PIC® and AVR® IoT – Transceiving Data with AWS User Guide

Introduction

Author: Lars Olav Skrebergene, Microchip Technology Inc.

Important: This user guide is a part of a series of tutorials originally published in the Microchip IoT Developer Guides repository on GitHub. The repository has more tutorials and information about using a PIC® or AVR® IoT Board with AWS.

In this tutorial, it will be showcased how the AVR-IoT and PIC-IoT Development Boards can be configured to communicate with the cloud using Amazon Web Services® (AWS) and the MQTT messaging protocol.

It will be demonstrated how a an example application can be developed where a network of IoT devices are configured to blink their LEDs whenever a button is pressed on any of them. Detailed step-by-step instructions will be provided and relevant concepts will also be covered when needed.

The primary goal of this tutorial is for the reader to experience how to develop their own applications with the AVR-IoT and PIC-IoT Development Boards.

Prerequisites

It is assumed that the reader has already provisioned their PIC-IoT and/or AVR-IoT Development Board(s) to communicate with their own AWS account, as described in the previous tutorial: Connect the Board to your AWS Account.

Before starting this tutorial, make sure that IoT device(s) are successfully sending sensor data to AWS IoT Core. It is also assumed that the reader has installed the MPLAB® X IDE and the XC8 (AVR-IoT) or XC16 (PIC-IoT) compiler.

Links to software and other useful tools and guides are provided in the Resources section at the end of this tutorial.

View Code Example on GitHub
Click to browse repository

Table of Contents

1. A Brief Introduction to the Firmware of the IoT Boards

The firmware that is pre-loaded onto the PIC-IoT and AVR-IoT Development Boards is available on GitHub and will form the starting point for the example application that will be designed in this tutorial. The MPLAB X projects for the different microcontroller families can be found here:

- GitHub repository for the **PIC**-IoT Development Boards
- GitHub repository for the **AVR**-IoT Development Boards

The `PICIoT.X` and `AVRIoT.X` projects contain many different files that handle cryptography, Wi-Fi connectivity, MQTT communication, and so on. The main focus of this tutorial will be `application_manager.c`, which is located under `Source Files -> MCC Generated Files` in MPLAB X. This file contains a lot of useful high-level functions that make it easy to develop an AWS application.

Here is a summary of some important functions in `application_manager.c` that is relevant for this tutorial:

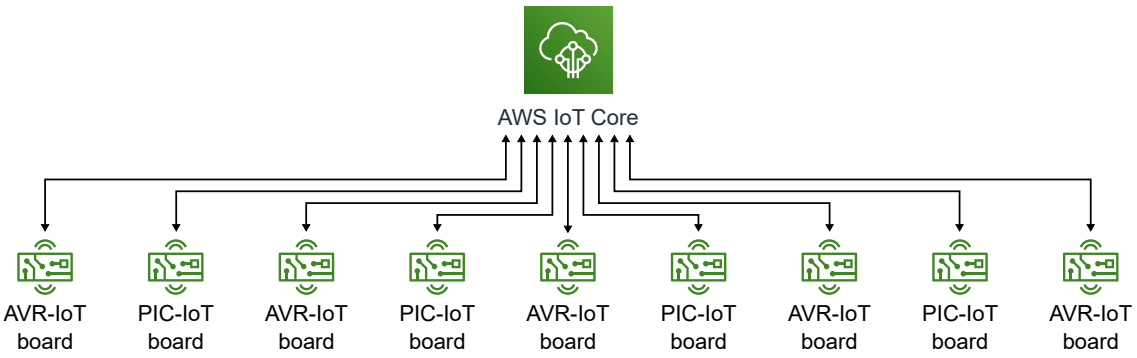
- `subscribeToCloud`
  - Defines which MQTT topics the IoT board should be subscribed to and which functions should be run when messages are received to these topics. In the unmodified `PICIoT.X` and `AVRIoT.X` projects, the board is only subscribed to its device shadow update MQTT topic, and `receivedFromCloud` is the function specified to handle these updates.
- `receivedFromCloud`
  - Runs when an MQTT shadow update message is received in the unmodified `PICIoT.X` or `AVRIoT.X` projects. This function analyzes the message and performs some action based on its contents. It then calls the `updateDeviceShadow` function to acknowledge that the shadow update has been received.
- `sendToCloud`
  - Called every second to send sensor data to the cloud as MQTT messages. This function is a good template to learn how to send custom MQTT messages to custom MQTT topics using the IoT boards.

The reader is encouraged to take a quick look at `application_manager.c`, and in particular, at these functions to get an overview of the existing functionality.

2. Implementing the Example Application

In this example, it will be demonstrated how devices can be configured to send and receive messages over custom MQTT topics. An example application will be implemented where button presses on any of the configured devices will cause all devices to flash their LEDs. All communication will be sent between the IoT boards and AWS IoT Core, as illustrated in the schematic below. No direct device-to-device communication will be used.

