EE2016 Microprocessor Lab & Theory -July -November 2024

Dr. R Manivasakan

Experiment 5: M2M Communications (under IoT Scenario) in AVR Processors

1 AIM

This experiment deals with the M2M communications (under IoT Scenario) in AVR processors.

1. Demonstrate M2M communication of two AVR-IoT boards through MQTT protocol using Mosquitto broker (server)
2. AVR-IoT board A publishes and AVR-IoT board B subscribes.
3. Board B connected to a LCD module and displays (a) message sent by Board A and (b) light sensor data published (sent) by Board A.

2 Equipments, Hardware Required

To perform this experiment, the following components are required.

1. AVR-IoT WG boards - 2

2. All neccessary cables (USB, power cables etc) and LCDs to be connected with the subscriber board B for displaying received data

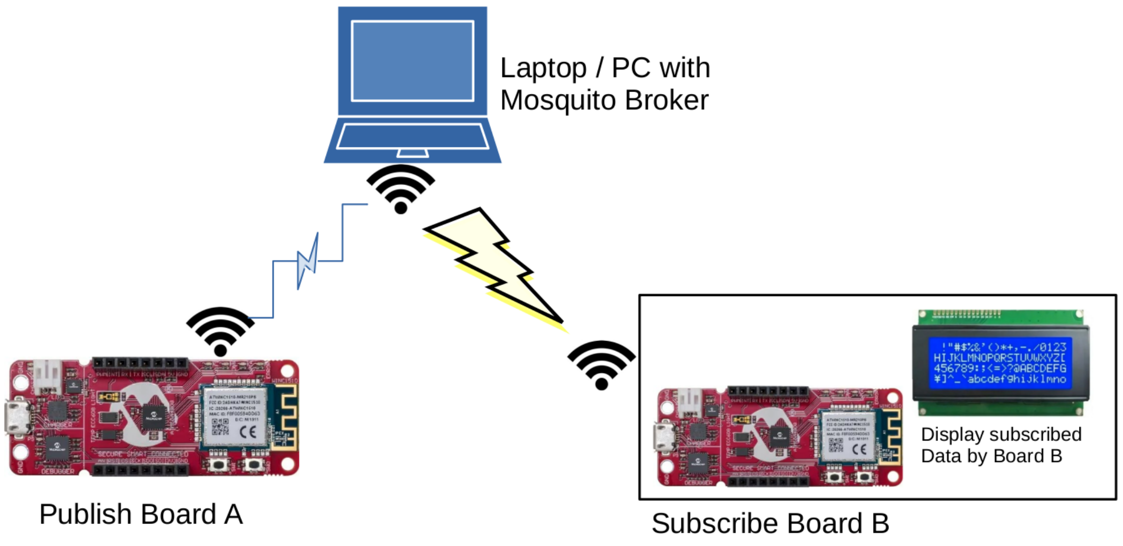
3. Laptop or desktop for configuring it as a mosquitto broker

4. Download free softwares: MQTT libraries, mosquito software and MPLAB IDE platform.

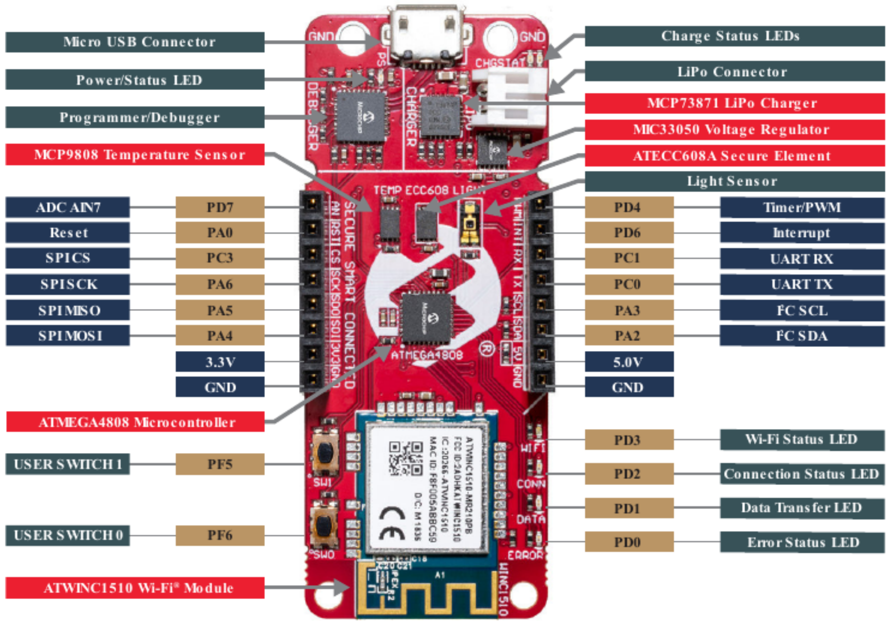
3 Methodology

3.1 Hardware Connections for M2M Communications

Powering the AVR-IoT boards can be done by connecting through USB cable. Board A (Publish) is connected to laptop (which is configured as 'Mosquitto' broker), wirelessly (through WiFi). Similarly, Board B (Subscribe) is connected to the Mosquitto broker through WiFi. Since, the received data at Board B (Subscribe) is to be displayed in an LCD, LCD is connected to Board B (Subscribe) as given below. [Note: Board A is NOT connected to Board B DIRECTLY].



