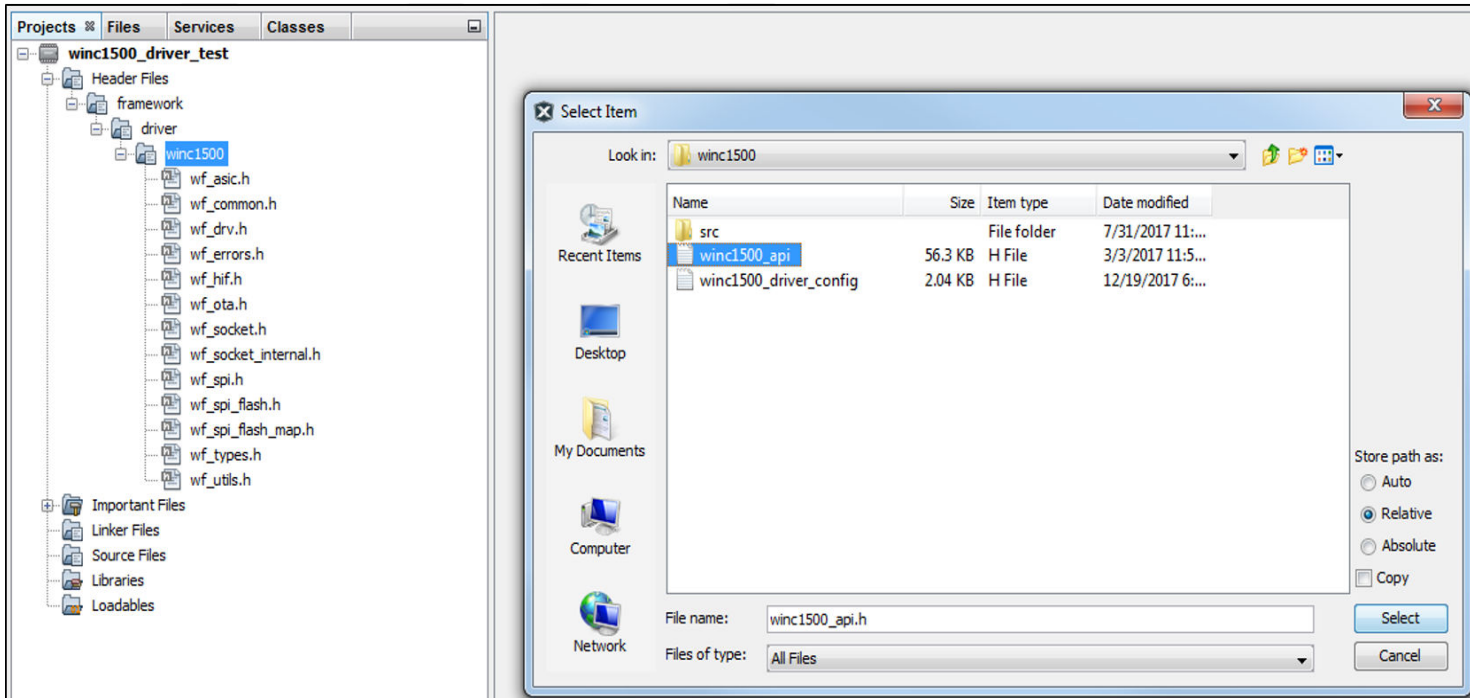
3. Right click on newly created logical “winc1500” folder and select “Add Existing Item”. In the browse window, select all the header files from the “src” folder and click Select. Select the storage path as “Auto”. Following are the two options for this:
 - 3.1. Copy the WINC1500 folder from the MLA driver into your project directory and browse to local location: `\winc1500_driver_test.X\winc1500\src\`.
 - 3.2. Browse to the WINC1500 MLA driver location. This way, multiple projects point to the same driver code instead of having individual driver copy for each project: `mla\{MLA version}\framework\driver\wifi\winc1500\src\`.

Figure 5-3. Adding Driver Header Files



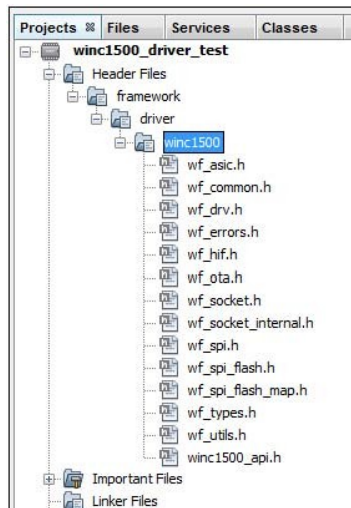
4. Add one more header file to this logical winc1500 folder. Follow same process as above and navigate to the file `winc1500_api.h` and add it.

Figure 5-4. Adding One More Driver Header Files



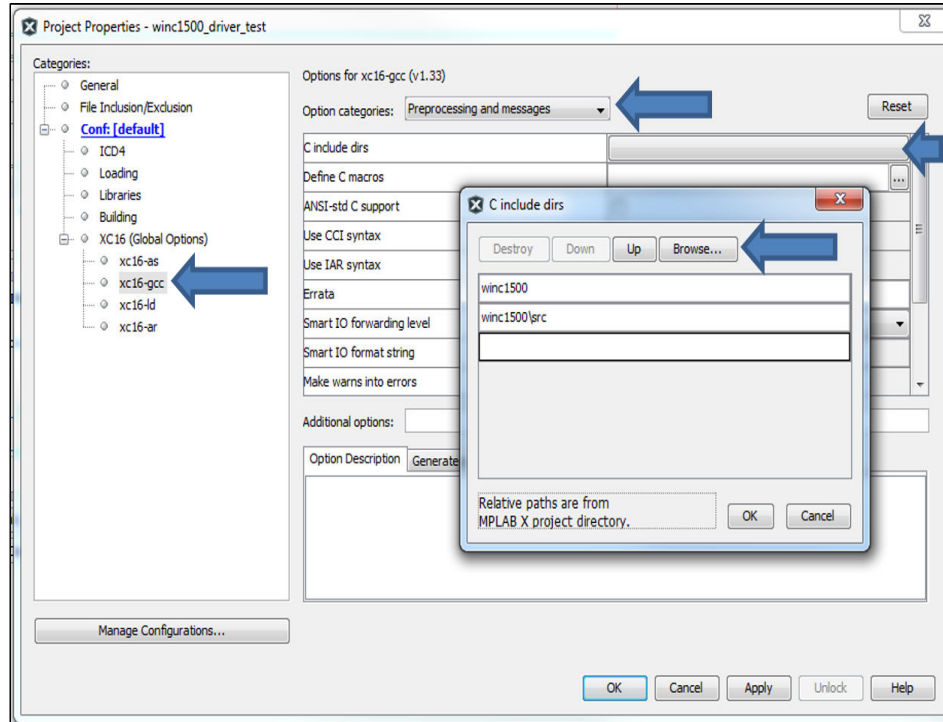
The header file structure is shown in the following figure:

