EEC172, A01
Lab 1 Report 1/18/18
Kendall Lui and Mary Florek

Introduction

Embedded system development is different than regular programs written for general PCs this is because they are generally run on separate hardware than the development environment. The goal of the lab was to understand the development cycle for the CC3200 Launchpad development board. The main development tool for the CC3200 Launchpad board is Code Composer, an IDE that is used to write, build, and run programs on the CC3200 Launchpad board. Other utilities such as the Uniflash tool, PinMux Tool, and Tera Term, can be used to flash programs, configure pinouts, and connect to the board's serial port. All of these utilities were used during this lab to aid in our understanding of developing for embedded devices.

Goals

Our goal in this experiment was to become acquainted with the CC3200 Launchpad, the various software tools, and the programming SDK that we will be using throughout this class. This was accomplished by first using code examples provided by Texas Instruments (TI) and then by writing a simple program that would blink the board's onboard LEDs either through a binary sequence or on and off in unison, depending on which switch was pressed as given by a set of specifications. After implementing this program, we learned how to flash the board so that the program could run without being connected to the IDE and after the board went through power cycles.

Methods

The software tools used in this lab were Code Composer Studio for programming, CCS Uniflash for flashing programs to the onboard flash, and TI PinMuxTool to appropriately configure the CC3200 I/O pins. The first step was to load and compile a series of programs provided by TI to understand how to use the basic I/O as well as UART functionality. After running the "blinky" and "uart_demo" projects successfully. We looked at the code to understand the proper function calls for setup and function calls for manipulating I/O pins and LEDs.

After gaining a general understanding of the software implementations of the UART and I/O, the goal was to implement a program that displayed a binary sequence using the LEDs or blink the LEDs in unison based on switches pressed. Using the blinky project as a starting point, we first configured the appropriate chip pins to onboard I/O developer pins using the Pin Mux Tool. Next, a simple polling sequence was written to poll the buttons for selection. Upon polling a switch pressed a flag was set for desired switch and appropriate functions were called to display a binary sequence or blinking sequence. The main loop kept track of the flags, delay time between LED changes and switch polling. In addition, UART functionality was implemented using a similar implementation as the one found in the uart_demo project.

Our initial implementation of the binary counter sequence was a naive approach. This implementation was a series of if/else statements within a for loop that increments a counter for

each binary value. This approach worked, but we realized that it was very limited as it only allowed for a set number of binary values. If the program specification changed and needed to count using 4 or more bits, more statements would have had to be written in. This is not only time consuming for the programmer but also not efficient as the program must check every if-else statement with O(n_{bits}) complexity.

A second approach was designed and implemented the binary sequence by using the bit shift operator to effectively turn on the LEDs in a single function call. This required understanding how the GPIOPinWrite function works in detail. The second argument describes pins to be changed/written to during the function call since the configuration uses 0x2,0x4,0x8 for the three LEDs adding the three gets 0xe or 1110. Next, knowing that the value of a binary number for 3 bits is 000 - 111. We simply shifted the number by 1 effectively multiplying by two. This would map each bit to the last 3 bits or the activated I/O as described by 0xe. This only works for this example if the pins were not spaced by two this method would not work and further processing would have been required.

The blinky led implementation was a lot more straightforward a variable was sent to the function indicating whether to blink on or off the LEDs. This value (0 or 1) was multiplied by 0xe and given to the GPIOPinWrite function. This implementation uses only 1 function call.

We then used the Uniflash tool to save the program in the board's memory. This required using a compiled binary file ".bin" that is created in Code Composer Studio when the program is built. We then setup various configurations as specified in the lab handout and flashed the board. In order to verify that the program was properly flashed, the board was reset with the programming jumper removed. The program successfully booted and ran as expected.

Discussion

The first part of the lab was simply running and compiling the programs given. The blinky program simply showed how to use delays and use the onboard LEDs. The uart program was more interesting, it allowed us to use the UART device (Universal Asynchronous Receiver Transmitter) to communicate with the pc serial terminal but in theory could be used to connect to any other device using a UART serial implementation.

One interesting thing we found was the use of the Pin Mux Tool. Pin Muxing effectively takes the pins from the processor, in our case CC3200 and maps them to the specific pins/functionality that they are needed whether it be the UART module or the general I/O pins found on the PCB. These can be manually mapped programmatically but the Pin Mux Tool helps by eliminating a lot of this manual configuration. It allows us to use a GUI to select the pins we want and generates c programs and header files. We simply call the function PinMuxConfig and the I/O will be setup as specified in the pinmux_config files. In the future if we need to remap the I/O we can look in the pinmux_config to further understand how to write these configurations manually.

This lab really helped us understand the api available to us. We used the documentation to understand the GPIOPinRead and GPIOPinWrite functions to be able to write more efficient code. This took us quite some time experimenting. At the end, we were able to reduce our programs from 8 different if/else statements to a single binary line. The improvements to our program not only makes our code neater but also improves performance and robustness.

Most of the difficulties we encountered were in debugging the program. Initially, when we would try to build the program, it would encounter several errors. We went through the errors one by one and determined that some of the settings were incorrect. After searching online we found the floating point support flag was incorrect and set it to “FPv4SPD16” and were then able to successfully build the project.

We also encountered some trouble in trying to connect to the target, a problem that was solved by disconnecting Tera Term’s connection to the COM before running the program. This is interesting because the FTDI chip is a dual-port so the chip should have been able to handle both serial connections from Tera Term and Code Composer. Using other computers revealed this issue only occurred on the lab pc.

Conclusion

This lab project was ultimately a successful one, as we were able to get the program to work as described in the lab specifications. We also learned to flash programs to memory and were able to run our program independent of the programming PC and after power cycles. Many of the issues found in this lab were a result of being new to the development environment and SDK. With this lab complete we now feel confident using the software tools and SDK provided for the class and understand the embedded system development workflow.