Lab 4C Internet of Things

This laboratory assignment accompanies the book, Embedded Systems: Real-Time Interfacing to ARM Cortex M Microcontrollers, ISBN-13: 978-1463590154, by Jonathan W. Valvano, copyright © 2017.

Goals

- Implement a system that connects to the internet via an IEEE 802.11 – **Wifi** module, CC3100
- Use DNS to convert name to IP address
- Configure a smart object that can retrieve data from a weather server using TCP
- Design a smart object that can store data onto an internet server using TCP
- Implement a web server to log data from your smart object

Review

- Valvano Section 11.4 on internet of things
- cc3100_getting_started_guide_swru375.pdf (in \ValvanoWareTM4C123\CC3100_docs)
- cc3100_boost_user_guide_swru371.pdf (in \ValvanoWareTM4C123\CC3100_docs\hardware)
- cc3100_boost_sch_rev3p3-a.pdf (in \ValvanoWareTM4C123\CC3100_docs\hardware)
- Instructions on how to create your own web server

Starter Project

- **CC3100GetWeather_4C123** in ValvanowareTM4C123 (**main.c** 9/2016 date or later)
- **ESP8266_4C123** in ValvanowareTM4C123 (9/16/2016 date or later)
  New projects can be downloaded at http://users.ece.utexas.edu/%7Evalvano/arm/

Background

Wifi is the name given to devices that communicate wirelessly employing the IEEE 802.11 standard. They typically operate in the 2.4 GHz Industrial-Scientific-Medical (ISM) unlicensed band that is shared with ZigBee (IEEE 802.15.4) and Bluetooth (IEEE 802.15.1) communications. The IEEE 802.11 standard describes how information is represented via radio frequency signals, i.e., the **physical** or **PHY** layer, and how communications are formatted and controlled, i.e., the **media access control** or **MAC** layer. Communication requirements beyond the PHY and MAC, e.g., flow control, error recovery, and routing, are handled at higher levels of the internet stack.

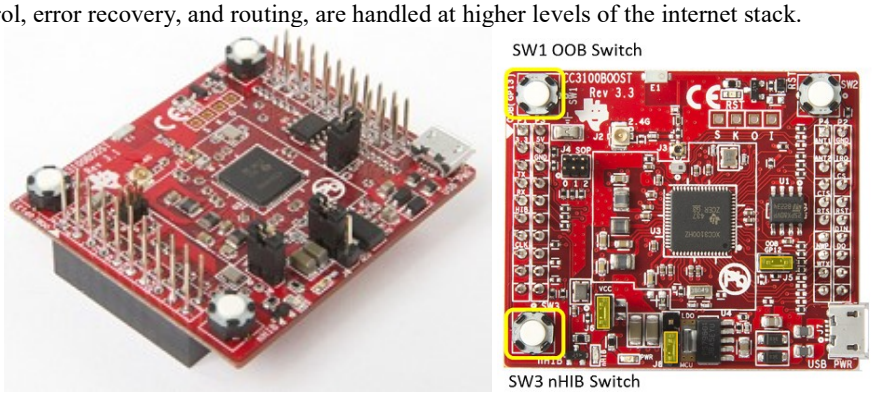

Figure 4.1. The TI CC3100 Wifi booster pack (CC3100BOOST).

Required Hardware www.ti.com

CC3100BOOST	Wifi Booster	$19.99
EK-TM4C123GXL	Tiva LaunchPad	$12.99

The emulation module (CC31XXEMUBOOST) is needed only if you wish to upgrade or reprogram the CC3100BOOST. The emulation module could also be used for situations where you wish to begin wireless programming before you have selected which microcontroller to use.

