

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

LAB 1 Verification

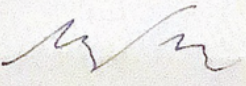
Team Member 1: Kunal Pai

Team Member 2: Steven To

Section Number/TA: A03

Demonstrate your working application to your TA:

Demonstrate the application program running from flash to your TA

Date	TA Signature	Notes
3/30		All done!

EEC 172 Lab 1 Report

Steven To, Kunal Pai

Introduction:

The objective of this lab was to provide insight into Code Composer Studio and its use in programming the TI CC3200 LaunchPad and utility programs like the TI Sysconfig (Pin Mux Tool) and CCS Uniflash that simplify the process of programming. To do this, the lab required the modification of two example programs: 'blinky' and 'uart_demo', to create a program that would instruct the CC3200 to display two different sequences of LED lights depending on which button on the board was pressed. The LED lights were configured as outputs and the switches were configured as inputs with the help of the TI Pin Mux Tool. Inputs on the board were read through using online documentation of the GPIOPinRead and the GPIOPinWrite functions on the pins as provided in the pin_mux_config files. The last part of the lab describes CCS Uniflash, which allows programs run without the Code Composer Studio.

Background:

Code Composer Studio is a program used to write code that interacts with the TI CC3200 board. It connects with the board via a Micro B USB cable. On loading a project into this Studio, the program within the project loads onto the RAM of the board, which can then detect the inputs and outputs as defined by the user. A downside to this is the fact that the program is loaded onto the volatile RAM, meaning that once the power is turned off, the program ceases to exist within the board.

TI Sysconfig or the Pin Mux Tool is a program used to define the pins that would be used for input and output of the board. This tool multiplexes the pins on the board so that peripheral functions can be linked to the GPIOs^[1]. It is also useful in the sense that it abstracts the process of pin-muxing – the user has to interact with drop down menus and select which pins acts as input or output or none and the program returns the configuration header files. Therefore, the chance for failure is less.

Uniflash is a tool used to overcome the downside of the loss of program data that occurs due to loading the program onto the RAM of the CC3200 board. It also makes the program run independently of Code Composer Studio, which increases portability.

The two modules used within this lab are as follows:

- GPIO: GPIO is an API that allows input/output signals within the configured pins of the board. This I/O interface is achieved through the use of various functions but in

this lab, we used GPIOPinRead and GPIOPinWrite. Both these functions return 1 on success, meaning if an input is read successfully or if an output (LED) is written successfully. These functions use a uIPort value and a ucPins value that is obtained from the initialization in the pin_mux_config files^[2].

- UART: UART is an API that connects the CC3200 board with a computer using serial connections. Adjusting the rate with which data is transmitted and received (baud rate), this connection can report and print out to terminal depending on the interactions with various pins on the board.

Goals:

Part 1: 'blinky'

- To explore the implementation of blinky and try to find methods to adjust the speed with which the LEDs turned on and off (ranging from two times faster to ten times faster). Prior to this, we needed to learn how to import example programs on CCS and debug/run them.

Part 2: 'uart_demo'

- To import/run the example program that receives alphanumeric keyboard input and echoes it as output on the PuTTY terminal that needs to be configured properly.

Part 3:

- To display the banner of the GPIO application along with the input pins and what they do using uart.
- To use GPIO functions to read inputs from two switches: SW2 and SW3:
 - o If SW3 is pressed, the three LEDs blink in a sequence from 000 to 111 with red being the least significant bit and green being the most significant bit. "SW3 pressed" is printed to the terminal.
 - o If SW2 is pressed, all the three LEDs blink on and off in unison. "SW2 pressed" is printed to the terminal.
- To set the output signal (P18) to high whenever SW2 is pressed and low whenever SW3 is pressed.
- To use Uniflash to allow the program to run without the CCS application.

Methods:

The following methodology was used in the code for making the LEDs blink faster:

1. `MAP_UtilsDelay(8000000);` in `void LEDBlinkyRoutine();` should be changed proportionally to the number of times the LEDs should blink faster. For example, to make the LEDs blink twice as fast, the Delay should be halved, i.e. `MAP_UtilsDelay(4000000);`, and to make the LEDs blink ten times as fast, the Delay should be multiplied by 1/10, i.e. `MAP_UtilsDelay(800000);`.