The only hardware connection intricacies are at the subscribing Board B. The right pins of the board B has to be connected to the right LCD pins.



Make the connections according to the connection diagram of the LCD with AVR-IoT board, as below. <Include, here a picture of connection diagrams for AVR-IoT board to LCD terminals?>

AVR Microcontroller Pin	**LCD Screen Pin**	**Connection**	
PA0 (RS)	RS	Control Pin (Register Select)	
Ground	RW	Control Pin (Read/Write, connect to Ground)	
PC3 (E)	E	Control Pin (Enable)	
PA4 (D4)	D4	Data Pin (4-bit mode)	
PA5 (D5)	D5	Data Pin (4-bit mode)	
PA6 (D6)	D6	Data Pin (4-bit mode)	
PD7 (D7)	D7	Data Pin (4-bit mode)	
+5V	VDD	Power Supply (5V)	
Ground	VSS	Ground	
Potentiometer Wiper	V0/VEE	Contrast Adjustment	

3.2 M2M Communications: MQTT Implementation along with Mosquitto Server (Broker)

The problem in this experiment is to implement M2M communications between two IoTs, through an intermediate broker. One board 'publish'es (Board A) on a default topic through the mosquito server (a laptop), while the other '*subscribes*' (Board B) to the above default topic (after connecting to the above mosquito broker through WiFi) and receives the message. In our experiment, the 'publish'ed message is displayed on the LCD connected to Board B. We also intend to display the light sensor data 'publish'ed (or transmitted) by Board A, in the LCD, connected to Board B. The Board B (Subscribe) independently publishes on EE2016IITM and is not monitored.

Following is the procedure for connecting two AVR-IoT boards using MQTT with a Mosquitto server to realize above M2M communication [McroChip1].

3.2.1 Outline of Procedure:

Mosquitto server: Install Mosquitto server on your laptop . Instructions for various operating systems can be found on the Mosquitto website (). We recommend that your laptop be connected to iitmwifi, download mosquitto.

AVR-IoT boards: Ensure your boards have Wi-Fi connectivity and are programmed with an MQTT library compatible with the AVR architecture, with Board A burnt with publish code (mainPublish.c supportPublish.c codes uploaded in moodle) and Board B with subscribe code (mainSubscribe.c supportSubscribe.c uploaded in moodle). Download and install Microchip's MPLAB IPE in your labtop. The publish & subscribe functions are tweaked & added in ([MPLAB1]), according to our requirement for our experiment. [Alternatively, you can also download, independently tweak them and use them to solve the problem posed. But it is time consuming and strenuous exercise. We instead reccommend supportPublish.c to copy in place of /Source Files/MCC_Generated Files/examples/mqtt_example.c file in the same directory in MPLAB environment [MPLAB1] and ditto for Board B (Subscribe)]. Dump the program into respective boards and run it simultaneously with the mosquitto broker ON.

Demonstration (Tasks): (1) Every 'publish' cycle (default, once every 10 sec), the LED should blink, repeate the experiment by changing the publish rate and LED blink rate would increase, (2) receive the published message through subscription and display in LCD and (3) publish, receive and display in LCD the light sensor data.

MPLAB IPE: For all of the above, download and install Microchip's MPLAB IPE in your labtop (ubuntu versions are also available). It does lot things, which makes things easier for you. Create a new project on IoT (details given below), download and include MQTT codes corresponding to the publish and subscribe functionalities of the IoT nodes, edit the mqtt_example.c in the MPLAB IPE (edited versions are supportPublish.c). The original mqtt_example.c is replaced with its edited version and this would be used to burn codes into the respective AVR-IOT WG boards (but after this, boards are disconnected and independently run, ideally as they are battery operated, emulating the real-world IoT scenario).

3.2.2 Steps:

Step 1: Set up Mosquitto Server (on your own laptop or desktop in IE lab): Here we describe, how to set up the AVR® IoT WG development board as a Message Queuing Telemetry Transport (MQTT) client [McroChip1]. For this, we can use more applications, like MyMQTT (android app), MQTT Explorer (desktop application) and Wireshark (a free and open-source packet analyzer) which are described in next subsection.

Since, the M2M communication we intend to implement is brokered (as opposed to peer-to-peer communication), we adopt mosquitto broker. In what follows, we give the procedure for installing the mosquitto broker [Msqtto].

1. Go to the Mosquitto web site.
2. In the Windows section, download Mosquitto (usually 64-bit version) and execute the EXE file to install it. (Alternatively, for ubuntu go to Linux distributions section).
3. Mosquitto Config (See [McroChip1])
   (a) Go to the file where you installed it and right-click on mosquitto.config then select Open with > Notepad++. That is the file we will execute. All lines are commented with "#" for now, so it is empty. Go to the end of the file, write, and save the following code [MPLAB1] :
   ```
   listener 9001
   protocol websockets
   listener 1883
   allow_anonymous true
   password_file C:\Program Files\mosquitto\pwfile.example
   ```
   - 1st line opens port 9001 as listener.
   - The 2nd line establishes port 9001 will use WebSocket protocol.
   - The 3rd line opens port 1883 as listener.
   - The 4th line allows connection to clients without a registered username (i.e., it's not in #pwfile.example).
   - The 5th line sets pwfile.example as the file that contains what usernames are registered, and what are the passwords for each of them.