Figure 5-5. Header File Structure



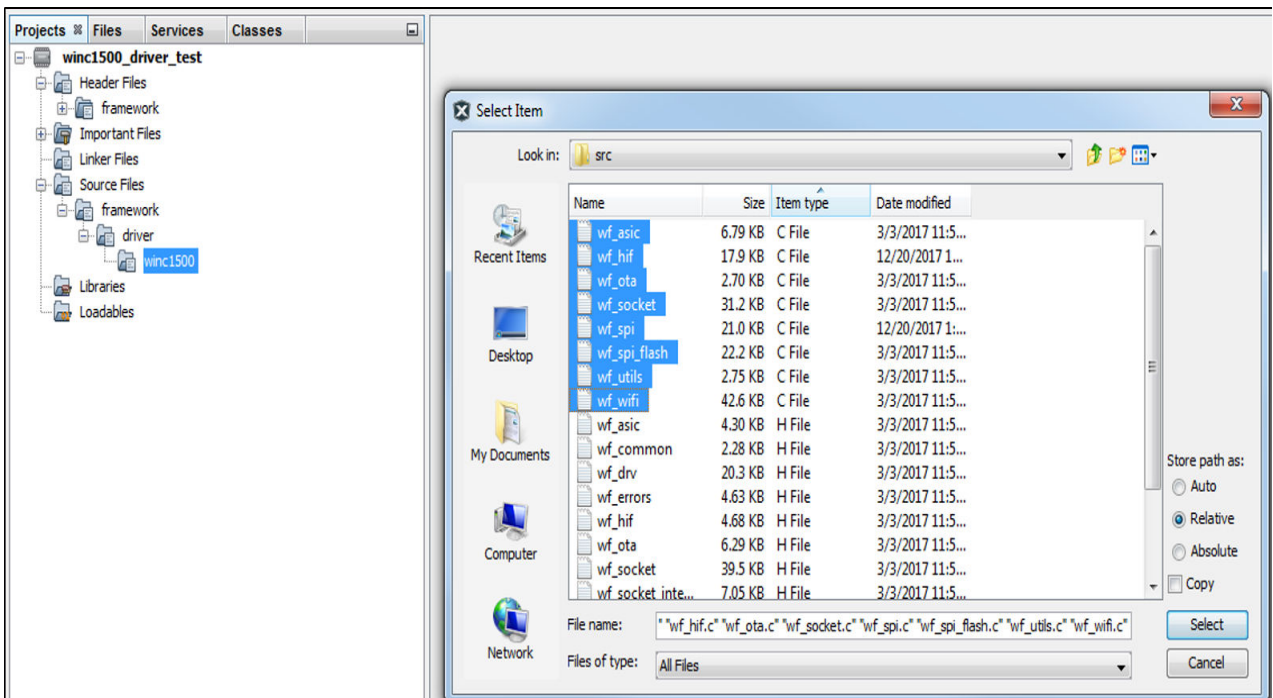
- Now, add the header paths to project properties. Go to Project Properties ->xc16-gcc ->Option dropdown: "Preprocessing and messages" ->C include dirs -> browse and add driver paths based on your choice from step 3. The following figure shows the paths for the local copy of the driver.

Figure 5-6. Adding Library Path



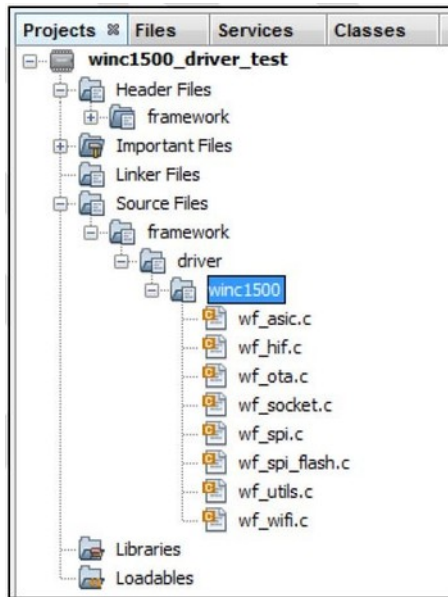
6. Similarly, create the logical folder structure under “Source Files” and add WINC1500 driver source ‘C’ files.

Figure 5-7. Adding Driver C File



The source file structure is shown in the following figure:

Figure 5-8. Source File Structure



7. Create a logical folder "config" under Header Files and add the winc1500_driver_config.h file.
Note: If you are pointing to MLA driver files (step 3 choice 2), you need to create a local copy of the file winc1500_driver_config.h into the project and add the local copy to the project.

Figure 5-9. Adding Driver Configuration Header File

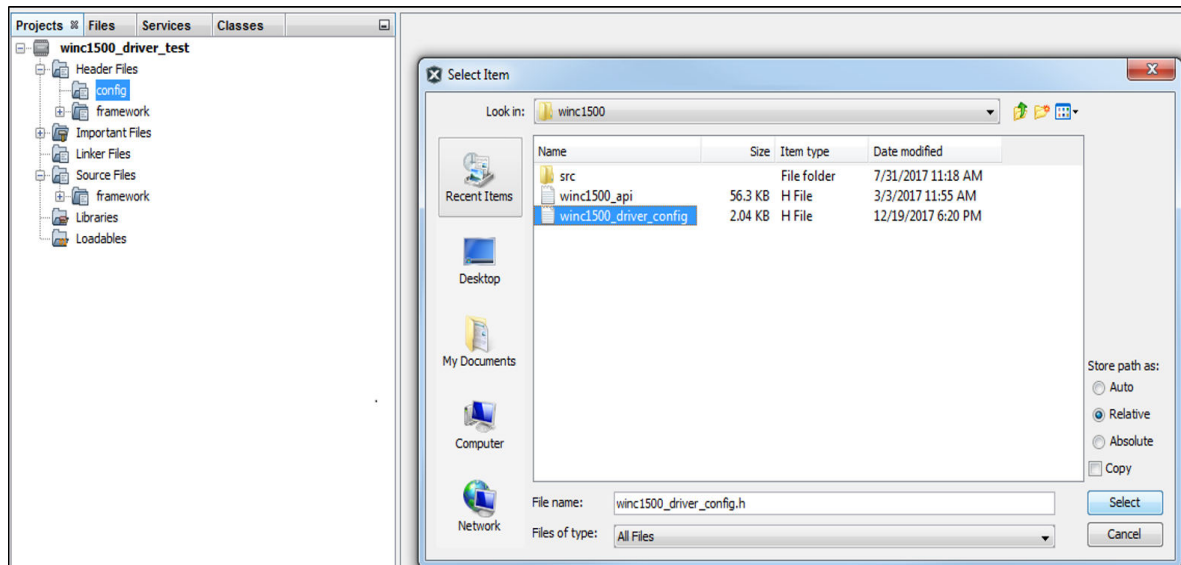
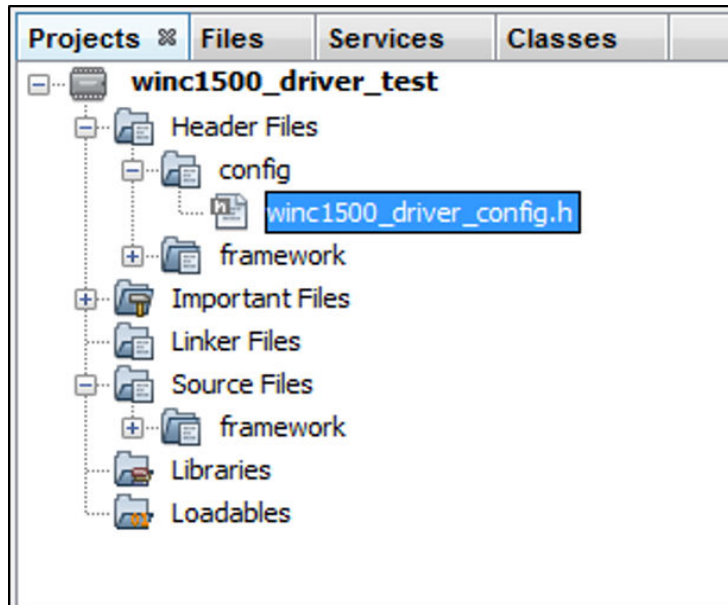


Figure 5-10. Driver Configuration Header File



5.2 Microcontroller Specific Settings

This section is highly dependent on the target microcontroller. Be sure to complete the following steps with respect to the device you are using. In short, this is where to configure the microcontroller's clock source, GPIO, peripherals, console services, and so on.

For PIC Microcontroller users, use the MPLAB Code Configurator (MCC) plugin in MPLAB X IDE for easy configuration and streamlined code generation. Refer to the online tutorial (<http://microchipdeveloper.com/install:mcc>) to learn more about MCC. Check the release notes for MCC (<https://www.microchip.com/mplab/mplab-code-configurator>) to know if your device is supported by MCC.

The user can also perform this without MCC. Ensure that the basic system settings such as configuration bits, UART, SPI are go through the following settings and write the code accordingly.

