EEC 172 Lab 1: Development Tools Tutorial | Section: A03

Manprit Heer
912839204
mkheer@ucdavis.edu

Cathy Hsieh
912753571
cchsieh@ucdavis.edu

OBJECTIVE

This lab covers the basic software development tools, including Code Composer Studio (CCS), CCS UniFlash, and the TI Pin Mux Tool. Split into three parts, this lab steps through various lab technologies while allowing the user to familiarize ourselves with the lab equipment.

INTRODUCTION

In the first part, we test two example programs in CCS in order to ensure full functionality of our CC3200 Launchpads. The first program, when run, blinks red, orange, and green LED pins in a continuous sequence. The second program verifies that we are connected to the launchpad through COM port by echoing back input text coming from the keyboard.

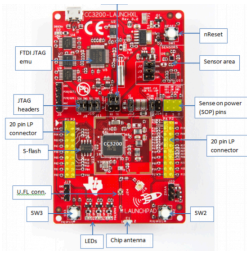
In the second part, we develop a simple program that interfaces to a console window and uses switches and LED pins as inputs and outputs respectively. This part requires for us to configure a signal that allows to control the switch inputs. In this part we use the TI Pin Mux Tool in order to configure the pins due to its easy-to-use interface and implementation method. We modify our program code to be able to count from 000-111 continuously through the three LEDs, and for the LEDs to simply blink ON and OFF when the other switch is pressed. In this part we become familiar with the CC3200 Peripheral Driver Library APIs, which allow us to communicate with the launchpad's interface directly.

The third part requires for us to write our program into the external flash memory component of the launchpad. Since we do not want to have to reload our program onto the ROM every time, we utilize the non-volatile feature of the flash memory component to store the program so that when we soft reset the pad, we are able to restart the program rather than having to take the additional steps to running the program.

MATERIALS AND METHODS

The equipment needed for this lab are listed as follows:

1. CC3200 LaunchPad (CC3200-LAUNCHXL)
2. USB Micro-B plug to USB-A plug Cable
3. Oscilloscope, jumper wires



CC3200 LAUNCHPAD[1]

In Lab 1, we utilize Code Composer Studio to be able to interact with the launchpad's functionality. In order to access the Launchpad's console, we emulate a terminal window through PuTTy. W also engage with the TI Pin Mux Tool to connect our physical CC3200 pins with the virtual pins in the studio. We utilize CCS UniFlash to be able to store programs in the built-in flash drive of the launchpad.

PROCEDURE

Part I. Testing Example Programs in Code Composer Studio

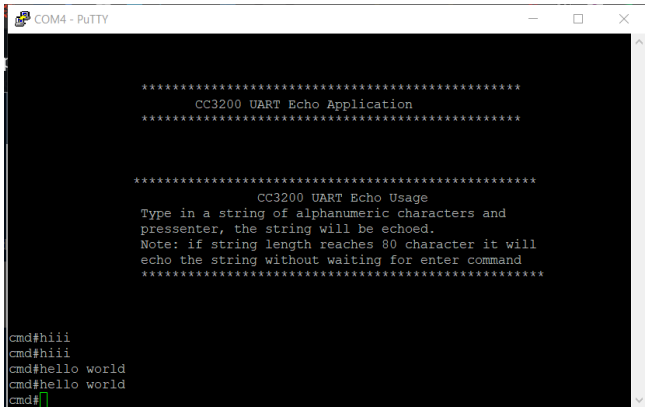
Blinky Example Program

The purpose of Blinky was to initially test out and get familiar with Code Composer Studio. We encountered a problem during this example due to faulty launchpads. Both devices refused to connect and be recognized by CSS as we would encounter an error stating, "Error connecting to the target." After numerous attempts at troubleshooting the problem, such as switching computers, replacing wires, and changing ports, we decided it best to replace the launchpads entirely.