   A common mistake in this code is to add comments or spaces in the code. In this program language, every character is considered. Also, be careful not to add any spaces in the code. If you make a comment, it will need to be the whole line. Do not make a comment after a coding line since the program will not understand it and it will fail.

The broker will open 9001 for WebSockets, and port 1883, where no username or password is needed. You can write “allow_anonymous false” instead of “allow_anonymous true”, so only users with identified usernames and passwords will be allowed. We will see the utility of pwfile.example in the next section.

An example (pwfile.example) is given in [McroChip1].

4. Execute Mosquitto as a Broker

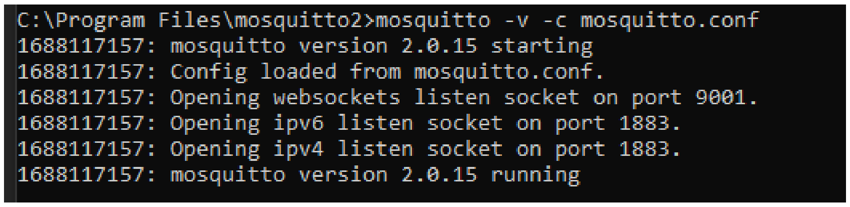
   (a) Initialize Mosquitto Open Command Prompt on your computer. Usually, this can be done by clicking on the Windows key and writing “Command Prompt”.

   (b) Open the directory where mosquitto has been installed. Usually, that can be done with the next line: cd C:\Program Files\mosquitto

   (c) Then execute this line to execute mosquitto.conf, which should initialize the Eclipse Mosquitto broker:

   `mosquitto -v -c mosquitto.conf`

   Now, you can see the initialization of the broker in the command prompt.



```
C:\Program Files\mosquitto2>mosquitto -v -c mosquitto.conf
1688117157: mosquitto version 2.0.15 starting
1688117157: Config loaded from mosquitto.conf.
1688117157: Opening websockets listen socket on port 9001.
1688117157: Opening ipv6 listen socket on port 1883.
1688117157: Opening ipv4 listen socket on port 1883.
1688117157: mosquitto version 2.0.15 running
```

   You can also execute mosquitto.config file straight away, but you will not see what events happen in it.

   (d) Now open another Command Prompt and the following command to encrypt J.Lo’s password:

   `mosquitto_passwd -U pwfile.example`

   (e) For closing Mosquitto, press Ctrl+C (for Windows) on the Command Prompt, or just close it.

Procedures for mosquitto already running and testing mosquitto, please see [McroChip1].

Step 2: General Procedure for the programming / configuring the AVR-IoT Boards (Both Board A (Publish) and Board B (Subscribe)) The first task is to add the MQTT library into MPLAB® Code Configurator (MCC) [MPLAB1]

Step 2.1: Addition of MQTT Library for MPLAB Code Configurator (MCC)

1. Download the Library

   (a) Download the Library MQTT Library for MCC MQTT Library.

2. Import the above downloaded MQTT library files into MPLAB X IDE: (If you are unfamiliar with the process, read the article How to Add a Library in MCC Import MPLAB).

3. Create a new project for AVR-IoT WG and open MCC. In Device Resources, click the Green Plus Icon next to MQTT.

4. The MQTT Foundation service should be added. When you change the Transport Service from Custom Service to Wireless, the WINC15XX and SPI0 peripheral libraries should appear. WINC1510 is connected to the microcontroller ATmega4808 in AVR-IoT WG.

5. The AVR-IoT WG board AVR-IOT-WG contains the WINC1510 wireless module and the ATmega4808 microcontroller. ATmega4808 microcontroller is programmed, but it is not capable of connecting to Wi-Fi® by itself. It connects to the WINC1510 using SPI, and WINC1510 connects to the Wi-Fi.

