Internet of Things

Updated Feb. 28.

Introduction

In this lab you will use a TI (Texas Instruments) CC3200 Wireless SoC (System on a Chip) microcontroller to create a IoT ("Internet of Things") device that publishes the RSSI (Received Signal Strength Indication) to a server using the MQTT (Message Queueing Telemetry Transport) protocol.

IoT

The decreasing cost of IC technology is making it feasible to add networking, typically wireless, to relatively low-cost devices such as appliances, lighting, sensors (e.g. thermostats), and many other devices. This has been termed the "Internet of Things" (IoT).

This lab is solely concerned with a study of a typical IoT implementation. However, it should be understood that connecting these devices to the Internet raises important questions of security, safety, reliability and privacy.

TI CC3200 Microcontroller

The TI CC3200 is a typical microcontroller. It includes an ARM Cortex M4 CPU, embedded SRAM, flash-erasable ROM, GPIO and serial interfaces to peripheral ICs. To support its target market of battery-operated devices it also has advanced power-management features such as integrated switching regulators and a μA-level sleep modes.

The CC3200 is also a wireless SoC – a combination of an application processor and a wireless interface. The IC includes an independent processor with its own memory and peripherals to handle the TCP/IP and 802.11 ("WiFi") protocols and the integrated 802.11b/g/n (WiFi) transceiver.

Another example of a wireless SoC is the Espressif ESP8266 that is priced at about $1 (the CC3200 is about $5).

Arduino and Energia

The Arduino project was designed to help students learn about microprocessors and programming. The project includes a simple IDE (integrated development environment) that cross-compiles a program (called a "Sketch") written in a subset of C++ on a host and downloads it to an Atmel AVR microcontroller. Arduino includes libraries (called "Wiring") to abstract and simplify interfacing to common peripherals.

The Energia project extends Arduino to TI microcontrollers. Arduino and Energia are meant for educational purposes. Larger projects typically use command-line tools (make, cc, git, ...) that operate on files. These tools allow building and testing to be automated by scripting and allow different vendors' tools to be used together.

MQTT

MQTT is a simpler alternative to the ubiquitous HTTP protocol. MQTT was designed for efficient transfer of small amounts of data to and from IoT devices. A client device can "publish" data by connecting to a "broker" server over TCP port 1883 and sending a "topic" and associated data. Client devices that have "subscribed" to these topics will then receive a copy of the data.

A message's "topic" is a string similar to a file system path. In order to communicate, two devices must agree on the broker to use, the topic name(s) and the semantics and syntax (e.g. text, XML, JSON, ...) of the content.

A broker typically distributes messages as soon as they are received. However, it's possible for clients to subscribe to a topic in a "persistent session" which allows them to connect to the broker intermittently.

Although the MQTT protocol is relatively simple, there are more details than can be covered here. For example, messages can be exchanged with different levels of reliability (QoS, Quality of Service) and mechanism are available to queue messages for de-

vices that are not connected or to generate notifications for devices that disappear.

Brokers allow devices to stay in sleep mode except when they need to communicate. This is important for battery-powered devices. In addition, much of the cost and complexity (e.g. authorization) can be centralized and shared among many devices.

Security is not part of MQTT. It can be provided by, for example, using TLS for encryption and signatures for authentication.

There are many public MQTT brokers you can use for testing (but not for this lab!) and many MQTT brokers that run on systems ranging from routers to "cloud"-based services.

Procedure

Install Energia, CC3200 Support and Host Drivers

Download and unzip the Energia IDE to a convenient location. If you are working on a lab PC this will probably be somewhere on the D: drive.

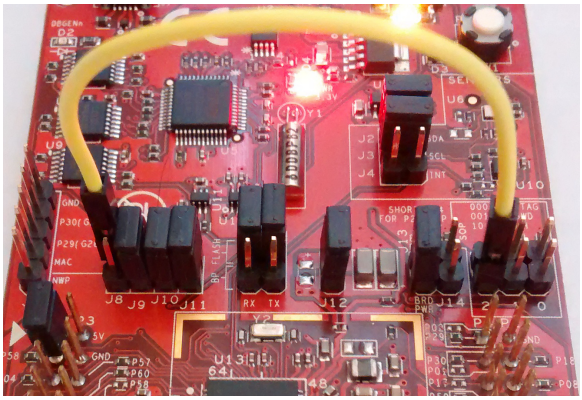
A BSP (Board Support Package) is a set of configuration files and libraries that support a specific microcontroller and peripherals. Typically this includes boot code that initializes the processor and memory. A BSP may also include libraries to provide interfaces to peripherals.

The CC3200 BSP can be installed by running the Energia IDE installed above and selecting Tools / Board: / Boards Manager .... and then selecting the Install button in the "Energia CC3200 boards" section. Note that on Windows this installs the BSP to **`/Users/.../AppData/Local/Energia15`** instead of the location where Energia was installed.