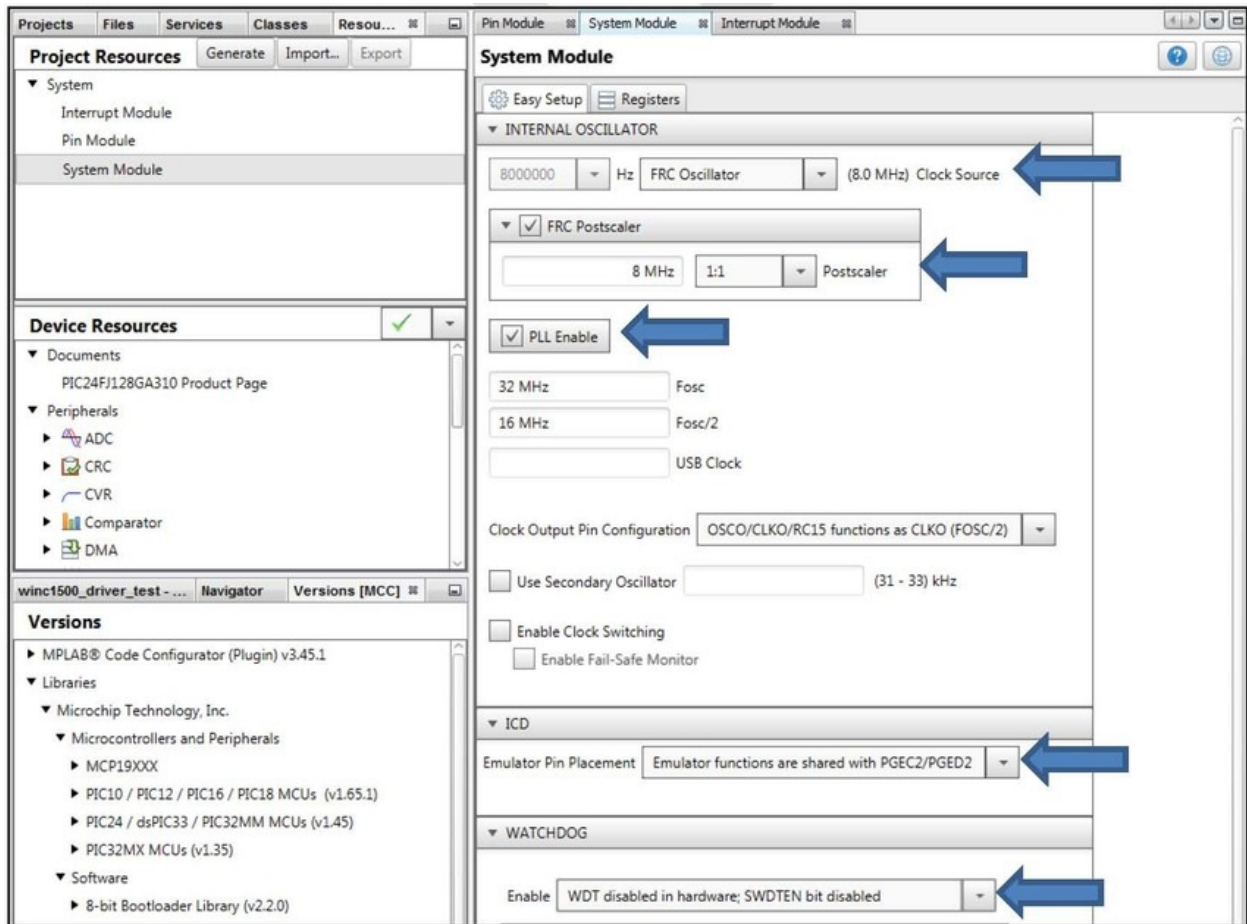
5.2.1 Generic System Settings

Configure your clock source, programmer settings, watchdog timer settings, and so on to following settings.

MCC users must update the following in MCC system module:

- 8 MHz FRC Oscillator with FRC postscaler 1:1
- Enable PLL to obtain system frequency (= $F_{osc}/2$) of 16 MHz
- Debugger uses EMUC2/EMUD2 pins (PGCx2/PGDx2)
- Disable Watchdog Timer

Figure 5-11. Generic System Settings



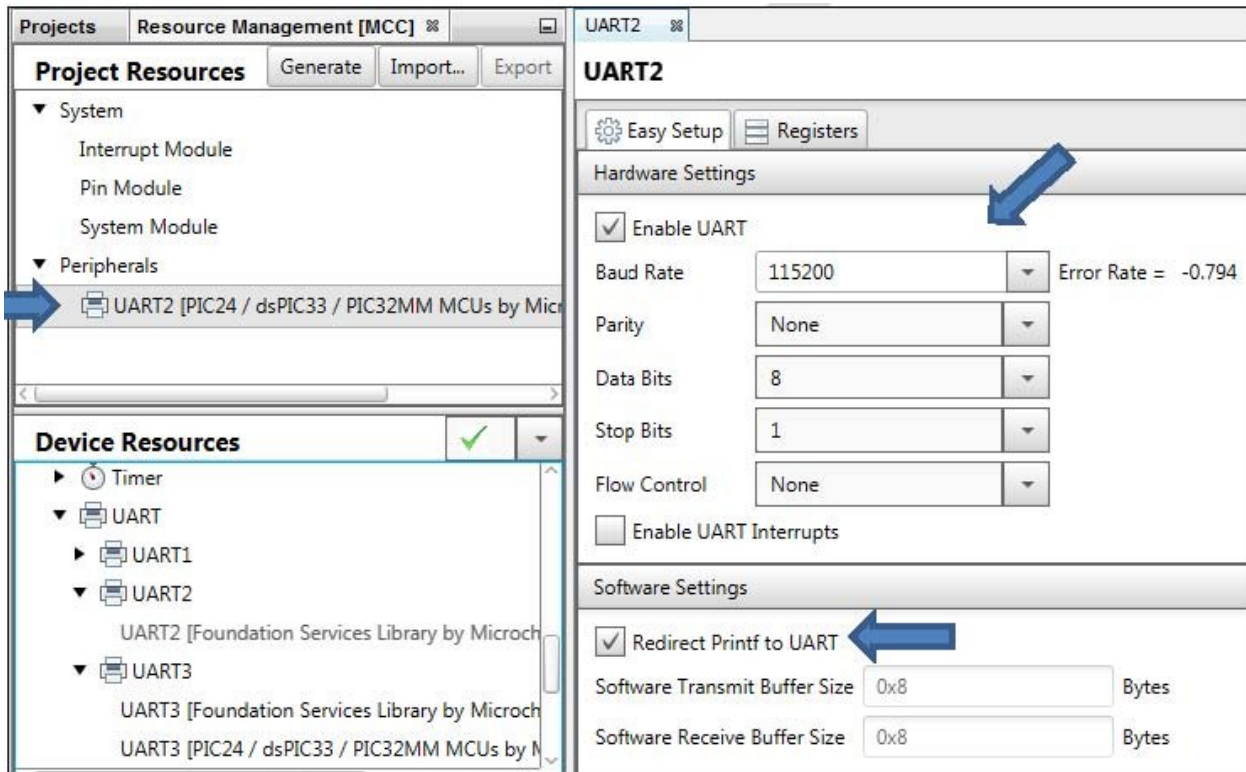
5.2.2 Console Service

Use UART2 to print messages to the serial terminal using the `printf` function. Ensure `printf` is configured to send data to the serial output. The Explorer 16 board features a UART communication interface connected to PIC24's UART2 module (see the Explorer 16/32 board's user's guide).