Specifications

This lab will create a "smart object" or "internet of things". Your system includes

- a PC, running PuTTY, connected to the TM4C123 through UART0 (optional for debugging)
- a TM4C123 LaunchPad running your software
- a CC3100BOOST Wifi booster pack connected to the internet
- a ST7735R or Nokia 5110 LCD showing measured data
- a sensor generating a voltage connected to an ADC (e.g., a 10k slide pot, or internal temperature sensor)
- a wireless access point, configured with or without security
- a web server, used to log data from your smart object.

You are free to define the "look and feel" of your system. Make it however you wish as long as you pull data from the weather server and push data to your server using TDP packets over the wifi link. Feel free to write your code using

the style found in the TivaWare driver or in the style found in the book. The starter project was derived from TI's TivaWare, so much of **CC3100GetWeather_4C123** is written in TivaWare style. You may perform this lab with Keil or CCS. You are allowed to use the ESP8266 with starter project **ESP8266_4C123**.

Interfaces

The TM4C123 bus clock will be 50 MHz. The PC is connected to the LaunchPad via UART0, which is embedded in the standard USB debugging cable. This link is used for debugging. The starter code uses 115200 bits/sec, 1 stop, no parity, and no flow control. Use the Windows **device manager** to determine the COM port used to communication with your LaunchPad. The CC3100/LaunchPad interface is shown in Figure 4.2 and Table 4.1. This interface uses GPIO, SSI, and UART ports on the LaunchPad.