After replacing the launchpads, we were able to successfully import the given example program and launch it on the launchpad. After following the tutorial, we build and run the project to find the red, yellow, and green LEDs blink in a continuous sequence. After making the simple modification in our code, we were able to make the lights blink faster by reducing the delay time between each blink.

Uart_Demo Example Program

Uart_demo program simply echoes the string that the user inputs into the console. The objective of this example is to ensure that we become familiar with the process of using PuTTy to access the device's console window.



LAB I, PART I | UART ECHO USAGE DEMO

The process of the second example program was much more smooth, and there were no errors along the executing of this part.

[1] https://www.researchgate.net/figure/TI-CC3200-Launchpad-board_fig1_312559421

Part II. Lab I Application Program Exercise

In this part, our objective is to devise a program that performs different functions depending on the switch triggered. We started the program off by copying and pasting key elements from the given example programs that we'd need to complete the task at hand.

```
112  DisplayBanner(char * AppName)
113  {
114
115      Report("\n\n\n\r");
116      Report("\t\t *************************************************\n\r");
117      Report("\t\t        CC3200 %s Application       \n\r", AppName);
118      Report("\t\t *************************************************\n\r");
119      Report("\n\n\n\r");
120  }
```

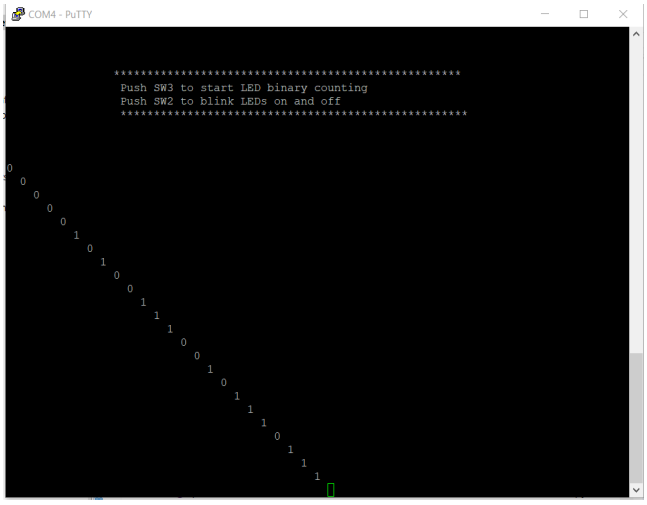
UART_DEMO | MAIN.C

From uart_demo, we extracted the commands as well as the files necessary to obtain a given print function for the console. Similarly, we extracted delay commands as well as APIs for GPIO LED manipulation.

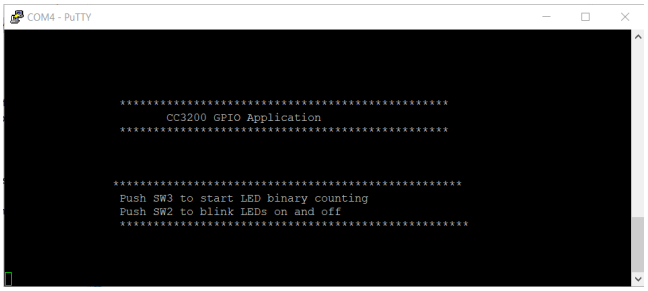
```
122      GPIO_IF_LedOff(MCU_ALL_LED_IND);
123      while(1)
124      {
125          //
126          // Alternately toggle hi-low each of the GPIOs
127          // to switch the corresponding LED on/off.
128          //
129          MAP_UtilsDelay(8000000);
130          GPIO_IF_LedOn(MCU_RED_LED_GPIO);
```

BLINKY | MAIN.C

To begin with the binary counter display, we decided to first establish an implementation method for the binary numbers. At first, we thought it was feasible to calculate each binary number as we go and output the values through the pins. However, the inability to retain numbers like "000" and "001" forced us to take a different approach and save the values from 000-111 as strings in an array. This would allow us to access each position of each string literal and connect its value to each pin, 0 for LOW and 1 for HIGH. To test our code, we print out our binary counter code to receive a successful print-out.