For MCC Users:

- Select "UART2" module (not foundation services) from device resources and double-click to add it to project resources
- Configure "UART2" as 115200 | None | 8 | 1
- Deselect "Enable UART Interrupts" checkbox
- Select "Redirect `printf` to UART"

Figure 5-12. UART2 Settings



- Use the MCC pin manager grid view to lock (assign) the pins RF4 and RF5 as U2RX and U2TX resp as shown in the following figure:

Figure 5-13. UART2 Pin Settings

Pin Manager: Grid View			Notifications [MCC]																														
Package:	TQFP100	Pin No:	-	-	-	-	-	-	87	88	52	51	49	50	55	54	53	40	39	-													
			A					B					C					D					E					Port F					G
Module	Function	Direction	-	-	-	-	-	-	0	1	2	3	4	5	6	7	8	12	13	-													
UART2	U2CTS	input									🔒	🔒	🔒	🔒			🔒	🔒	🔒														
	U2RTS	output									🔒	🔒	🔒	🔒			🔒		🔒														
	U2RX	input									🔒	🔒	🔒	🔒			🔒	🔒	🔒														
	U2TX	output									🔒	🔒	🔒	🔒			🔒		🔒														

5.2.3 External Interrupt for ATWINC1500

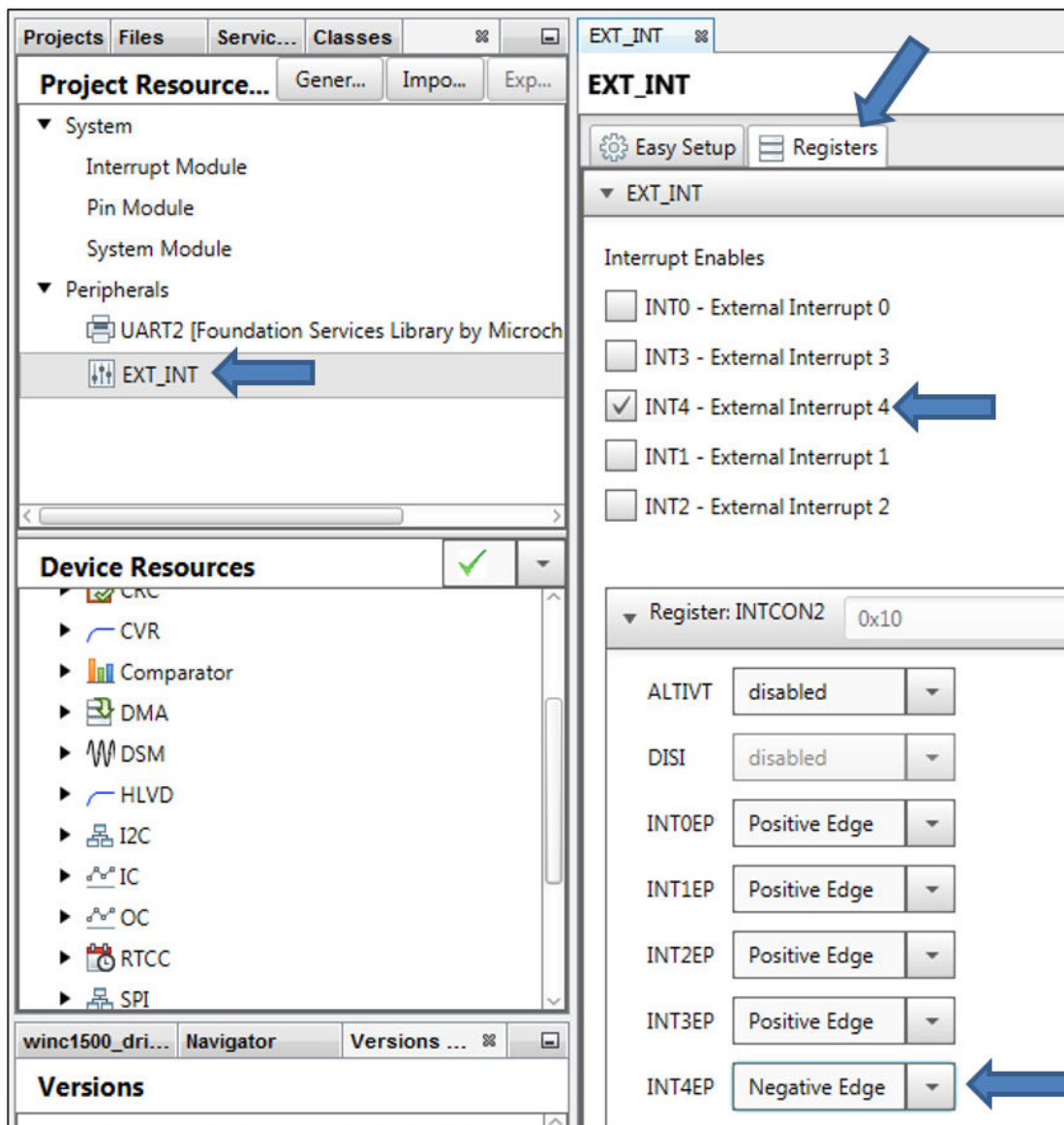
The ATWINC1500 interrupts the host MCU when events occur by setting the IRQN line low. The host MCU must be configured to trigger an interrupt on a falling edge.

Use INT4 since the ATWINC1500 is connected to PIC24's INT4 module (see [6.1 Pin Mapping on the Explorer 16/32 Board between PIC24F Plug-in-Module and WINC1500 Click Module](#)).

For MCC Users:

- Select the **EXT_INT** from Device Resources and double-click to add it to Project Resources
- Switch to the **Registers** tab and select only INT4 (disable others)
- Select **Negative Edge** in INT4EP register (ignore **Easy Setup** tab, even if it shows INT0 selected)

Figure 5-14. External Interrupt Settings



- Use MCC pin manager grid view to lock (assign) pin **RA15** as INT4 as shown in the following figure:

Figure 5-15. External Interrupt Pin Settings

Output	Notifications [MCC]	Pin Manager: Grid View																		
Package:	TQFP100	Pin No:	17	38	58	59	60	61	91	92	28	29	66	67	-	-	-	-	-	-
			Port A ▼												B ▶	C ▶	D ▶	E ▶	F ▶	G ▶
Module	Function	Direction	0	1	2	3	4	5	6	7	9	10	14	15						
EXT_INT ▼	INT0	input																		
	INT1	input																		
	INT2	input																		
	INT3	input																		
	INT4	input																		

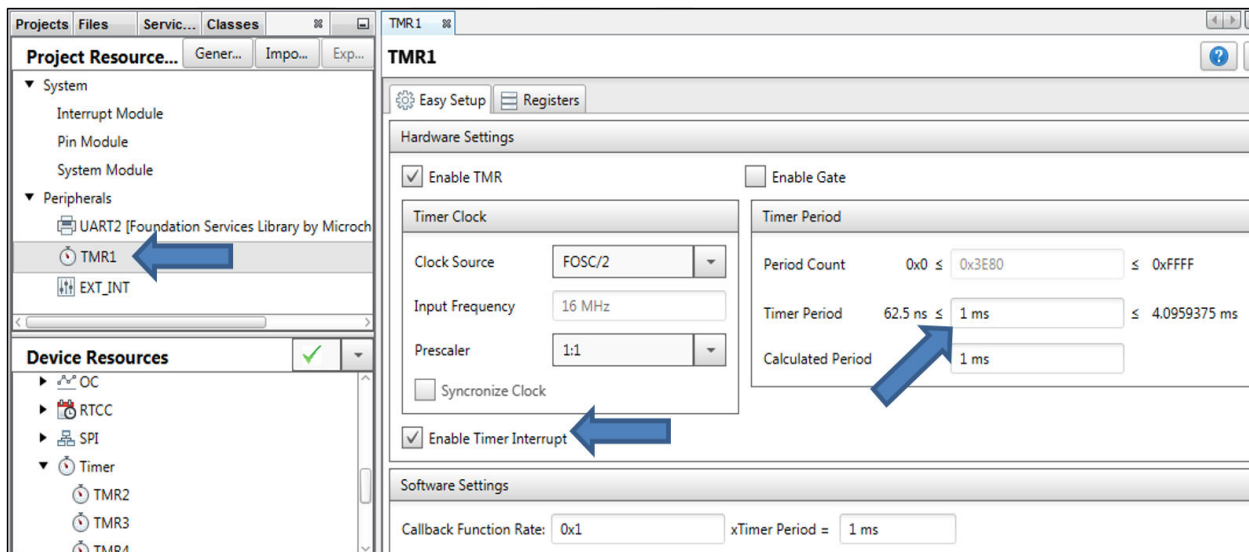
5.2.4 Timer Module

The ATWINC1500 state machines require a timer with one millisecond resolution. The timer is a 32-bit counter that counts up starting at $0x00000000$ and wraps back to 0 after reaching $0xffffffff$. Use TMR1 peripheral to interrupt every 1ms.