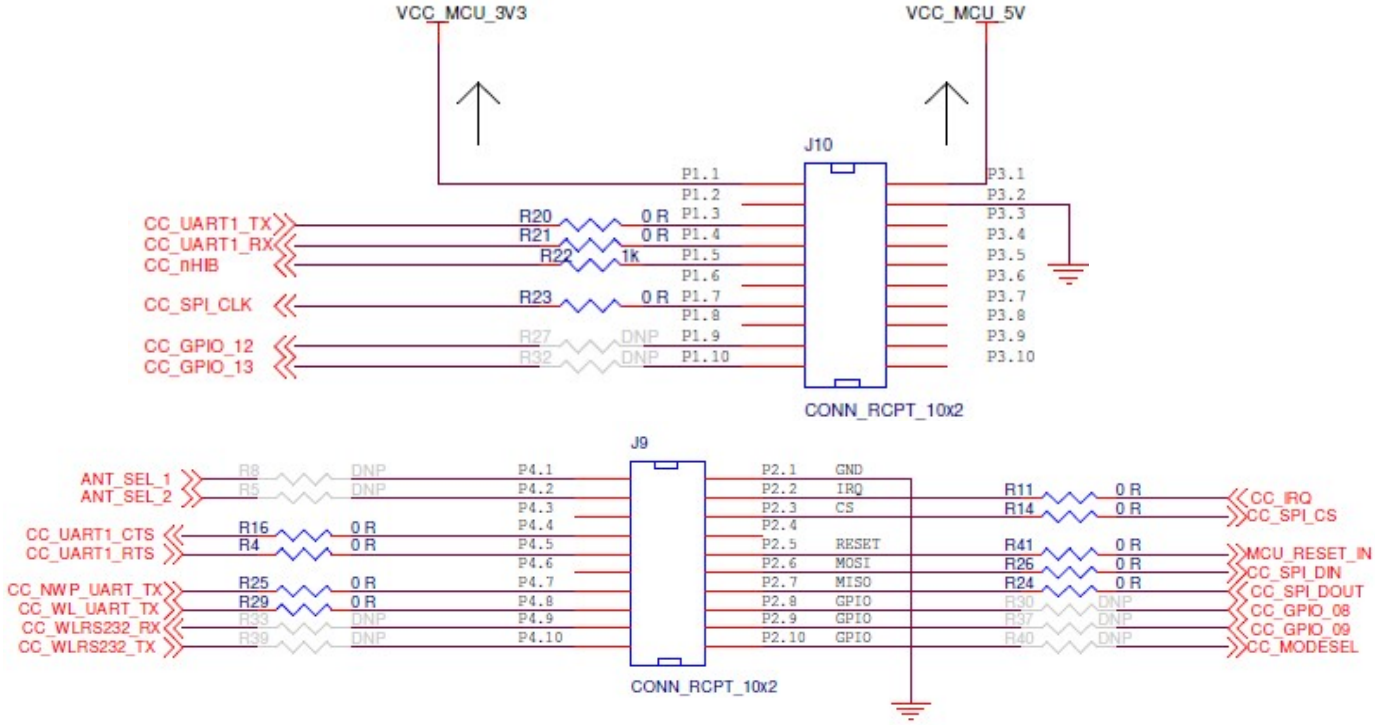

Figure 4.2. The CC3100-LaunchPad interface (DNP means do not populate).

Pin	Signal	Direction	Pin	Signal	Direction
P1.1	3.3 VCC	IN	P2.1	Gnd GND	IN
P1.2	PB5 SPI2_FSS	IN	**P2.2**	**PB2 IRQ**	**OUT (GPIO interrupt)**
P1.3	**PB0 UART1_TX**	**OUT**	P2.3	PE0 SPI_CS	IN
P1.4	**PB1 UART1_RX**	**IN**	P2.4	PF0 UNUSED	NA
P1.5	**PE4 nHIB**	**IN**	**P2.5**	**Reset nRESET**	**IN**
P1.6	PE5 UNUSED	NA	P2.6	PB7 SPI2_MOSI	IN
P1.7	PB4 SPI2_CLK	IN	P2.7	PB6 SPI2_MISO	OUT
P1.8	PA5 UNUSED	NA	P2.8	PA4 UNUSED	NA
P1.9	PA6 UNUSED	NA	P2.9	PA3 UNUSED	NA
P1.10	PA7 UNUSED	NA	P2.10	PA2 UNUSED	NA

Pin	Signal	Direction	Pin	Signal	Direction
P3.1	+5 +5 V	IN	P4.1	PF2 UNUSED	OUT
P3.2	Gnd GND	IN	P4.2	PF3 UNUSED	OUT
P3.3	PD0 UNUSED	NA	P4.3	PB3 UNUSED	NA
P3.4	PD1 UNUSED	NA	**P4.4**	**PC4 UART1_CTS**	**IN**
P3.5	PD2 UNUSED	NA	**P4.5**	**PC5 UART1_RTS**	**OUT**
P3.6	PD3 UNUSED	NA	P4.6	PC6 UNUSED	NA
P3.7	PE1 UNUSED	NA	P4.7	PC7 NWP_LOG_TX	OUT
P3.8	PE2 UNUSED	NA	P4.8	PD6 WLAN_LOG_TX	OUT
P3.9	PE3 UNUSED	NA	P4.9	PD7 UNUSED	IN (see R74)
P3.10	PF1 UNUSED	NA	P4.10	PF4 UNUSED	OUT(see R75)

Table 4.1. CC3100 connections to the TM4C123 LaunchPad (PB5 is enabled as FSS but not connected). Pins marked

in blue are used for communication between TM4C123 and CC3100.

Preparation

Part a) Watch the 10 short videos Yerraballi and Valvano created for the MOOC. The YouTube channel is at
https://www.youtube.com/playlist?list=PLyg2vmIzGxXGy6TnmO1tyR33HW6n9IvPr

Part b) Find the **CC3100GetWeather_4C123** project in ValvanowareTM4C123 and make sure the **main.c** file has a date of July 2015 (or later). Make a copy of this project that will become your Lab 4C solution. You should be able to compile this project. http://edx-org-utaustinx.s3.amazonaws.com/UT601x/ValvanoWareTM4C123.zip

Part c) On both **CC3100GetWeather_4C123** and your Lab4C project, modify lines 97 98 99 so that the smart object can connect to your access point. This is how I connect to my cell phone.

```
#define SSID_NAME  "ValvanoJonathaniPhone"
#define SEC_TYPE   SL_SEC_TYPE_WPA
#define PASSKEY    "y2uvdjfi5puyd"
```

There are nine choices for the **SEC_TYPE** (see lines 141-149 of **wlan.h** file). The above code selects WPA encryption, but I have also used a router with no encryption by setting the type to **SL_SEC_TYPE_OPEN**.