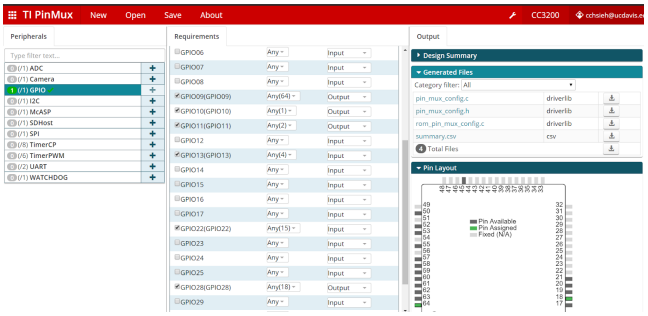


LAB I, PART II | BINARY COUNTER PROGRAM

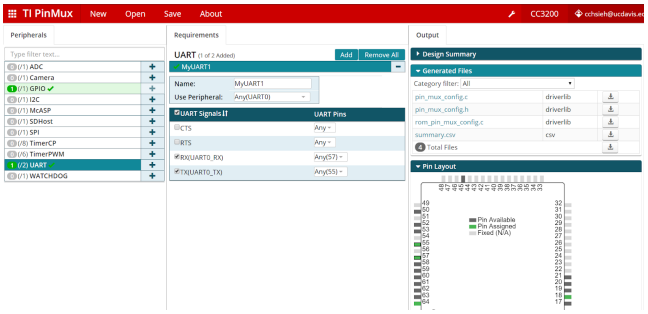


LAB I, PART II | START OF APPLICATION

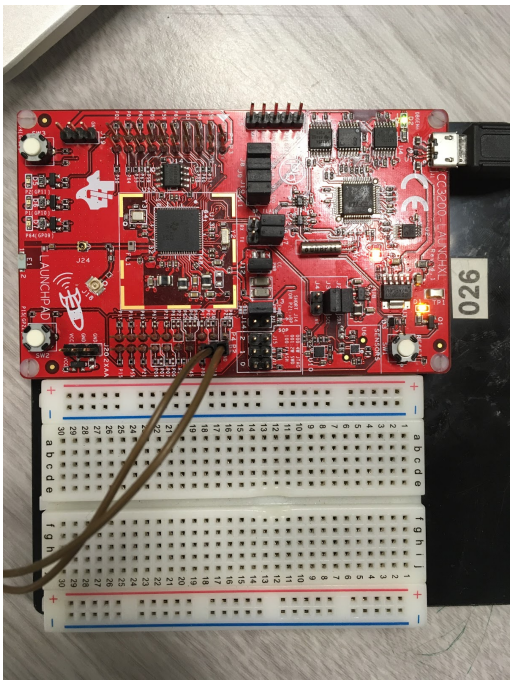
In order to direct each switch's input, we had to find the correct parameters of the pinRead and pinWrite APIs. These were found through the files given to us by the Pin Mux Tool. After locating and inserting the correct information, we were able to connect the pins with their respective addresses and use the right commands to turn the LEDs ON or OFF depending on the iteration.



LAB I, PART II | TI PINMUX TOOL (GPIO REQUIREMENTS)

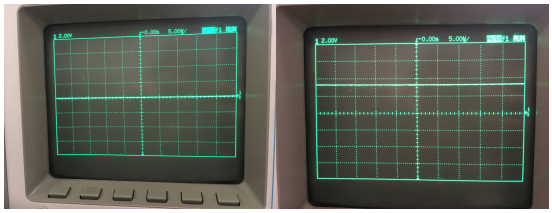


LAB I, PART II | TI PINMUX TOOL (UART REQUIREMENTS)



LAB I, PART II | CHECKING P18'S OUTPUT SIGNAL

Using the Oscilloscope, we tested out whether our code's program reflected the application specifications. Indeed, the code was correct and the readings were HIGH once SW2 was pressed.