You may also have to install OS drivers so that the CC3200 board's USB interface will appear as a serial port on the host to allow uploading code and for it to be used as a virtual serial port.

Check and Install Jumpers

Get a CC3200 board from the instructor. The package should include a micro-USB connector and a jumper wire. Ensure the jumpers are configured as shown below:



including the jumper wire between TCK and SOP pin 2.

Connect the board to your host PC using the USB cable.

Build and Run the Blink Sketch

It is suggested you first build and run a simple example to ensure the software and hardware are configured correctly.

Start Energia and under the Tools menu select:

- Board: CC3200-LAUNCHXL
- Port: the serial port that's connected to the CC3200 board (check the Ports section in Windows' Device Manager if you're not sure).
- Programmer: dslite

Under **`File/Examples/01.Basics`** choose **`Blink`**. This will open up an editor window with the sample code.

Select **`Sketch/Verify/Compile`** (control-R) to compile the code (there may be warnings).

Select **`Sketch/Upload`** (control-U) to Upload the program to the CC3200's flash memory.

The red LED should now be blinking on and off every two seconds.

Add an MQTT Publisher

Save your code under a new name and add MQTT client code that does the following in the **`setup()`** function:

- connects over WiFi using the network name (ESSID) **elex7860** and the WiFi password **lab4-iot**[1]

and the following in the **loop()** function:

- creates a TCP connection to the host **test.mosquitto.org** (note the spelling!) at port **1883**
- creates an MQTT connection
- publishes a message using the (case-sensitive) topic **elex7860/**<*secret*> where <*secret*> is a unique string given to you by the instructor. The data should be an integer (probably negative) containing the current RSSI in dBm.
- calls the **yield(2000)** method of the MQTT client object. Since you are not subscribed to any topics this will simply result in a delay of 2 seconds.
- disconnects the MQTT connection
- disconnects the TCP connection

A <*secret*> is used because everything uploaded to this broker is public and can be read by anyone.

Compile, fix any errors and run the code on the CC3200. The instructor will display the messages being received by the broker. Inform the instructor when you see your topic being displayed so that you can get credit for completing the lab.

Lab Report

Your lab report should contain the code you used to publish your secret and a screen capture of the serial output showing the RSSI values.

Appendix - Eclipse Paho MQTT C++ Client

The Energia CC3200 libraries include a C++ MQTT client library. This library was adapted from the Eclipse Paho project for embedded systems. The documentation is sparse; the best is probably that published by the mbed RTOS project.

[1]The Energia networking libraries do not support the EAP authentication required by BCIT's wireless network so a WiFi access point will be set up during the lab for your use.

The Energia WiFi library is based on the Ethernet library. Both are similar to the corresponding Arduino libraries.

An incomplete program is provided below to reduce the time required to get your code working. It does not include (intentional) logic errors so it should be sufficient to read through the code, fix syntax errors and modify it to meet the above requirements. There are also several examples available in the Energia IDE under **Files/Examples/MQTT**.

```
/*
  lab4.ino - ELEX 7860 IoT MQTT example
  Ed.Casas 2019-2-27

  This code is incomplete and/or contains errors.
*/

#include <WifiIPStack.h>
WifiIPStack ipstack;  // IP via WiFi

// initialization: set up WiFi connection

void setup() {

  Serial.begin(115200);

  WiFi.begin( ESSID, WiFi password ) ;

  while ( WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(300);
  }

  while (WiFi.localIP() == INADDR_NONE) {
    Serial.print("+");
    delay(300);
  }

  Serial.print  (String("\nConnected to ") + WiFi.SSID() + " with IP=");
  Serial.println(WiFi.localIP()) ;
}

#include <Countdown.h> // timeouts
#include <MQTTClient.h>

// continously publish RSSI

void loop()
{
  int rc = 0;
  char data[80] ;

  MQTT::Client<WifiIPStack, Countdown> client(ipstack);  // client

  rc = ipstack.connect( brokerhostname , MQTT port ); // connect to broker
  rc || Serial.println("TCP connect returned " + rc) ; // should return true

  rc = client.connect();
  rc && Serial.println("MQTT connect returned: " + rc);

  snprintf(data, 80, "RSSI=%d", WiFi.RSSI()) ;
  Serial.println(String("publishing:") + data);

  rc = client.publish("my/important/topic", data, strlen(data));
  rc && Serial.println("MQTT publish returned: " + rc);

  client.yield(2000) ;

  rc = client.disconnect() ;
  rc && Serial.println("MQTT disconnect returned: " + rc);

  rc = ipstack.disconnect(); // disconnect
  rc && Serial.println("TCP disconnect returned: " + rc);
}
```