For MCC Users:

- Select **TMR1** from device resources and add it to project resources by double-clicking
- Select **Enable TMR**
- Select **Clock Source** as $Fosc/2$ (= CPU Instruction clock)
- **Prescaler** is 1:1
- Enter **Timer Period** value of 1 ms
- Select **Enable Timer Interrupt**

Figure 5-16. TMR1 Settings



5.2.5 SPI Module for ATWINC1500 Interface

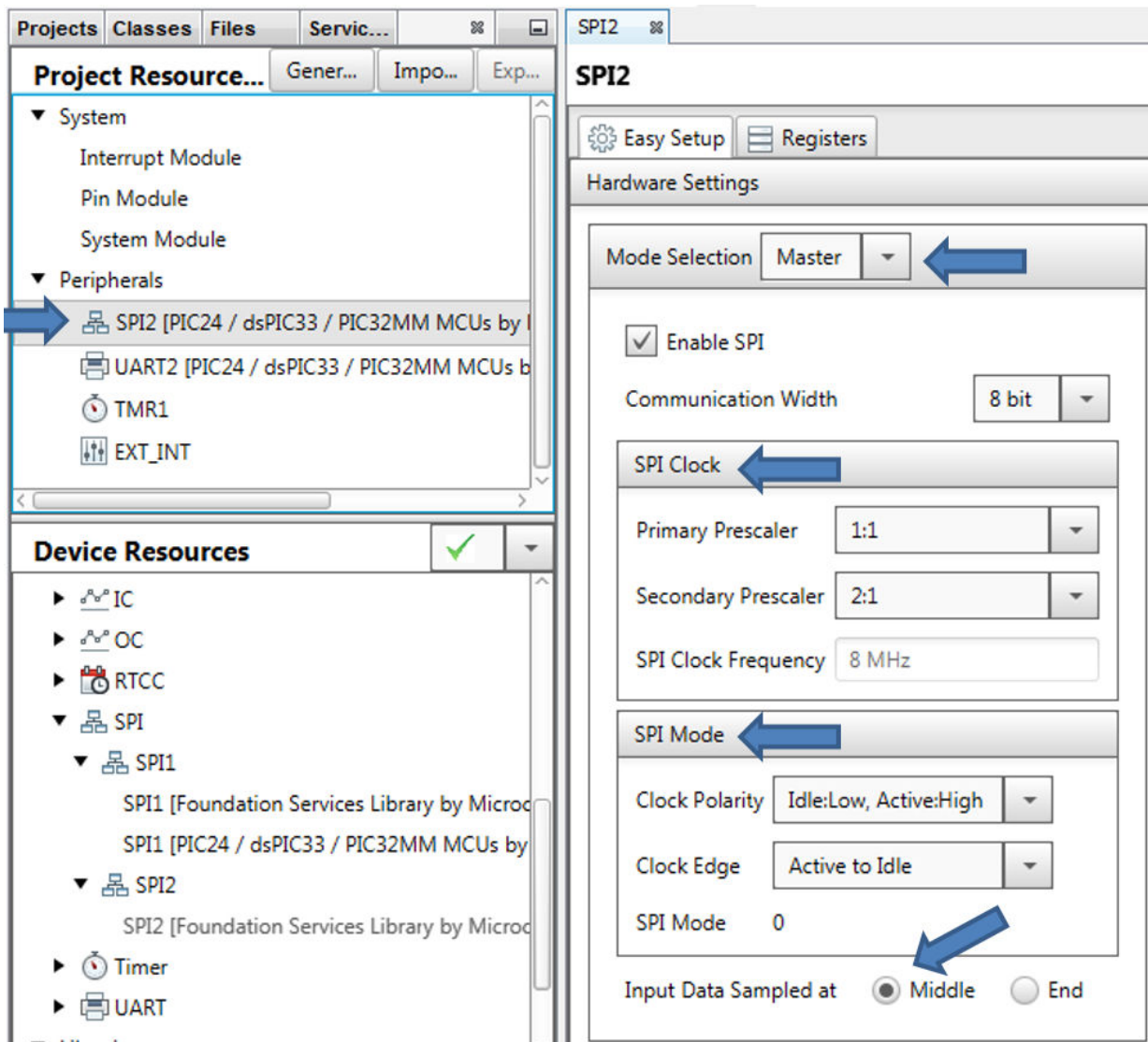
The host MCU communicates to the ATWINC1500 via the SPI interface. SPI mode must be set to mode 0 and SPI clock should be 8 MHz or less.

Use SPI2 since the ATWINC1500 is connected to PIC24's SPI2 module. (See [6.1 Pin Mapping on the Explorer 16/32 Board between PIC24F Plug-in-Module and WINC1500 Click Module](#)).

For MCC Users:

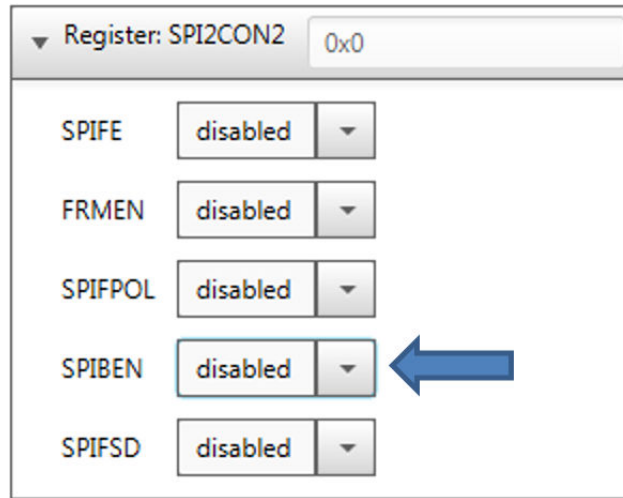
- Add SPI2 driver to Project Resources and set **Mode Selection** as Master
- Select **Enable SPI**
- Set **Communication Width** to 8-bit
- Set **Primary Prescaler** as 1:1 and **Secondary Prescaler** as 2:1 to get **SPI Clock Frequency** of 8 MHz
- Set **SPI Clock Polarity** as Idle:Low, Active:High and Clock Edge as "Active to Idle" to get mode 0
- Select **Input Data Sampling** "Middle" radio button

Figure 5-17. SPI2 Settings



- Go to the **Registers** tab and disable the Enhanced Buffer mode (SPIBEN) to allocate SPI to the ATWINC1500 driver in Standard Buffer mode (to process the SPI data as soon as it is received whereas Enhanced mode uses FIFO buffers).

Figure 5-18. SPI2 Register Settings



- Use MCC pin manager grid view to lock (assign) pin **RG6** as SCK2, **RG7** as SDI2, and **RG8** as SDO2 as shown in the following figure. It is not required to assign SS2; use the GPIO pin for this in the next step.

Figure 5-19. SPI2 Pin Settings

Output	Notifications [MCC]	Pin Manager: Grid View	Pin No: - - - - - 90 89 57 56 10 11 12 14 96 97 95 1																	
Package:	TQFP100		A B C D E F													Port G				
Module	Function	Direction	-	-	-	-	-	-	0	1	2	3	6	7	8	9	12	13	14	15
SPI2	SCK2OUT	output											🔒	🔒	🔒	🔒				
	SDI2	input											🔒	🔒	🔒	🔒				
	SDO2	output											🔒	🔒	🔒	🔒				
	SS2OUT	output											🔒	🔒	🔒	🔒				

5.2.6 GPIO Pins for ATWINC1500 Reset, Enable and Chip Select

The ATWINC1500 driver needs to control Chip Enable, Reset and SPI Slave Select GPIO outputs of the ATWINC1500. The GPIOs described in this section must be configured as outputs and defaulted low prior to the ATWINC1500 driver running.

Set **RG9** (ATWINC1500 Slave Select), **RG14** (ATWINC1500 Reset) and **RD0** (ATWINC1500 Chip Enable) as GPIO outputs since the ATWINC1500 module connects to these pins on the Explorer 16/32 board. (Refer to [6.1 Pin Mapping on the Explorer 16/32 Board between PIC24F Plug-in-Module and WINC1500 Click Module](#)).