LAB I, PART II | BEFORE AND AFTER SW2 TRIGGER

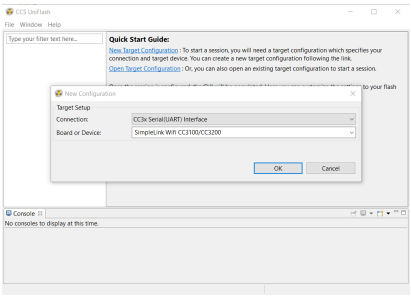
After establishing the pins, we continued to modify our C code to meet the program specifications. We created two functions for the binary counter application. flash_current() would be in charge of each individual LED's function, while binaryCount() was in charge of the sifting through the binary versions of 0-7 in that order.

```
120 void flash_current(char *binary){
121
122     int i;
123
124     for(i = 0; i < 3; i++){
125         if(binary[i] == '1'){
126             if(i == 0){
127                 GPIO_IF_LedOn(MCU_GREEN_LED_GPIO);
128             } else if(i == 1){
129                 GPIO_IF_LedOn(MCU_ORANGE_LED_GPIO);
130             } else {
131                 GPIO_IF_LedOn(MCU_RED_LED_GPIO);
132             }
133         }
134     }
135 }
136
137 void binaryCount(int i){
138     GPIO_IF_LedOff(MCU_ALL_LED_IND);
139
140     char *binary[8] = {"000","001","010","011","100","101","110","111"};
141     flash_current(binary[i]);
142     MAP_UtilsDelay(9000000);
143     GPIO_IF_LedOff(MCU_ALL_LED_IND);
144 }
145
```

Part III: Using CCS Uniflash

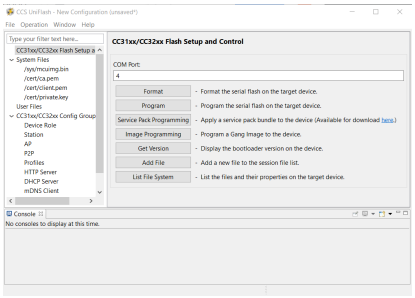
In this part, our objective is to learn how to program the external flash memory so that the program will remain in non-volatile memory through power cycles. Then we will load our part II program onto the external flash device on the CC3200 LaunchPad.

After launching the UniFlash tool, we started by creating a new configuration with "CC3x Serial(UART) Interface" for the connections and "SimpleLink Wifi CC3100/CC3200" for the board/device.



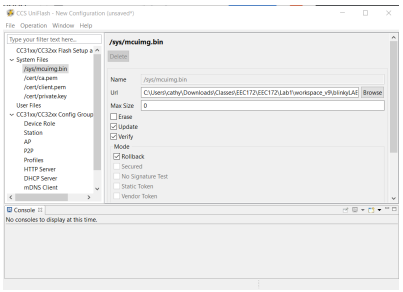
LAB I, PART III | CCS UNIFLASH TOOL

After creating a new configuration, we checked our COM port by using Device Manager while the launchpad is connected with the computer. After entering the correct COM Port number into the text box, we selected the "/sys/mcumimg.bin" option on the System Files on the left of the screen. By doing so, the panel allows us to identify the compiled .bin file which will be loaded into the flash memory.



LAB I, PART III | CCS UNIFLASH TOOL (SETUP AND CONTROL)

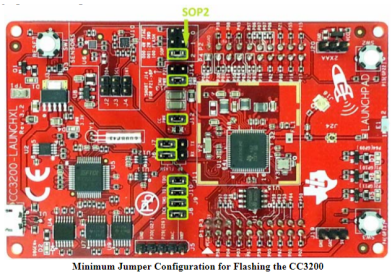
After selecting the correct .bin file and the options that we needed (*Update* and *Verify*). The UniFlash tool is ready to flash the CC3200 LaunchPad hardware.



LAB I, PART III | CCS UNIFLASH TOOL (SETUP AND CONTROL)

Before loading the .bin file into flash memory, we disconnect the launchpad from the computer and place a jumper on SOP2. Once all the jumpers are in the correct place, we connect the launchpad

back to the computer, and return to the “Flash Setup and Control” panel to run the Program.



Minimum Jumper Configuration for Flashing the CC3200

To check if we had successfully programmed the board, we first disconnected the launchpad and removed the SOP2 jumper, and reconnected the board to the computer. At this point, our board wasn’t able to successfully run the program. We tried to troubleshoot the problem by going back to the previous step and redo the part again, replacing wires, and using different launchpads. We still could not run our program successfully. We decided to try to run the Blinky program instead of our own, and the board was able to load the program successfully. After noting this success, we decided to then load our own program again, we got the same results as before, and we were prompted to move on, as this error was not vital to the objective of the lab.

SUMMARY

After completing the lab 1 tutorial for Code Composer Studio, CCS UniFlash, and the TI Pin Mux Tool, we ended with a better understanding of how CC3200 Launchpad communicates with the software programs and how to modify the program to display what was expected. This lab also introduced us to possible bugs that can arise in the process as well as strategies to tackle those error messages. An example of this would be having faulty equipment. Whenever the CCS system could not detect our launchpads, we would start troubleshooting around the wires and ports, as that is where we had the most success in previous instances.

Although we encountered numerous obstacles throughout the tutorial, after numerous attempts at troubleshooting, we successfully solved the conflicts. On the other hand, there were several areas that we could have improved to increase the efficiency of troubleshooting. In one of the parts that we had trouble compiling the program, if we had read the error messages more carefully and tried to understand the underlying issue, we could have saved time trying to debug the program. For example, we were having trouble running the program because we didn't compile it in the correct order. We could also have improved by reading the instructions and questions more carefully. If we had finished reading all the instructions before starting to program, our solution code would not have had to be modified due to lack of functionality.

Overall, we were able to understand the basic level of how CC3200 Launchpad works as well as the software programs and how they connect. We are now strongly equipped for the labs to come.

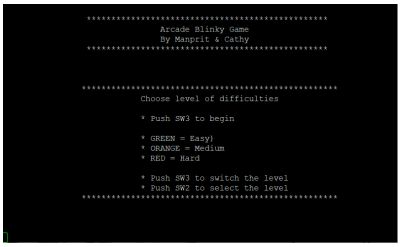
LAB EXTENSION: ARCADE BLINKY GAME

We went above the lab requirements and utilized the tools and knowledge that we acquired from lab 1 to create a mini arcade game. In our game, the player has to stop the light with a switch at a specific LED light to win the game. The player can also choose the level of difficulty to make the game more challenging. Our inspiration came from the Cyclone arcade game found in most classic arcades.

CYCLONE ARCADE GAME[2]

One problem we came across was naturally the response time of the user hitting the button at the correct LED flash. Although it did make it harder for the player to hit, thus making the game itself a challenge, it was an unintended function that we wanted to fix. In attempts to find a fix, we went back into our lab code and tried to place the delay() function in different locations to no avail. The TA was then able to explain the concept of interrupts to us and how that functionality could have solved the problem we had spent so much time trying to figure out.

When the game is loaded, it will display a main page at the terminal, including the title and the instructions on how to start the game. There are a total of three levels of difficulty which player can choose from. The player can press SW3 button to switch the level of difficulty, and press SW2 button to select the level. This functionality, of course, changes once the player has chosen and is ready to play.



ARCADE BLINKY | MAIN PAGE

Once the player has selected a level, another game instruction will display on the terminal that says "How to play". When the player is ready, they can press SW2 button to start, which then displays a countdown on the terminal before starting the game.

[2] https://www.icegame.com/category/93/cyclone

ARCADE BLINKY | INSTRUCTIONS PAGE

When the game starts, the LED lights will light up one at a time with different speeds and patterns depending on the level of difficulty. Once the player hits the light at the right spot, a "YOU WIN" message will display on the terminal, and the game ends and main page will show up again.

ARCADE BLINKY | “YOU WIN” AND RESET