Step 2.2 Configuring MQTT Window Setting Explanation

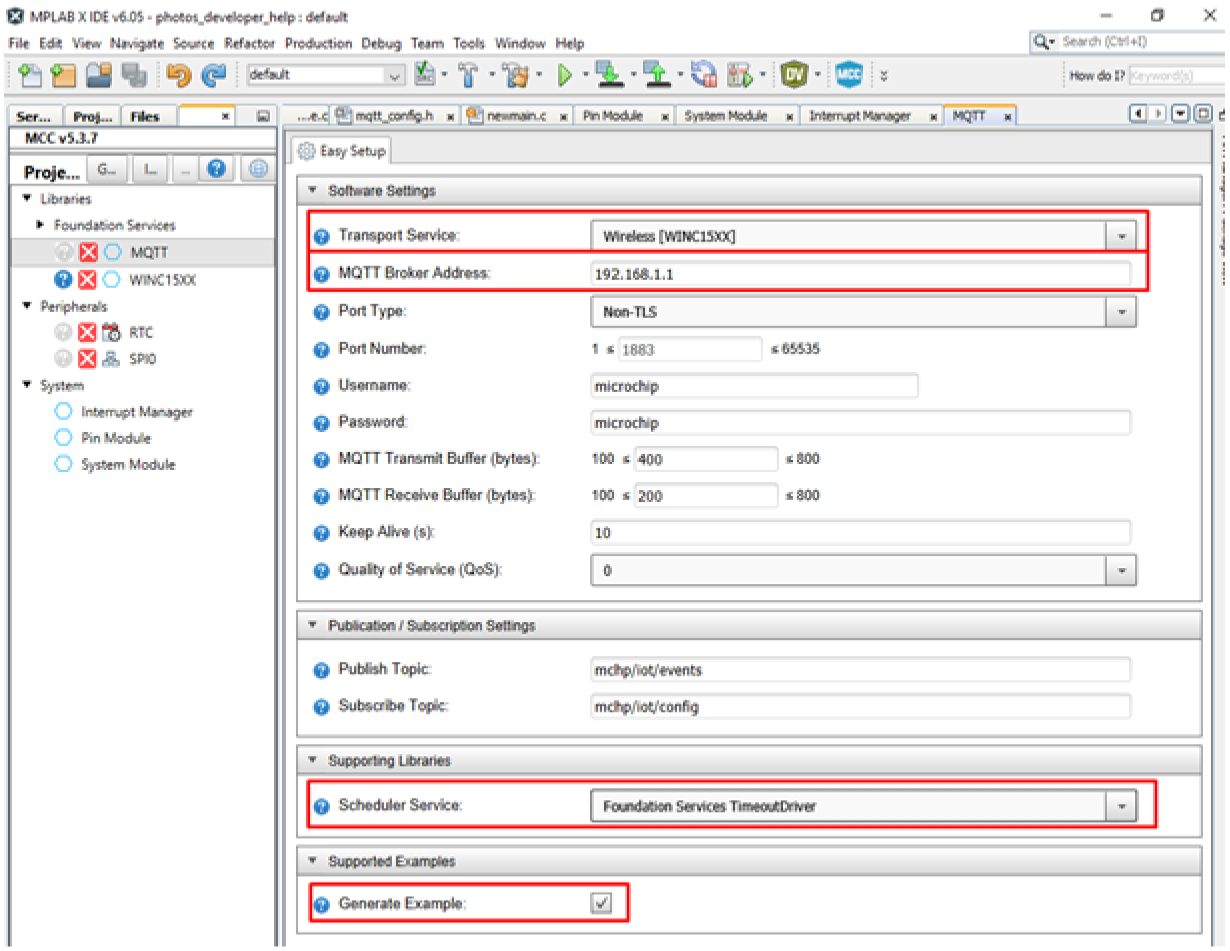
Transport Service –> Selects how MQTT is going to be used (Wired or Wireless)

MQTT Broker Address –> This is the IP, "the name" of the broker device that we are going to connect to

Publish Topic –> mchp/iot/events (Default - You can change) [In our experiment, we are going to have Board B does receive the published message from default topic (published by Board A) by subscribing to that default topic, while simultaneously publishing some other topic, which we don't monitor. Board A only publishes on a default topic].

Scheduler Service –> This setting selects, what tool will be used to measure time to make periodic publishing.

Generate Example should be checked (This would generate default files for MQTT)


MPLAB X IDE v6.05 - photos_developer_help : default
File Edit View Navigate Source Refactor Production Debug Team Tools Window Help
MCC v5.3.7
Libraries
Foundation Services
MQTT
WINC15XX
Peripherals
RTC
SPI0
System
Interrupt Manager
Pin Module
System Module
Easy Setup

Software Settings

Transport Service:	Wireless [WINC15XX]
MQTT Broker Address:	192.168.1.1
Port Type:	Non-TLS
Port Number:	1 ≤ 1883 ≤ 65535
Username:	microchip
Password:	microchip
MQTT Transmit Buffer (bytes):	100 ≤ 400 ≤ 800
MQTT Receive Buffer (bytes):	100 ≤ 200 ≤ 800
Keep Alive (s):	10
Quality of Service (QoS):	0

