

# Lab1: Development Tools Tutorial and Lab Exercise

Alexander Arasawa, Jiawei Zheng

June 11, 2022

EEC 172

Professor Soheil Ghasihafezi

# 1 Check off from TA:

UNIVERSITY OF CALIFORNIA, DAVIS  
Department of Electrical and Computer Engineering  
EEC 172 Winter 2019

## LAB 1 Verification


Team Member 1: Alex Arasawa

Team Member 2: Jiawei Zheng

Section Number/TA: A03/Ryan Tsang

Demonstrate your working application to your TA:

Demonstrate the application program running from flash to your TA

Date	TA Signature	Notes
9/30		All done!

## 2 Introduction

The goal of this lab is to learn about and get used to the tools Code Composer Studio (CCS), CCS UniFlash, and the TI Pin Mux Tool for programming a CC3200. The goal is also to learn how to set up a terminal emulator program, PuTTY or Tera Term, for input and output to a CC3200. The hardware we used includes a CC3200(the machine we want to program), a PC(to use the APIs and available softwares to program the CC3200), and a USB cable to connect the CC3200 to our PC. In part I of the lab, we looked at two example CSS projects, Blinky and Uart\_demo, to get a gist of how we can manipulate LEDs on the CC3200 using GPIO signals and how we can communicate with CC3200 using UART signals. In part II of the lab, we apply what we learned from part I to create our own program which basically configures the CC3200 into a finite state machine. Finally, in part III, we learned how to flash a program into the onboard serial Flash chip of a CC3200 so that program can run without us downloading the program from CSS into RAM in debug mode.

## 3 Background

CCS software is an IDE for configuring TI's microcontrollers and embedded processors. The software contains many tools to help with creating and debugging applications: CSS includes an optimized C/C++ compiler, a code editor, a project build environment, a debugger, and many other features.

CCS Uniflash is a standalone tool used to program flash memory of TI's microcontrollers and program flash memory of Sitara processors.

TI Pin Mux Tool is a software used for configuring pins, peripheral signals, and other components of a system.

PuTTY or Tera Term are software terminal emulators that supports SSH, kerberos, telnet, and serial port connections. Serial port is the one we will be using for this lab since the CC3200 is connected to our PC with a USB cable.

References:

<https://www.ti.com/tool/CCSTUDIO>

<https://www.ti.com/tool/UNIFLASH>

<https://www.ti.com/tool/SYSCONFIG>

## 4 Goals/Objectives

### 4.1 Part I. Blinky example program

Modify the Blinky example program to make the LEDs blink faster.

What you should learn from this task:

- How to build and debug a simple program for CC3200.
- How does the Blinky program work.
- How does the Blinky program turn on and off the LEDs on the launchpad using GPIO signals.

### 4.2 Part I. Uart\_demo example program

Run the program and verify that it echoes what is typed into the terminal window.

What you should learn from this task:

- How does the Uart\_demo program work.
- How does the Uart\_demo program print messages to the terminal through UART signals.
- How does the Uart\_demo keep cycling to receive inputs from the user to echo those inputs back as outputs to the terminal.

### 4.3 Part II. Modify Blinky to meet the below application requirements

Display usage instructions to the terminal when the program starts. When SW3 is pressed, have the LEDs flash in binary counting from 000 - 111 continuously. When SW2 is pressed, blink all LEDs on and off. The console should print which switch has been pressed and should not print again until after the other switch has been pressed.

What you should learn from this task:

- How to use TI Pin Mux Tool to configure desired GPIO signals.
- How to poll switches on the CC3200 using GPIO signals.
- How to set values of pins on the CC3200 using GPIO signals.
- How to write a basic algorithm for displaying binary sequence 0 - 7 in a finite state machine.

### 4.4 Part III. Using CCS UniFlash

Use CCS UniFlash utility to flash your application created in part II into the onboard serial Flash chip of CC3200 so the program can run without downloading the program from CSS into RAM in debug mode.

What you should learn from this task:

- How to flash a program into the onboard serial Flash chip of a CC3200 board.

## 5 Methods

### 5.1 Part I. Blinky example program

Inside the LEDBlinkyRoutine(), we decreased the value within the MAP\_UtilsDelay(8000000) function to decrease the delay in which the CC3200 turns on and off LEDs. If we want 2x, it would be  $8000000/2 = 4000000$  and if we want 10x, it would be  $8000000/10 = 800000$ .

### 5.2 Part I. Uart\_demo example program

Using PuTTY, we made a serial connection from our PC to the CC3200. The only thing we had to configure was to set the baud rate(how much data to read per second) to 115200 so it matches up with how much data the CC3200 writes per second. Then, we verified that the program does indeed work by sending inputs to the CC3200 and checking the terminal.

### 5.3 Part II. Modify Blinky to meet certain application requirements

First, we follow the instructions in the lab manual to configure GPIO and UART signals with TI pin mux tool so we can actually use those signals in our program.

Then, we made many changes to main() and the LEDBlinkyRoutine() to make the CC3200 become our desired finite state machine.