Figure 2-1. Flowchart showing how information flows



2.1 Step 1: Start With an Unmodified Version of the Github Project

The starting point for this example is an unmodified copy of the GitHub project that is compatible with the microcontroller family of the device that will be used:

- GitHub repository for the **PIC**-IOT Development Boards
- GitHub repository for the **AVR**-IOT Development Boards

Download the correct repository and open the `PICIoT.X` and/or `AVRIoT.X` project in MPLAB X.

2.2 Step 2: Sending MQTT Messages to the Cloud

The first thing to do is to detect when a button is pressed, which will be done using interrupts. The procedures for this differ somewhat for AVR-IoT and PIC-IoT boards. Follow Procedures for AVR-IoT boards or Procedures for PIC-IoT boards, depending on which board you are using, and then continue with Procedures for both AVR-IoT and PIC-IoT boards.

Procedures for AVR-IoT Boards

In `application_manager.c`, add the following code just after the `SYSTEM_Initialize()` call in the `application_init` function:

```
SW0_EnableInterruptForFallingEdge();
PORTF_SW0_SetInterruptHandler(sendButtonPressToCloud);
```

The first line enables falling edge interrupt detection for the `SW0` button on the AVR-IoT boards, and a function handler for this interrupt is then assigned on the second line. The interested reader is encouraged to take a look at the `pin_manager.c` file to see how these functions are implemented.

Skip ahead to *Procedures for both AVR-IoT and PIC-IoT boards* to complete this step.

Procedures for PIC-IoT Boards

In `pin_manager.c`, perform the following edits:

1. In the `PIN_MANAGER_initialize` function, enable interrupts for the `SW0` button (which is connected to RA7) and clear its interrupt flag by including these two lines:

```
IOCNAbits.IOCNA7 = 1;      //Pin : RA7
IOCFAbits.IOCFA7 = 0;      //Pin : RA7
```

2. Add another variable below the `INT_InterruptHandler` variable to store the interrupt handler for the `SW0` hardware button:

```
void (*SW0_InterruptHandler)(void) = NULL;
```

3. Add a function that sets the variable we just created (place it just after the `INT_SetInterruptHandler` function):

```
void SW0_SetInterruptHandler(void (* InterruptHandler)(void))
{
    IEC1bits.IOCIE = 0; //Disable IOCI interrupt
    SW0_InterruptHandler = InterruptHandler;
    IEC1bits.IOCIE = 1; //Enable IOCI interrupt
}
```

4. Modify the `_IOCInterrupt` interrupt service routine to also handle the `SW0` button presses (the interrupt service routine is located near line 155 in `pin_manager.c`). The `SW0` button is connected to the `RA7` pin. The fully modified interrupt service routine is provided below. Either copy and replace `_IOCInterrupt` in its entirety or add the second nested `if` statement to your project.

```
void __attribute__ (( interrupt, no_auto_psv )) _IOCInterrupt ( void    )
{
    if(IFS1bits.IOCIF == 1)
    {
        // Clear the flag
        IFS1bits.IOCIF = 0;
        if(IOCFAbits.IOCFA12 == 1)
        {
            IOCFAbits.IOCFA12 = 0; //Clear flag for Pin - RA12
            if(INT_InterruptHandler)
            {
                INT_InterruptHandler();
            }
        }

        // Handle SW0 button presses
        if(IOCFAbits.IOCFA7 == 1)
        {
            IOCFAbits.IOCFA7 = 0; //Clear flag for Pin - RA7
            if(SW0_InterruptHandler)
            {
                SW0_InterruptHandler();
            }
        }
    }
}
```

In `pin_manager.h`, add a declaration of the `SW0_SetInterruptHandler` function that was just added to make it available in other files, for example after the declaration of the `INT_SetInterruptHandler` function:

```
void SW0_SetInterruptHandler(void (* InterruptHandler)(void));
```

In `application_manager.c`, set the `SW0` interrupt handler just after the call to the `SYSTEM_Initialize()` in the `application_init` function:

```
// Set interrupt handler for button presses
SW0_SetInterruptHandler(sendButtonPressToCloud);
```

Procedures for Both AVR-IoT and PIC-IoT Boards

Now, any time the SW0 button is pressed, the sendButtonPressToCloud function will be called. Prior to implementing this function, declare a variable for the MQTT topic that will be used. Add the following declaration to application_manager.c (e.g., below the declaration of the mqttSubscribeTopic variable):

```
char tutorialMqttSubscribeTopic[SUBSCRIBE_TOPIC_SIZE];
```