Publication / Subscription Settings

Publish Topic:	mchp/iot/events
Subscribe Topic:	mchp/iot/config

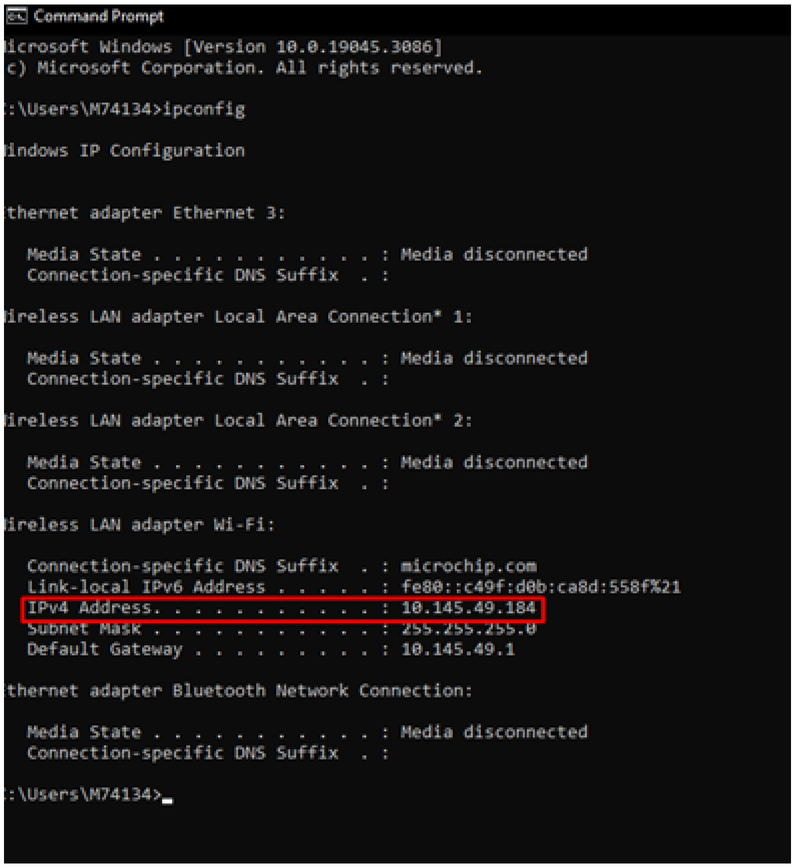
Supporting Libraries

Scheduler Service:	Foundation Services TimeoutDriver

Supported Examples

Generate Example:	✓

MQTT broker address is the IP address of your laptop (in which mosquitto software is loaded). To know its address, open the command prompt, and type ipconfig. The necessary address is contained in the IPv4 address's digits.



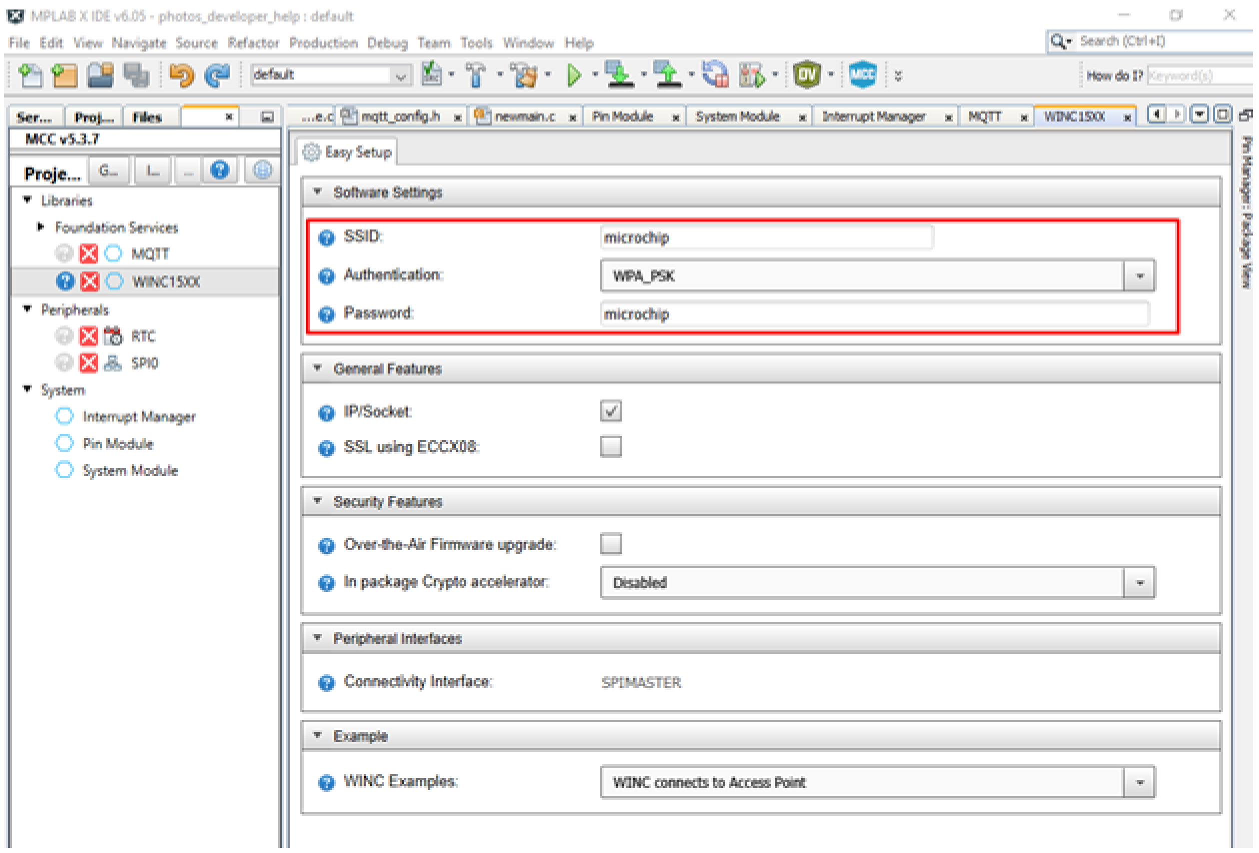
```
icrosoft Windows [Version 10.0.19045.3086]
c) Microsoft Corporation. All rights reserved.

:\Users\M74134>ipconfig

indows IP Configuration


thernet adapter Ethernet 3:

  Media State . . . . . . . . . . . : Media disconnected
  Connection-specific DNS Suffix  . :

ireless LAN adapter Local Area Connection* 1:

  Media State . . . . . . . . . . . : Media disconnected
  Connection-specific DNS Suffix  . :

ireless LAN adapter Local Area Connection* 2:

  Media State . . . . . . . . . . . : Media disconnected
  Connection-specific DNS Suffix  . :

ireless LAN adapter Wi-Fi:

  Connection-specific DNS Suffix  . : microchip.com
  Link-local IPv6 Address . . . . . : fe80::c49f:d0b:ca8d:558f%21
  IPv4 Address. . . . . . . . . . . : 10.145.49.184
  Subnet Mask . . . . . . . . . . . : 255.255.255.0
  Default Gateway . . . . . . . . . : 10.145.49.1

thernet adapter Bluetooth Network Connection:

  Media State . . . . . . . . . . . : Media disconnected
  Connection-specific DNS Suffix  . :

:\Users\M74134>_
```