For MCC Users:

Use MCC pin manager grid view to lock (assign) above GPIO pins to output as shown in the following figure:

Figure 5-20. GPIO Pin Settings

Output	Notifications [MCC]	Pin Manager: Grid View	Pin No: - - - 72 76 77 78 81 82 83 84 68 69 70 71 79 80 47 48 - - 90 89 57 56 10 11 12 14 96 97 95 1																														
Package:	TQFP100		A B C													Port D			E F			Port G											
Module	Function	Direction	-	-	-	0	1	3	4	5	6	7	8	9	10	11	12	13	14	15	-	-	0	1	2	3	7	8	9	12	13	14	15
Pin Module	GPIO	input				🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	
	GPIO	output				🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	🔒	

5.2.7 Checkpoints for Pin Assignments

For MCC Users:

- Select **Pin Module** from MCC Project Resources

- Ensure assigned pins are digital (deselect any analog boxes enabled by default)
- Select the **Start High** check box for UART transmit pin RF5 (needed for correct start bit sequence)
- Give custom names to GPIO pins as shown in the following figure (**IO_RD0_CE**, **IO_RG9_SS2**, and **IO_RG14_RESET**).

The following figure shows a quick summary of used pins.

Figure 5-21. Pin Module Settings

Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	WPD	OD	IOC
RF4	UART2	U2RX		<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RF5	UART2	U2TX		<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RG6	SPI2	SCK2OUT	SCK2OUT	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RG7	SPI2	SDI2	SDI2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RG8	SPI2	SDO2	SDO2	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RD0	Pin Module	GPIO	IO_RD0_CE	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RG9	Pin Module	GPIO	IO_RG9_SS2	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RG14	Pin Module	GPIO	IO_RG14_RESET	<input type="checkbox"/>		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RB6	ICD	PGC2		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RB7	ICD	PGD2		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RA15	EXT_INT	INT4		<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none

- If any pins are missing, go back to the MCC pin manager grid view to lock (assign) those pins
- Refer to [6.1 Pin Mapping on the Explorer 16/32 Board between PIC24F Plug-in-Module and WINC1500 Click Module](#) for pin mapping between PIC24FJ128GA310 plug-in module and the ATWINC1500 click board on the Explorer 16/32 board

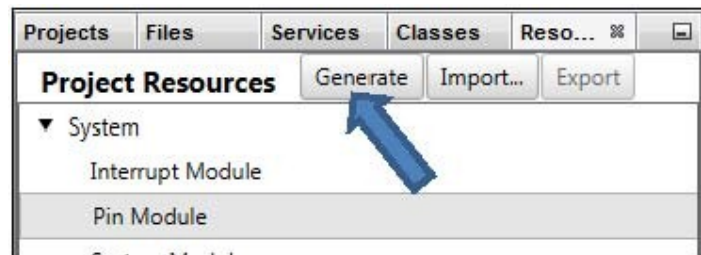
5.2.8 Generate Code

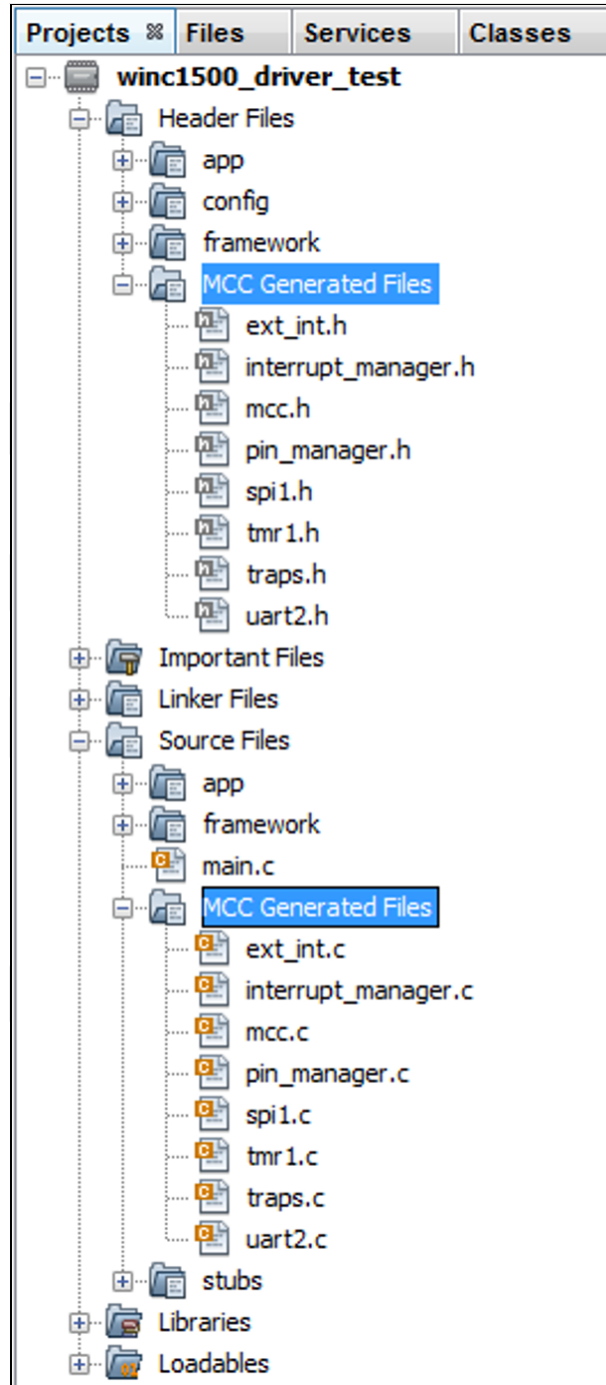
Click the **Generate** button in MCC to generate source code for system settings and peripheral drivers as configured. You can make use of the driver API during the update of sample port files.

Note:

1. If you get a MCC warning to turn on "clock-switching", you can turn it on by going to system module.
2. MCC generates a `main.c` file for the user.

Figure 5-22. Generate Code Settings





5.3 Reviewing and Updating Necessary Port files

The porting process involves the following tasks:

1. Customizing the ATWINC1500 driver to the target microcontroller architecture (file: `winc1500_driver_config.h`)
2. Implementing stub functions (GPIO control, SPI transaction, Interrupt handler, and so on.)
3. Implementing event callbacks (Wi-Fi, socket, error, and so on.)

This application note provides code snippets from MLA demo `winc1500_pic24fj128ga310_ex16.X`.

5.3.1 Customizing the ATWINC1500 Driver

The header file `winc1500_driver_config.h` contains configuration macros to customize the ATWINC1500 driver for the microcontroller. It also contains macros to enable/disable features in the ATWINC1500 driver.

The following is the summary of recommended changes:

1. Set Endianness in MCU. If MCU is big-endian, un-comment the macro `HOST_MCU_BIG_ENDIAN`.
2. Set the size of host MCU pointer in bytes using macro `M2M_POINTER_SIZE_IN_BYTES`. Few examples for existing PIC microcontrollers are already defined. Define the pointer size if a different microcontroller is used.
3. Enable/disable desired features.
4. Enable/disable ATWINC1500 debug log (keep default setting, disabled for efficiency).

Note: Ignore the macro `USING_PICTAIL` and related code at the end. This is included for demos in MLA.

5.3.2 Implementing Stub Functions

Sample implementations are available for reference in the demo file `wf_mcu_driver_stub.c` (`C:\microchip\mla\v2017_03_06\apps\tcpip\wifi_winc1500_demo\winc1500_pic24fj128ga310_ex16.X\`). The MLA driver calls these functions internally, and they must be coded by the user. The stub functions control microcontroller-specific hardware and event handling.

1. GPIO stub functions:

The ATWINC1500 driver needs to control three GPIO outputs to the ATWINC1500 connected to ATWINC1500 Chip Enable, Reset and SPI Slave Select. The core driver calls these functions to set these pins high or low. The user can use the MCC generated functions (based on custom names given previously) to handle GPIO high/low commands:

```
void m2mStub_PinSet_CE(t_m2mWifiPinAction action)
{
    if (action == M2M_WIFI_PIN_LOW)
    {
        IO_RD0_CE_SetLow();
    }
    else
    {
        IO_RD0_CE_SetHigh();
    }
}
```

```
void m2mStub_PinSet_RESET(t_m2mWifiPinAction action)
{
    if (action == M2M_WIFI_PIN_LOW)
    {
        IO_RG14_RESET_SetLow();
    }
    else
    {
        IO_RG14_RESET_SetHigh();
    }
}
```

```
void m2mStub_PinSet_SPI_SS(t_m2mWifiPinAction action)
{
    if (action == M2M_WIFI_PIN_LOW)
    {
        IO_RG9_SS2_SetLow();
    }
    else
    {
        IO_RG9_SS2_SetHigh();
    }
}
```

2. Interrupt stub functions:

The core driver calls these functions to enable/disable MCU interrupt when necessary.

```
void m2mStub_EintEnable(void)
{
    EX_INT4_InterruptEnable();
}
```

```

}

void m2mStub_EintDisable(void)
{
    EX_INT4_InterruptDisable();
}

```

Also, define external interrupt handler for Host MCU. The Host MCU interrupt handler must call `m2m_EintHandler()` each time when a ATWINC1500 interrupt occurs, and then clear the interrupt.