We imported uart\_if.c so we can use the same API that Uart\_demo uses to print messages to the terminal. The Message() function provided by the library prints characters to the terminal until the end of string input by implicitly calling on MAP\_UARTCharPut() multiple times to send character by character to the console through the UART signals. Thus, we just used this function to print the desired usage instructions in main() so on initial boot, the CC3200 will display those usage instructions to the terminal.

Then, for satisfying the SW3 and SW2 behaviors, we reused the LEDBlinkyRoutine() in main(). We figure out that there are basically 10 states our CC3200 could be in: -1(an invalid state), display 0 with LEDs, display 1 with LEDs, display 2 with LEDs, display 3 with LEDs, display 4 with LEDs, display 5 with LEDs, display 6 with LEDs, display 7 with LEDs(which goes back to display 0 with LEDs state naturally), and another display 7 with LEDs state(which goes back to display 7 with LEDs state naturally). We assigned integers -1, 0, 1, 2, 3, 4, 5, 6, 7, and 8 to the above states respectively. The behavior for when SW3 is pressed will go through states 0, 1, 2, 3, 4, 5, 6, 7, then restart from 0. The behavior for when SW2 is pressed will go through states 8, 8, 8, or keep looping through state 8.

On call to the LEDBlinkyRoutine(), the CC3200 gets stuck in an infinite while loop that initially polls SW3 and SW2, change states based on which button was pressed, turns on/off LEDs according to the current state, and then updates the current state to the appropriate state for the next iteration. We also initialized an int variable outside of LEDBlinkyRoutine() called "state" to keep track of the current state of the FSM. The "state" variable is initialized to -1 because the CC3200 should initially be in an invalid state and not display anything with LEDs when no buttons have been pressed yet.

At the beginning of the while loop, we poll the two switches using GPIO signals by calling the GPIOPinRead() function. After, we have if-loops to update the current state of the finite state machine. If the FSM is currently in invalid state and no switches are high on poll, the FSM will remain in invalid state. If SW3 was high, we write low to PIN 18 using GPIO signals by calling GPIOPinWrite(). If SW2 was high, we write high to PIN 18 using GPIO signals by calling GPIOPinWrite(). If SW3 was high and we are currently not in state 0, 1, 2, 3, 4, 5, 6, or 7, we would print "SW3 pressed" to the terminal and set the current state of the FSM to 0. If SW2 was high and we are currently not in state 8, we would print "SW2 pressed" to the terminal and set the current state of the FSM to 8. After the if-loops to update the state, we have another loop for displaying the

appropriate LEDs according to states and updating the current state to the appropriate state for the next iteration. The condition to enter this loop is that the current state is not invalid or -1 because we should not display anything for the invalid state. Inside this loop, we would first turn on/off the LEDs corresponding to the current state number using GPIO signals with `GPIO_IF_LedOn()` and `GPIO_IF_LedOff()` functions. For example, for state 7 and state 8, we turn on/off all the LEDs. Inside this loop and after displaying the LEDs, we would update the state appropriately for the next iteration. For example, for states 0, 1, 2, 3, 4, 5, and 6, the next states would be 1, 2, 3, 4, 5, and 7 respectively. For state 7, the next state would be state 0 (this is the continuous sequence 0 - 7 for SW3). Finally, for state 8, the next state would be still state 8 (this is the keep flashing all LEDs sequence for SW2).

## 5.4 Part III. Using CCS UniFlash

We follow the instructions provided in the lab manual to flash our program in Part II into the flash memory of our CC3200.

## 6 Limitations

For Part II of our lab, since we have not learned how to do interrupts yet, the user could press and release buttons during `MAP_UtilsDelay()` and we would not be able to poll for sw3 or sw2 and redirect the CC3200 to the next correct state.

## 7 Discussion

One of the difficulties that we faced was figuring out the inputs for the `GPIOPinRead()` and `GPIOPinWrite()` functions so we can poll switches and modify PIN 18. We found out we can get this information from `pinmux.c`. For example, SW3 was PIN 4 so we look at `GPIODirModeSet(...)` in section "Configure PIN\_04 for GPIO Input" for the appropriate inputs for polling SW3 with `GPIOPinRead()`. Another difficulty that we faced was figuring out the states of the CC3200 FSM in part II. Drawing an FSM diagram was very helpful and from the FSM, the algorithm for the program of part II came very easily.

## 8 Conclusion

This lab aided in learning how to setup CSS projects and configure software to get programs that we write to build and run on the CC3200 board. Also how to modify example programs, for example, to lower the delay in Blinky to speed up the frequency of the blinking LEDs. Using `UART_demo` example we learned how the connection between the board and computer relies on a serial port. In addition, we can display messages on that port and take input on the serial port console. We learned how to use the TI Pin Mux Tool to map physical components of the board to pin variables for the GPIO. Combining all of what we learned we modified the Blinky example program to poll switches on the CC3200 and flash different LED patterns based on what switch was pressed. Lastly, we learned how to use UniFlash to upload our program onto the flash memory of the circuit. This allowed it to run independently of Code Composer.