Step 2.3: Configuring WINC1510 Module in WIN15XX Software Settings Window Change these three settings in the WIN15XX Software Settings window, as a part of configuration settings in MCC

SSID –> Name of the Wi-Fi you are connecting to

Authentication –> Select if the Wi-Fi uses a password, WPA of WEP. Usually, your device will use WPA2. If so, select WPA_PSK

Password –> Password of the Wi-Fi you are connecting to


MPLAB X IDE v6.05 - photos_developer_help : default
File Edit View Navigate Source Refactor Production Debug Team Tools Window Help
MCC v5.3.7
Libraries
Foundation Services
MQTT
WINC15XX
Peripherals
RTC
SPI0
System
Interrupt Manager
Pin Module
System Module
Easy Setup

Software Settings

SSID: microchip

Authentication: WPA_PSK

Password: microchip

General Features

IP/Socket: ✓

SSL using ECCX08:

Security Features

Over-the-Air Firmware upgrade:

In package Crypto accelerator: Disabled

Peripheral Interfaces

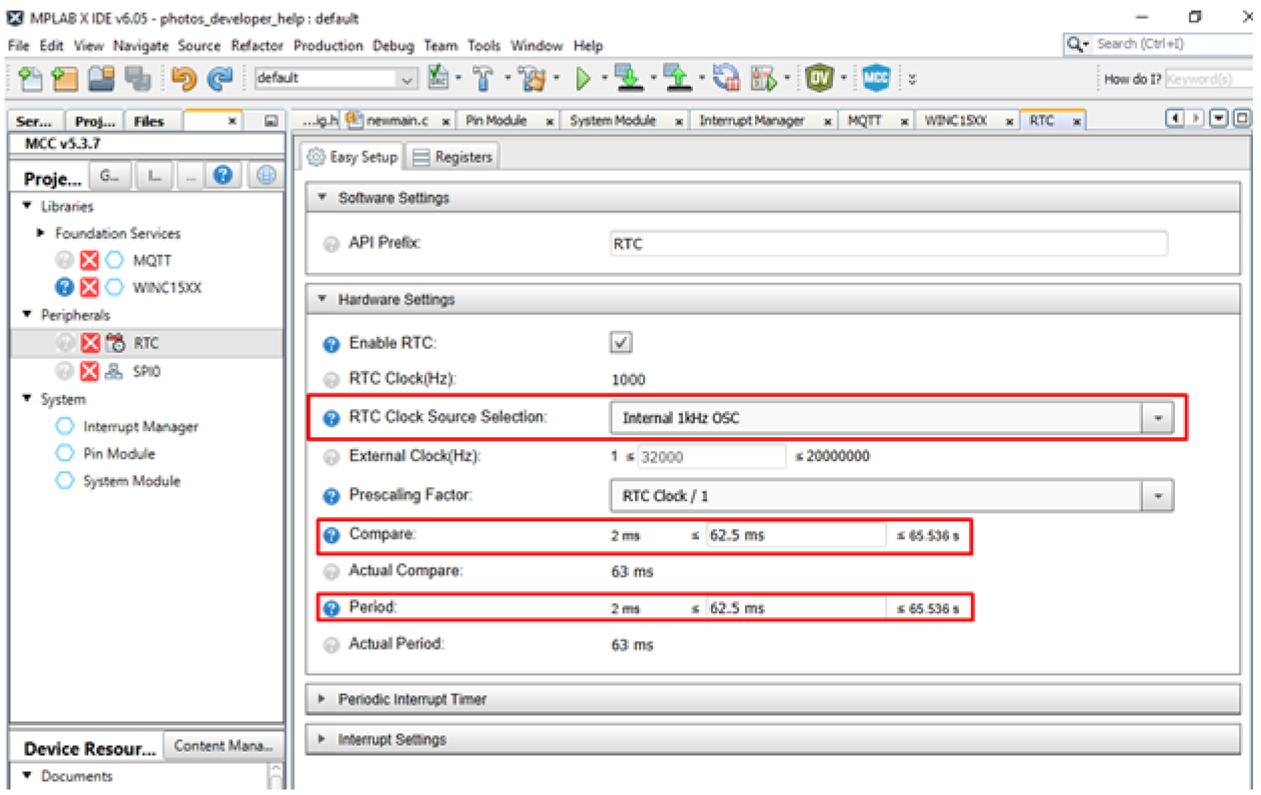
Connectivity Interface: SPIMASTER

Example

WINC Examples: WINC connects to Access Point

Step 2.4: RTC Window and SPI0 While you were configuring the different options, the RTC and SPI0 peripherals should have appeared. RTC is a timer used to know when to send data through MQTT, and SPI0 is the peripheral used to communicate with the WINC1510 device. If you don't see RTC and SPI0 peripherals, you may need to click on the small arrow at the left side of "Peripherals".