For MCC Users:

The user modifies the `ext_int.c` file to include `m2m_EintHandler()` in the `_INT1Interrupt` ISR.

Note: To use `m2m_EintHandler()` function, the user needs to include `winc1500_api.h` in `ext_int.c` file.

```

#include "winc1500_api.h"

/**User Area Begin->code: Add External Interrupt handler specific headers
extern void m2m_EintHandler(void);
/**User Area End->code: Add External Interrupt handler specific headers

void __attribute__ ( ( interrupt, no_auto_psv ) ) _INT4Interrupt(void)
{
    /**User Area Begin -> code: INT4 - External Interrupt 4**
    m2m_EintHandler();
    /**User Area End -> code: INT4 - External Interrupt 4**
    EX_INT4_InterruptFlagClear();
}

```

3. Timer stub function:

The ATWINC1500 state machines require a timer with one millisecond resolution. The timer is a 32-bit counter that counts up starting at `0x00000000` and wraps back to 0 after reaching `0xffffffff`. Implement "Atomic" function to return 1ms counter.

For MCC Users:

The `TMR1_CallBack()` function is already called in timer ISR. The user can use this function to update counter. The following code is implemented in `tmr1.c`.

```

uint32_t m2mStub_GetOneMsTimer(void)
{
    return GetOneMsCounter();    //function added to MCC generated timer1 code
}

//added after MCC-generated code
static volatile uint32_t g_oneMsCounter = 0;

uint32_t GetOneMsCounter(void)
{
    uint32_t tmp;

    IEC0bits.T1IE = false; // disable timer interrupt
    tmp = g_oneMsCounter; // get a clean copy of counter variable
    IEC0bits.T1IE = true; // enable timer interrupt

    return tmp;
}

```

4. SPI stub function:

Implement this function as a one-byte transfer. Do not use Enhanced Buffer mode since the received byte must be captured immediately. The Standard Buffer mode is required because the ATWINC1500 driver waits during the response of this transfer.

For MCC Users:

At present MCC does not provide a standard buffer implementation for one-byte SPI transfer in the Blocking mode. Therefore, implement a blocking function that transfers one byte over SPI and returns one byte received during this transfer.

```
void m2mStub_SpiTxRx(uint8_t *p_txBuf,
                    uint16_t txLen,
                    uint8_t *p_rxBuf,
                    uint16_t rxLen)
{
    uint16_t byteCount;
    uint16_t i;

    /* total number of byte to clock is whichever is larger, txLen or rxLen */
    byteCount = (txLen >= rxLen) ? txLen : rxLen;

    for (i = 0; i < byteCount; ++i)
    {
        /* if still have bytes to transmit from tx buffer */
        if (txLen > 0)
        {
            SPI2BUF = *p_txBuf++;
            --txLen;
        }
        /* else done writing bytes out from tx buffer */
        else
        {
            SPI2BUF = 0x00; /* clock out a "don't care" byte */
        }

        /* wait until tx/rx byte to completely clock out */
        WaitForDataByte();

        /* if still have bytes to read into rx buffer */
        if (rxLen > 0)
        {
            *p_rxBuf++ = SPI2BUF;
            --rxLen;
        }
        /* else done reading bytes into rx buffer */
        else
        {
            SPI2BUF; /* read and throw away byte */
        }
    } /* end for loop */
}
```

5. Event stub functions:

Event stub functions can be implemented using API or callback functions defined in the application. The following is the default code example within `winc1500_pic24fj128ga310_ex16.X` demo – `winc1500_ap_scan`. In this example, the various event handlers are implemented in separate files.

Refer to the following files and event implementations.

- 5.1. **Wi-Fi event handler** – `wifi_event_stub.c`
(C:\microchip\mla\2017_03_06\apps\tcpip\wifi_winc1500_demo\firmware\src\winc1500_ap_scan\wifi_event_stub.c)
- 5.2. **Socket event handler** – `socket_event_stub.c`
(C:\microchip\mla\2017_03_06\apps\tcpip\wifi_winc1500_demo\firmware\src\winc1500_ap_scan\socket_event_stub.c)

The user can implement necessary events in the event stubs. For more details, refer to Section 7.1 Wi-Fi Events and Section 7.2 Socket Events of the *WINC1500 MLA User's Guide*.

Global flags are used by the application state machine and event handlers to capture and communicate the status of various parameters. Getter functions are used to ensure proper tracking of flags.

```
//event flags for demo applications
static bool g_driverInitComplete;
static bool g_scanComplete;
static bool g_scanResultReady;
static bool g_connectionStateChanged;
static bool g_ipAddressAssigned;
```

```

static bool g_rssiReady;
static bool g_provisionInfoReady;
static bool g_wpsReady;
static bool g_prngReady;
void m2m_wifi_handle_events(t_m2nNifiEventType eventCode, t_wifiEventData *p_eventData)
{
    switch (eventCode)
    {
        case M2M_WIFI_DRIVER_INIT_EVENT:
            g_driverInitComplete = true;
            break;

        case M2M_WIFI_CONN_STATE_CHANGED_EVENT:
            // event data in p_eventData->connState
            g_connectionStateChanged = true;
            break;

        case M2M_WIFI_SYS_TIME_EVENT:
            // event data in p_eventData->sysTime
            dprintf(" EVENT: M2M_WIFI_SYS_TIME_EVENT\n");
            break;
    }
}

```

```

static bool g_bindOccurred;
static bool g_listenOccurred;
static bool g_acceptOccurred;
static bool g_connectOccurred;
static bool g_dnsResolved;
static bool g_recv;
static bool g_recvFrom;
static bool g_sendTo;
static bool g_send;
static bool g_pingReply;

static tpfAppSocketCb gpfAppSocketCb = NULL;
static tpfAppResolveCb gpfAppResolveCb = NULL;

void m2m_socket_handle_events(SOCKET sock, t_m2m3ocketEventType eventCode,
t_socketEventData *peventData)
{
    switch (eventCode)
    {
        case M2M_SOCKET_BIND_EVENT:
            // event data in p_eventData->bindStatus
            g_bindOccurred = true;
            break;

        case M2M_SOCKET_LISTEN_EVENT:
            // event data in peventData->listenStatus
            g_listenOccurred = true;
            break;
    }
}

```

6. OTA and error event handler functions:
OTA event handler and error event handler are not implemented.

5.3.3 main.c

This file includes `winc1500_api.h` and other required header files.

Initialize the following applications:

- SPI
- UART drivers
- Clock
- Console services

Start the infinite loop executing application task and winc1500 driver task.

For more details about the Application task, refer to the *WINC1500_MLA_Getting_Started_Guide.pdf* document. It details how to run the AP scan demo on Explorer 16 boards.

```

//=====
// INCLUDES

```

```
//=====
#include "winc1500_api.h" // primary WINC1500 include file
#include "demo_config.h" // selects which demo to run
#include "bsp.h" // defines for LED's and push buttons on board

//=====
// FUNCTION PROTOTYPES
//=====
static void BlinkLed(void);

//=====
// Main application entry point.
//=====
int main void
{
    BspInit();

    #if defined(USING_PICTAIL)
        printf("Project build for PICTail\r\n");
    #elif defined(USING_CLICK_BOARD)
        printf("Project build for Click Board\r\n");
    #endif

    printf("Starting driver initialization...\r\n");
    m2m_wifi_init();

    while (1)
    {
        ApplicationTask();
        m2m_wifi_task();

        BlinkLed();
    }
}
```

The demo application is ready. The user must compile and load the demo application into hardware.