The following methodology was used in the code for instructing the CC3200 to display two different sequences of LED lights depending on which button on the board was pressed:

1. The inputs on SW3 and SW2 were read using the `GPIOPinRead` functions. The `uiPort` and `ucPins`^[2] of the functions were provided in the `pin_mux_config` files generated through the Pin Mux Tool. For SW3, they were `GPIOA1_BASE` and `0x20` and for SW2, they were `GPIOA2_BASE` and `0x40`. Since bits 31:8 of the return values are ignored, and SW3 is in bit 4 and SW2 is in bit 5, the return values of the functions were right-bit shifted by 5 and 6 respectively, and LOGIC ANDed with 1, to see if the switch was actually pressed.
2. The LED routines are in a while loop which is always set to true, i.e., an infinite loop. The condition to escape the loop is the pressing of the other button. So, to escape the while loop for SW3's LED Routine, SW2 should be pressed, and vice versa.
3. If SW3 is pressed, the LEDs should flash in a sequence from 000 to 111, but in the middle of the sequence, SW2 can still be pressed, and the LEDs should switch to flashing on and off simultaneously. This was achieved with *polling*. A for loop was run from 0 to 7, and for every value of *i* in the loop, the corresponding LEDs were turned on (least significant is red and most significant is green). Before the next iteration of the loop, all the LEDs are turned off.
4. For SW2's LED sequence, all the LEDs were turned on and off using `MCU_ALL_LED_IND` which is a variable that affects all of the LEDs instead of just one.
5. A flag variable was used to avoid multiple reports of the same button being pressed. The flag variable worked as follows: 0 if no button is pressed, 1 after SW3 is pressed, 2 after SW2 is pressed. If SW3 is pressed, the flag changes to 1, and it would not enter the if condition that prints that SW3 is pressed. If SW2 is pressed, it exits the while loop that is polling, enters into the if condition for SW2 and then flag is set to 2, and it cannot reenter the if condition to print out "SW2 pressed" more than once.

6. On the first instance of pressing SW3 or SW2, along with changing the flag variable, the P18 value is changed to reflect high on pressing SW2 and low on pressing SW3. Since the value of P18 is on bit 4, left shifting 1 by 4 places and assigning the bit shifted value to GPIOA3_BASE with a value of 0x10 would make P18 high, and doing the same but left shifting 0 by 4 places would make P18 low.

Discussion:

One of the most difficult problems we faced in our lab was to switch from SW3's LED Routine to SW2's LED Routine and vice versa. Our earlier implementation went through the entire sequence from 000 to 111 when SW3 was pressed and then checked if SW2 was pressed after the LEDs blinked for 111. This meant that if SW2 was pressed in the middle of the sequence, it would not register. We changed our approach to the for loop to ensure that polling is done at the end of each number in the sequence. This made sure that if the button was pressed after any part of the sequence, it could be switched to SW2. Since the SW2 sequence had only one component (all three LEDs) turning on and off, this issue was not faced.

Another issue we faced was with the imports. Since we built our lab off the blinky project, the necessary header files for UART were not present in the directory. We had to resolve it by finding the source file in the workspace directory for the UART and copying it into the Lab 1 directory.

Conclusion:

In this lab, we were introduced to the usage of Code Composer Studio and complementary programs like TI Sysconfig (TI Pin Mux Tool) and Uniflash which help run a program on the CC3200. We learned how to adjust frequencies on the LED outputs by creating and adjusting delays and using serial connections to display messages on the PuTTY terminal depending on the inputs on the board using GPIO. We gained hands-on experience with pin-muxing and deciding which pins would serve as inputs and outputs. Using APIs like the GPIO functions and UART, we edited the example blinky program to display different sequences of LEDs depending on which button was pressed. We also used Uniflash to ensure our program ran independently of Code Composer Studio.

Sources:

[1] "Getting started - libtungsten." https://libtungsten.io/tutorials/pin_muxing (accessed Mar. 31, 2022).

[2] “CC3200 Peripheral Driver Library User’s Guide:

GPIO_General_Purpose_InputOutput_api.”

https://software-dl.ti.com/ecs/cc31xx/APIs/public/cc32xx_peripherals/latest/html/group___g_p_i_o___general___purpose___input_output___api.html (accessed Mar. 31, 2022).