In this case, you won't need to change SPI0, only RTC. Select 1 kHz rate for source rate because that's how the MQTT example that you are going to generate was designed and select times for "Compare" and "Period" that are inside of the new range, between 2 ms and 65 s.



API Prefix: RTC

Enable RTC: ✓

RTC Clock(Hz): 1000

RTC Clock Source Selection: Internal 1kHz OSC

External Clock(Hz): 1 ≤ 32000 ≤ 20000000

Prescaling Factor: RTC Clock / 1

Compare: 2 ms ≤ 62.5 ms ≤ 65.536 s

Actual Compare: 63 ms

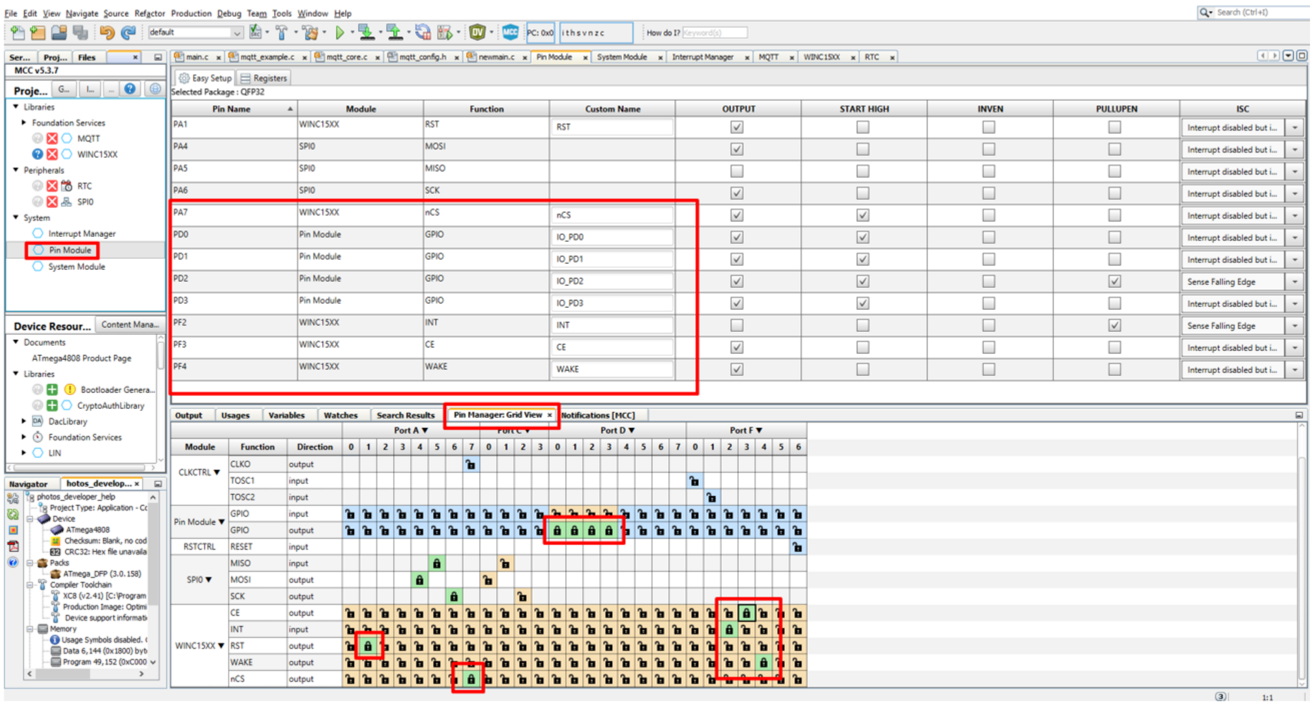
Period: 2 ms ≤ 62.5 ms ≤ 65.536 s

Actual Period: 63 ms

Step 2.5: PIN Module As previously mentioned, the microcontroller communicates with the ATWINC1510 W-Fi module through the Serial Peripheral Interface (SPI) protocol, using the SPI0 peripheral. SPI protocol uses 4 wires; nCS, MOSI, MISO, and SCK. All these wires are properly configured except nCS, which you need to add. nCS is connected to the PA7 pin. Although it is not needed in this exercise, more information about the SPI communication protocol can be found on the Microchip SPI Peripheral page.

To communicate with the MCU, the WINC1510 device also has the RST pin for resetting, INT pin for interrupts, EN for enabling, and WAKE pin for waking up. These pins are used for the MCU and are connected to pin PA1, pin PF2, pin PF3, and pin PF4, respectively.

Apart from nCS, RST, INT, EN, and WAKE pins, we will also use LEDs. LEDs are connected to PD0, PD1, PD2 and PD3.