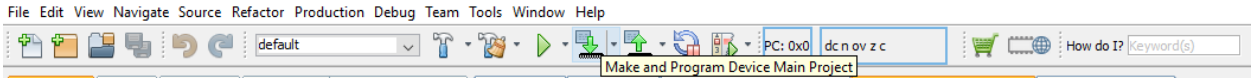
Implement the aforementioned function handler by adding the following code to application_manager.c:

```
static void sendButtonPressToCloud(){
    // Ensure that we have a valid cloud connection
    if (shared_networking_params.haveAPConnection)
    {
        static char tutorialPayload[PAYLOAD_SIZE];
        int tutorialLen = 0;

        // Set MQTT topic
        memset((void*)tutorialMqttSubscribeTopic, 0, sizeof(tutorialMqttSubscribeTopic));
        sprintf(tutorialMqttSubscribeTopic, "buttonPresses");

        // Construct payload
        tutorialLen = sprintf(tutorialPayload,"{\"thing_name\":\"%s\"}", cid);

        // Publish data to cloud
        CLOUD_publishData((uint8_t*)tutorialMqttSubscribeTopic ,(uint8_t*)tutorialPayload,
tutorialLen);
    }
}
```

This function closely resembles the sendToCloud function that was mentioned earlier and will publish an MQTT message to the buttonPresses topic. The content of the message will be a JSON object that contains the name of the thing/device that sent the message.

Build the modified project and program it onto the device using MPLAB X. This is done by clicking on the **Make and Program Device Main Project** button on the MPLAB X toolbar (see the image below).

Figure 2-2. How to make and program device in MPLAB X



If you are unfamiliar with the MPLAB X integrated developer environment (IDE), check out the following guide: Get Started with MPLAB® X IDE and Microchip Tools

Verify that Messages are Successfully Being Sent to AWS

When the device has been successfully programmed, the next step is to make sure that messages are being received in AWS:

1. Sign in to the AWS Management Console and select the IoT Core service.
2. Select **Test** in the menu on the left-hand side
3. In the **Subscription topic** field, enter buttonPresses.
4. Click the **Subscribe to topic** button.
5. Press the SW0 button on the board and observe that the button press is successfully registered in the cloud.

2.3 Step 3: Receiving MQTT Messages from the Cloud

Now that the project has been successfully modified to send messages to a custom topic, it is time to find a way to subscribe to this topic:

1. Change the definition of NUM_TOPICS_SUBSCRIBE in mqtt_config.h (Header Files -> MCC Generated Files -> config) to allow up to two simultaneous MQTT topic subscriptions:

```
#defineNUM_TOPICS_SUBSCRIBE2
```

2. Edit the `subscribeToCloud` function in `application_manager.c` to include a subscription to the `buttonPresses` topic. The fully modified function is provided below. Either copy and replace the `subscribeToCloud` function in its entirety or add the last two lines of the code below in your MPLAB X project.

```
staticvoidsubscribeToCloud(void)
{
   sprintf(mqttSubscribeTopic, "$aws/things/%s/shadow/update/delta", cid);
   CLOUD_registerSubscription((uint8_t*)mqttSubscribeTopic,receivedFromCloud);
   sprintf(tutorialMqttSubscribeTopic, "buttonPresses");

CLOUD_registerSubscription((uint8_t*)tutorialMqttSubscribeTopic,receiveButtonPressFromClo
ud);
}
```

   – The second parameter of the `CLOUD_registerSubscription` function is a handler that dictates which function will be run when a message is received to the specified topic. The `receiveButtonPressFromCloud` function will therefore have to be implemented to handle any received messages.

3. Add the following function definition to `application_manager.c` (somewhere above the `subscribeToCloud` function) to make the device's LEDs blink twice when a message is received:

```
staticvoidreceiveButtonPressFromCloud(uint8_t *topic, uint8_t *payload){
    LED_test();
    LED_test();
}
```

4. Build the project and program the device in MPLAB X. If multiple AVR-IoT or PIC-IoT devices are available, try programming several of them using the same project.

   Remember that all of the devices first will have to be provisioned for use with an AWS account. Note also that even though AVR-IoT and PIC-IoT devices can be connected to AWS simultaneously and communicate with each other over MQTT, the GitHub project used in this tutorial is only compatible with either AVR-IoT devices or PIC-IoT devices. To use devices from two different device families together, it is necessary to complete this tutorial individually for the AVR-IoT and PIC-IoT repositories on GitHub and program the devices with the compatible firmware.

The device(s) should now be configured correctly. If the `SW0` button is pressed on any of the configured IoT kits, the LEDs on all configured IoT kits should blink twice. If this is not the case, make sure that the tutorial has been followed correctly and that the devices are properly conditioned.

3. Resources

- PIC-IoT WA Development Board Product Page
- AVR-IoT WA Development Board Product Page
- Get Started with MPLAB® X IDE and Microchip Tools
- AWS IoT Developer Guide
- IoT Provisioning Tool
- GitHub repositories with the IoT Boards' pre-loaded firmware:
  - For the PIC-IoT Development Boards
  - For the AVR-IoT Development Boards

4. Revision History

Revision	Date	Description
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 specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "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

Trademarks

ISBN: 978-1-5224-6655-0

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