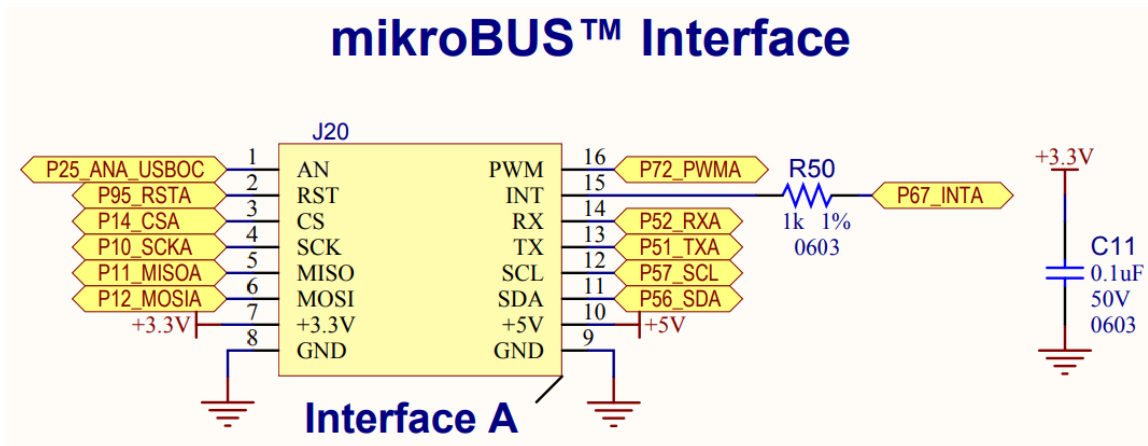
6. Appendix

6.1 Pin Mapping on the Explorer 16/32 Board between PIC24F Plug-in-Module and WINC1500 Click Module

The following points describe the pin mapping:

1. mikroBUS™ Interface on Explorer 16/32 board:
See http://ww1.microchip.com/downloads/en/DeviceDoc/Explorer_16_32_Schematics_R6_3.pdf.

Figure 6-1. Pin Mapping



2. Pinout on WiFi 7 click:
See <https://www.mikroe.com/wifi-7-click>.

The following table shows how the pinout on WiFi 7 click corresponds to the pinout on the mikroBUS socket.

Table 6-1. Pinout on WiFi 7 click

Notes	Pin	mikroBUS™				Pin	Notes
	NC	1	AN	PWM	16	EN	Module Enable
Active-Low Hard Reset	RST	2	RST	INT	15	INT	Device Interrupt Output
Chip Select	CS	3	CS	TX	14	NC	
SPI Clock	SCK	4	SCK	RX	13	NC	
SPI Master Input Slave Output	MISO	5	MISO	SCL	12	NC	
SPI Master Output Slave Input	MOSI	6	MOSI	SDA	11	NC	
Power Supply	+3.3V	7	3.3V	5V	10	NC	
Ground	GND	8	GND	GND	9	GND	Ground

7. Document Revision History

Revision	Date	Section	Description
B	07/2019	5.2.6 GPIO Pins for ATWINC1500 Reset, Enable and Chip Select	Minor update to section
A	01/2019	Document	Initial Release

The Microchip Website

Microchip provides online support via our website at <http://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 <http://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 web site at: <http://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.

© 2019, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-4775-7

Quality Management System

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

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
<p>Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: http://www.microchip.com</p> <p>Atlanta Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455</p> <p>Austin, TX Tel: 512-257-3370</p> <p>Boston Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088</p> <p>Chicago Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075</p> <p>Dallas Addison, TX Tel: 972-818-7423 Fax: 972-818-2924</p> <p>Detroit Novi, MI Tel: 248-848-4000</p> <p>Houston, TX Tel: 281-894-5983</p> <p>Indianapolis Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380</p> <p>Los Angeles Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800</p> <p>Raleigh, NC Tel: 919-844-7510</p> <p>New York, NY Tel: 631-435-6000</p> <p>San Jose, CA Tel: 408-735-9110 Tel: 408-436-4270</p> <p>Canada - Toronto Tel: 905-695-1980 Fax: 905-695-2078</p>	<p>Australia - Sydney Tel: 61-2-9868-6733</p> <p>China - Beijing Tel: 86-10-8569-7000</p> <p>China - Chengdu Tel: 86-28-8665-5511</p> <p>China - Chongqing Tel: 86-23-8980-9588</p> <p>China - Dongguan Tel: 86-769-8702-9880</p> <p>China - Guangzhou Tel: 86-20-8755-8029</p> <p>China - Hangzhou Tel: 86-571-8792-8115</p> <p>China - Hong Kong SAR Tel: 852-2943-5100</p> <p>China - Nanjing Tel: 86-25-8473-2460</p> <p>China - Qingdao Tel: 86-532-8502-7355</p> <p>China - Shanghai Tel: 86-21-3326-8000</p> <p>China - Shenyang Tel: 86-24-2334-2829</p> <p>China - Shenzhen Tel: 86-755-8864-2200</p> <p>China - Suzhou Tel: 86-186-6233-1526</p> <p>China - Wuhan Tel: 86-27-5980-5300</p> <p>China - Xian Tel: 86-29-8833-7252</p> <p>China - Xiamen Tel: 86-592-2388138</p> <p>China - Zhuhai Tel: 86-756-3210040</p>	<p>India - Bangalore Tel: 91-80-3090-4444</p> <p>India - New Delhi Tel: 91-11-4160-8631</p> <p>India - Pune Tel: 91-20-4121-0141</p> <p>Japan - Osaka Tel: 81-6-6152-7160</p> <p>Japan - Tokyo Tel: 81-3-6880-3770</p> <p>Korea - Daegu Tel: 82-53-744-4301</p> <p>Korea - Seoul Tel: 82-2-554-7200</p> <p>Malaysia - Kuala Lumpur Tel: 60-3-7651-7906</p> <p>Malaysia - Penang Tel: 60-4-227-8870</p> <p>Philippines - Manila Tel: 63-2-634-9065</p> <p>Singapore Tel: 65-6334-8870</p> <p>Taiwan - Hsin Chu Tel: 886-3-577-8366</p> <p>Taiwan - Kaohsiung Tel: 886-7-213-7830</p> <p>Taiwan - Taipei Tel: 886-2-2508-8600</p> <p>Thailand - Bangkok Tel: 66-2-694-1351</p> <p>Vietnam - Ho Chi Minh Tel: 84-28-5448-2100</p>	<p>Austria - Wels Tel: 43-7242-2244-39 Fax: 43-7242-2244-393</p> <p>Denmark - Copenhagen Tel: 45-4450-2828 Fax: 45-4485-2829</p> <p>Finland - Espoo Tel: 358-9-4520-820</p> <p>France - Paris Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79</p> <p>Germany - Garching Tel: 49-8931-9700</p> <p>Germany - Haan Tel: 49-2129-3766400</p> <p>Germany - Heilbronn Tel: 49-7131-72400</p> <p>Germany - Karlsruhe Tel: 49-721-625370</p> <p>Germany - Munich Tel: 49-89-627-144-0 Fax: 49-89-627-144-44</p> <p>Germany - Rosenheim Tel: 49-8031-354-560</p> <p>Israel - Ra'anana Tel: 972-9-744-7705</p> <p>Italy - Milan Tel: 39-0331-742611 Fax: 39-0331-466781</p> <p>Italy - Padova Tel: 39-049-7625286</p> <p>Netherlands - Drunen Tel: 31-416-690399 Fax: 31-416-690340</p> <p>Norway - Trondheim Tel: 47-72884388</p> <p>Poland - Warsaw Tel: 48-22-3325737</p> <p>Romania - Bucharest Tel: 40-21-407-87-50</p> <p>Spain - Madrid Tel: 34-91-708-08-90 Fax: 34-91-708-08-91</p> <p>Sweden - Gothenberg Tel: 46-31-704-60-40</p> <p>Sweden - Stockholm Tel: 46-8-5090-4654</p> <p>UK - Wokingham Tel: 44-118-921-5800 Fax: 44-118-921-5820</p>