Selected Package : QFP32

Pin Name	Module	Function	Custom Name	OUTPUT	START HIGH	INVEN	PULLUPEN	ISC
PA1	WINC15XX	RST	RST	✓				Interrupt disabled but i...
PA4	SPI0	MOSI		✓				Interrupt disabled but i...
PA5	SPI0	MISO						Interrupt disabled but i...
PA6	SPI0	SCK		✓				Interrupt disabled but i...
PA7	WINC15XX	nCS	nCS	✓	✓			Interrupt disabled but i...
PD0	Pin Module	GPIO	IO_PD0	✓	✓			Interrupt disabled but i...
PD1	Pin Module	GPIO	IO_PD1	✓	✓			Interrupt disabled but i...
PD2	Pin Module	GPIO	IO_PD2	✓	✓		✓	Sense Falling Edge
PD3	Pin Module	GPIO	IO_PD3	✓	✓			Interrupt disabled but i...
PF2	WINC15XX	INT	INT				✓	Sense Falling Edge
PF3	WINC15XX	CE	CE	✓				Interrupt disabled but i...
PF4	WINC15XX	WAKE	WAKE	✓				Interrupt disabled but i...

We need to tell the MCU that we are going to use these eight pins: RST, INT, nCS and WAKE and D0, D1, D2 and D3 as GPIO output. Click in Pin Manager: Grid View and add the nine pins that we need. While you are selecting, you should see how pins are being added if you click Pin Module.

It is recommended to make D0, D1, D2, and D3 start high, by ticking the respective box.

Your final pin setup should look like the above image:

Step 2.6: Generate Code If every thing goes fine, you would not see a window: Ref [MPLAB1]

Step 2.7: Write Code in main.c Write your own main() code, which would adopt the most important key MQTT library function `app_mqttScheduler()` among others.

Step 3: Writing the ADC for light sensor data & LCD Interfacing

Step 3.1: Writing the ADC for Light Sensor Data

Step 3.2: LCD Interfacing

Step 3: Burning the Object Code into the Boards In this step, we would edit the base / example codes (one for Board A (Publish) and another for Board B (Subscribe)) in MPLAB. This tweaked / edited file being supportPublish.c, to copy in place of /Source Files/MCC_Generated Files/examples/mqtt_eample.c file in the same directory in MPLAB environment [MPLAB1].

Burn the code [MPLAB1]. Go through the user guide AVR-IoT-WG of AVR-IoT WG board [AVR-IOT-WG-UsrGde] to know how the hardware connections to be made, during burning the code.

Step 3.1: Burning Code in Board A (Publish) The code (mainPublish.c) along with the suitable support files (SupportPublish.c) would only publish and is meant for Board A (Publish). Download these codes from moodle, repeat the procedure given in Step 2 (import, configure in MPLAB in your laptop, replace with SupportPublish.c, and so on). Connect the Board A to your laptop through USB and click the "*run main project*" in the MPLAB window to burn the code into Board A (Publish).

Step 3.2: Burning Code in Board B (Subscribe) The code for Board B (Subscribe) consists of two parts: (a) the subscribe part of MQTT protocol (and there is also a publish part, inherently comes along) and (b) the received message through subscription has to be displayed in the LCD (i) Hello message and (ii) light sensor data.

Part (a) Similar to the publish node, code (mainSubscribe.c) and suitable support files (supportSubscribe.c) are different, in which the Board B (Subscribe) does receive the published message from default topic, by subscribing to the default topic, while simultaneously publishing some other topic (EE2016IITM), which we don't monitor. Hence, in Configuring MQTT Window need to choose the right topic for both publishing and subscribing. In case, you shut down the mosquitto server, you need to reset the boards. Run the Mosquitto server and view the results.

Part (b): We use the 16 x 2 LCD (Hitachi See HitchiLCD) to display the message received through subscription. Refer [AVR-IOT-WG-UsrGde] to the PIN connections of AVR-IoT WG, and the details given in section 3.1 and accordingly make the connections for the LCD to AVT-IoT WG board terminals.

4 Tasks

1. Do the experiment in your hostel room itself by configuring your laptop as a MOSQUITTO broker and your mobile & your friend's mobile as a publish and subscribe nodes.

2. Change the Publish rate and observe the corresponding received rate through Subscription.

   (a) Go to (in MPLAB) main()–>header files –> MCC Generated files –> config –> MQTT_config.h CFG_MQTT_CONN_TIMEOUT 10 (as generated by MPLAB - you can change the publish rate to 6, 10, 14, 18 & 22 - & run for each time. Observe the LED blink rate for each case).

3. Receive, 'Hello', message & display it. Independently transmit to display Hello in LCD. Now, interface it with MQTT iot protocol. The problem is that the MQTT uses many of the pins in each available ports (PORT A, C & D). Seems you need to take some pins from a port and remaining from another port and combine them to send nibble at a time to the display LCD.

4. Receive the light sensor data and display it.

Acknowledgements: The author acknowledges, Yuvasri and Prasanna to take up this as a summer internship project 2024, and working on it on a daily basis, Saran Prithive for implementation of software aspects, Anand for fixing the LCD interfacing part, apart from Bhuvaneshwari for management.

References

[McroChip1] https://developerhelp.microchip.com/xwiki/bin/view/products/wireless-connectivity/Embedded-Wi-Fi/mqtt-in-mplab/mqtt-environment/

[Msqtto] https://mosquitto.org/download/

[MPLAB1] https://developerhelp.microchip.com/xwiki/bin/view/products/wireless-connectivity/Embedded-Wi-Fi/mqtt-in-mplab/first-steps-in-mplab/

[AVR-IOT-WG-UsrGde] https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocumen
IoT-WG-Development-Board-User-Guide-DS50002809D.pdf

[HitachiLCD] HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver)