EEC 172, A03
Lab #1: Learning how to use the CC3200 Launchpad
Joseph Kong, Ming Cheng

Objective (Goal)

The goal of this lab report is to learn about and familiarize ourselves with the hardware and software we will be using throughout this course. This includes hardware such as the CC3200 Launchpad and software such as Code Composer Studio (CCS), CCS Uniflash, and the TI Pin Mux tool. This was accomplished by first using example codes such as blinky and uart_demo and then writing a simple program that blinks the LEDs either in binary sequence or on and off in unison depending on which switch is pressed. In addition to this, another goal was to gain some experience reading and extracting data from datasheets to learn how to use the CC3200 Launchpad as well as learn how to use Uniflash to store the program in non-volatile memory of the CC3200 Launchpad so that the program can run at launch instead of debug mode.

Introduction

In this lab we learned how to use the software in conjunction with the hardware in order to make the CC3200 Launchpad turn the LED lights on and off and to make it to binary counting. The software we used was the Code Composer Studio integrated development environment where we put all the necessary files such as our pinmux configuration and our code into one project and compiled our project. Here we also wrote our code, debugged it, and ran it. The CCS Uniflash was used to program the external flash memory so that the program would remain in non-volatile memory through power cycles. The TI Pin Mux tool is used to select an appropriate pin multiplexing configuration for the application we want to create. Here we can select which pins we want to use and whether they should be used for input or output. This tool will generate the code to configure the pins for us, which we can then put into our project in CCS. Finally, the hardware we used was the CC3200 Launchpad is a board with a microcontroller on it and various pins and switches on it. In this lab, we used the board to blink the red, yellow, and green LEDs in a particular sequence that we coded up. We also used an oscilloscope to display the waveform of the electronic signal of one of the pins in order to check if that pin had the value we wanted.

Methods

Below is a high level description of the steps we used to successfully accomplish this lab.

1. In order to first learn how to use Code Composer Studio to code, we first imported an example project called blinky and changed properties such as compiler version and changing the floating point library to get rid of errors.

2. We got an idea of how the launchpad and CCS work together by connecting the launchpad to the computer. Then we ran a debug session in CCS, which enabled us to configure our target to the CC3200 launchpad.
3. After configuring the target to the launchpad, we can now run the program. We pressed the play button, and the code in the CCS IDE was run and we could see the output on the launchpad. We looked at the code and saw the corresponding output on the launchpad which enabled us to see how various functions worked and how the launchpad corresponds to the code. We then made some simple modifications to the blinky code in order to make the LED lights blink slower by increasing the number inside the `MAP_UtilsDelay` function in main.c
4. After this, we used the same method to import a second project into CCS called uart_demo. The purpose of this was to also use this in conjunction with the terminal. We determined which COM part was used to communicate with the launchpad by going to the device manager. Once we found out which COM port to use, we opened up this same COM port in the terminal. We configured the terminal to use a serial connection and set the baud rate to 115200. Then, we ran it and confirmed it worked by typing into the terminal and seeing the program echo it back.
5. For the next part, we created a new empty project which would interface to a console window and to switches and LEDS. The first thing we had to do was to configure the pins for our project using the TI Pin Mux Tool. For this lab, we used the cloud version of the Pin Mux Tool. For this project, we wanted to interface with the SW2 and SW3 switches, the P18 pin, and the red, yellow, and green LEDs. We determined which GPIO number corresponded to which pin number and what it interfaced. Then we added the necessary GPIO signals for our design and selected whether we wanted it to be input (switches), output (LEDs and P18).
6. Next we had to configure the UART peripheral. We configured UART0_RX to pin number 57 and UART0_TX to pin number 55. We then saved our design in our project directory. We then downloaded the generated pin_mux_config.c and pin_mux_config.h files and replaced the old pinumx configurations with our own design.
7. After we had our pinmux configuration, we started to modify the program code to meet the application specifications. Our task was to start LED binary counting if the SW3 switch was pressed and to blink LEDs on and off if SW2 was pressed. Furthermore, the output signal for P18 should be high whenever SW2 is pressed and low whenever SW3 is pressed. Our logic was to put the polling in an infinite while loop so it would always poll, and then check for whichever switch is pressed using bit masking to determine which switch was pressed. We had flags for determining whether we were doing counting or blinking. These flags were calculated based on which switch was pressed. To do the blinking, we simply had an increasing count which we would modulus by 2. If the result of the modulus was a 0, we turned all the lights off, if the result of the modulus was a 1

we turned on the lights on. For LED binary counting, we figured out that each of the LED had an address which were multiplies of two, and so we used the count multiplied by two which gave us the mask with which we could determine which LEDs got switched on. We then reset the count to 0 once it gets past 7. Finally, we used jumper cables to connect the launchpad to an oscilloscope. Upon pressing either SW2 or SW3, we used the oscilloscope to confirm whether P18 was high or low by looking at the waves it produced.

8. For the last part, we learned how to use the CCS Uniflash. We opened up the Uniflash tool and made a new configuration. We configured it to the CC3x Serial (UART) interface with SimpleLink Wifi CC3100/CC3200. We identified the correct COM port and load the correct .bin file into the flash memory. Before connecting the launchpad to the computer, we place a jumper on SOP2, then connect the launchpad to the computer and program it. We then check if it correctly flashed by disconnecting the launchpad, removing the jumper, and reconnecting to see if it runs right away.

Discussion

In this lab report, we mainly familiarized ourselves with the CC3200 launchpad, the pin muxing tool, and code composer studio. Pin muxing is a way to configure the pins to satisfy our specific project requirements. This is very useful because a lot of pins can have multiple functions, so pin muxing allows us to choose which pins we want to include in our project and what its specific function is. This pin muxing tool is also useful because it will implement our desired configurations without error by outputting a pin_mux_config c and header file that we can include in our project, whereas if we did it by hand it would be a lot more error prone.

UART stands for Universal Asynchronous Receiver/Transmitter and is used for the serial communication through the serial port. It can communicate with peripheral devices such as personal computers, and since there's no synchronized clock it depends on the baud rate to receive and transmit data properly. A UART module contains functions and configuration of the serial communication between the Launchpad programs and the terminal window. For example, our code included a header file called uart_if.h, and we used functions in it to initialize the terminal and sending string messages to the terminal.

One of the challenges we met was having a faulty oscilloscope. Our problem was that once we pressed a switch, the oscilloscope would only show high or low for a very short duration and then change back. We spent a lot of time trying to figure out what was wrong with the code until we finally realized it was a problem with the oscilloscope and not with the code. Another obstacle was figuring out that we had to unplug the launchpad from the computer before putting the jumper on SOP2 in order to program the flash. We didn't realize that the order in which we connected things mattered so it took us a while to figure that out.

Another challenge we met was to figure out how to use GPIOPinWrite and e GPIOPinRead functions which are crucial to our program. To understand these two functions, we read the CC3200 Peripheral Driver Library User's Guide and try to write simple lines of code to test if the functions work as our expected. After testing for a long time, we figured out that to use these functions, we had to use the correct address for each pin which we could find in the pin_mux_config.c file. We also learned that to set the pin high we have to set the value as the same as the address of the pin.

Conclusion

In conclusion, I think we successfully accomplished the goal of this lab, which was to familiarize ourselves with the hardware and software we will be using the rest of the quarter. We managed to complete all the requirements for the application and learned about different functions that we could use to program the board with. We learned how the CC3200 Launchpad, the pinmux tool, and Code Composer Studio work together to accomplish simple tasks such as binary LED counting and turning LEDs on and off.