Correction. Look at the line right after the **while(1)** in **main.c**. It should be

```
  while(1){
    strcpy(HostName,"api.openweathermap.org"); // works 9/2016
```

not

```
  while(1){
    strcpy(HostName,"openweathermap.org");
```

Part d) Use *openweathermap.org* and search for the current temperature in Austin Texas. Figure 4.3 shows the contents of the TCP payload, **Recvbuff**, which is returned by *openweathermap.org* when a smart object queries it for Austin's weather. Observe the Austin temperature data highlighted in yellow. Write a C function that extracts the Austin temperature from this buffer. Create a fixed-length string like **"Temp = 8.568 C"**

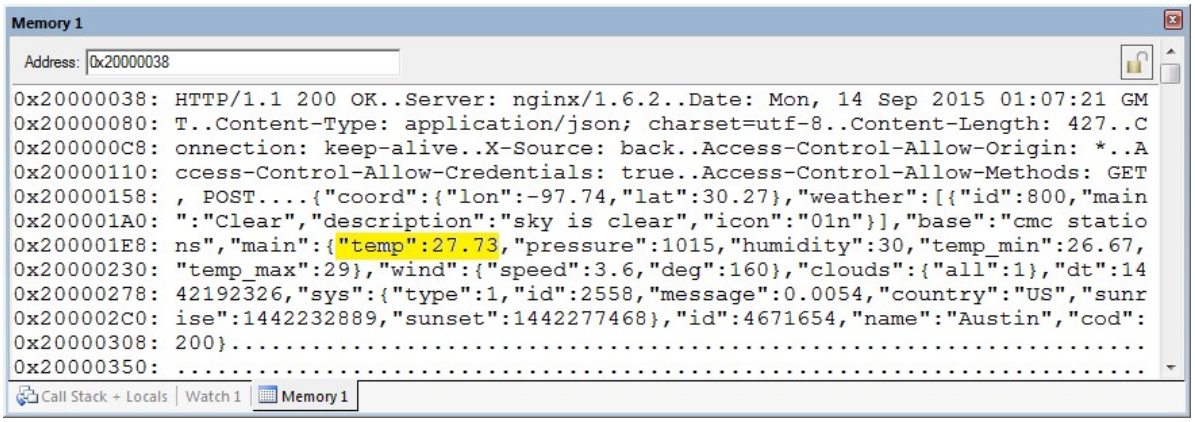

*Figure 4.3. Example **Recvbuff** response from the openweathermap.org server.*

Part e) If you have an ST7735R add your ST7735.c and ST7735.h driver files to the project, otherwise you can interface a Nokia 5110. After receiving the weather information, run the function in Part d) to parse the TCP packet in **Recvbuff**, and display the Austin temperature on the LCD. Each time you press and release the button the temperature data should update.

Part f) Write software to sample the ADC once using 64-point hardware averaging and calculate a physical parameter with units. Examples include but are not limited to a voltmeter, an ammeter, a distance measurement (slide pot like EE319K), or an internal temperature measurement (see section 13.3.6 and register ADCSSCTL3 in the data sheet). Create an ASCII string similar to

```
Voltage~1.23V
Current~123uA
Distance~1.23cm
Int_Temp~41C
```

Avoiding characters **= & % ? \n \r** and **spaces** will simplify parsing in the server. Add code to the main loop, so the

ADC is sampled, the parameter is calculated, the string is created, and the results are displayed on the LCD each time the switch is pressed.

Part g) Follow the instructions to create your own data logging server on google.
http://users.ece.utexas.edu/%7Evalvano/EE445L/Lab4GoogleAppEngineServerTutorial.html

Procedure
Part a) You will need to debug code with a wifi access point for which you know the SSID, type of encryption, and password. First verify the hardware configuration is operational by running the original **CC3100GetWeather_4C123** project with lines 97-99 modified to connect to your access point. You can observe the returned weather in the debugger or using PuTTY.

Part b) Debug your code at parses the **`Recvbuff`** and displays the Austin temperature.

Part c) Debug your code at samples the ADC, calculates a parameter, creates a string, and outputs to the LCD.
Part d) Go to *embedded-systems-server.appspot.com* and observe the look and feel for the EE445L data logging server. There are fields for where the data was collected, who collected the data, and the data itself.

Part e) Write code that gets called once in your main loop after the ADC is sampled and converted to a string. Log this information onto the server using these steps in a similar manner to getting weather data:
1) Use **`sl_NetAppDnsGetHostByName`** to find the IP address of *embedded-systems-server.appspot.com*
2) Use **`sl_Socket`** to create a socket
3) Use **`sl_Connect`** to open the socket
4) Use **`sl_Send`** to send a TCP packet to the server
5) Use **`sl_Recv`** to receive a response from the server
6) Use **`sl_Close`** to close the socket

The format of the TCP payload you will send is

```
GET /query?city=<city>&id=<name>&greet=<data>&edxcode=8086 HTTP/1.1\r\nUser-
Agent: Keil\r\nHost: embedded-systems-server.appspot.com\r\n\r\n
```

where <city> specifies where the data was collected, <name> includes you and your partner, and <data> is the measurement. If you wish to include spaces use **%20** and use **%3D** if you want an equals sign. For example

```
GET
/query?city=Austin%20Texas&id=Jonathan%20Valvano&greet=Int%20Temp%3D21C&edxcod
e=8086 HTTP/1.1\r\nUser-Agent: Keil\r\nHost: embedded-systems-
server.appspot.com\r\n\r\n
```

When you create your own server using the instructions, it will not have an **`edxcode`**. So when working with your server, you can skip the **`edxcode`**.

Part f) Instrument the system so you can measure the time it takes to collect weather data from the internet and how long it takes to store data on your server. Run it 10 times and calculate the minimum, maximum, and average times and record the number of lost packets (my experience is that it either works and there are no lost packets, or it is broken and every packet is lost).

Deliverables (exact components of the lab report)
A) Objectives (1/2 page maximum)
B) Hardware Design (only if needed for the sensor)
C) Software Design (a hardcopy software printout is due at the time of demonstration)
D) Measurement Data
Percentage of lost packets. Basically how reliable is the system (assuming you have a connection to the AP)
Minimum, maximum, and average times from 10 transmissions to openweathermap.org
Minimum, maximum, and average times from 10 transmissions to your server
E) Analysis and Discussion (1 page maximum)
1) In the client server paradigm, explain the sequence of internet communications sent from client to server and from server to client as the client saves data on the server. Assume the client already is connected to the wifi AP and the client knows the IP address of the server.
2) What is the purpose of the DNS?
3) What is the difference between UDP and TCP communication? More specifically when should we use UDP and

when should we use TCP?

Checkout
Demonstrate that your system can display data and log measurements on the your server. (It is easy to spoof the server so you will have to show your system running to the TA)
Upload your software as instructed by your TA.

Extra credit
Perform at most one of these for an extra 10%. Extra credit is in addition to the regular lab.
0) Extend your data logging server so that it does something with the data.

1) Looking over the list of starter projects listed in book Section 11.4.8, find something interesting and propose it to your TA. Please get approval first, before spending a lot of time.

2) Combine Labs 3 and 4 and use the GMT in the weather packet to synchronize their alarm clock. In order to make it all fit within 32k limit you can use CCS or simplify the clock.