ECE/CSE 474

Rania Hussein

Ishan Dane & Anael Aguayo-Chong

11/09/2022

LAB 3 Report

**Section 1: Procedure**

**Task1:**

For task 1a we were tasked with controlling the LEDs using the 10 kOhm potentiometer. The right and left pins were used for the voltage limits 0V (ground) and 3.3V. The middle pin was to change the resistance. We started off by configuring the ports in our header file for the LEDs, TIMER0A, and ADC0. We then set up ADC0 Sample Sequence 3(SS3) from the steps provided in the task 1a code and initialized TIMER0A to trigger ADC0 at 1Hz. After configuring all the registers required to initialize and set up our ADC0 correctly (according to the provided code and comments), we moved to setting up our ADC0 handler method in main. Given the ADC\_value global variable. We would update this value in the handler, on every ADC0 interrupt, with the new ADC value provided by the ADCSSFIFO3 register and clear the interrupt flag status for both TIMER0A and ADC0. Our ADC value would change according to our potentiometer providing the resistance in our circuit from 0 to 10 kOhm. To use our ADC value to manipulate the LEDs, we used the equation given to us from the datasheet, seen in Fig.1, to calculate our resistance which would be updated each time the ADC value is changed by the handler. Using this resistance from the equation and table 1 we were able to implement logic to manipulate the LEDs according to the resistance value. So, as we increased the resistance on the potentiometer, if it was less than 2.5 k ohms LED1 would turn on, and so on as seen in Fig 2.

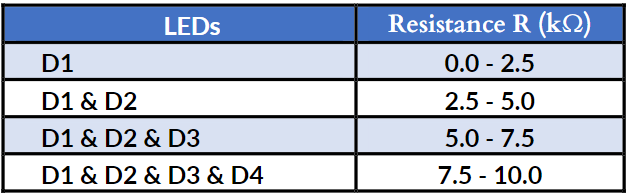




Figure 1: Resistance of ADC0 equation

Figure 2: Table 1 Potentiometer Resistance Threshold

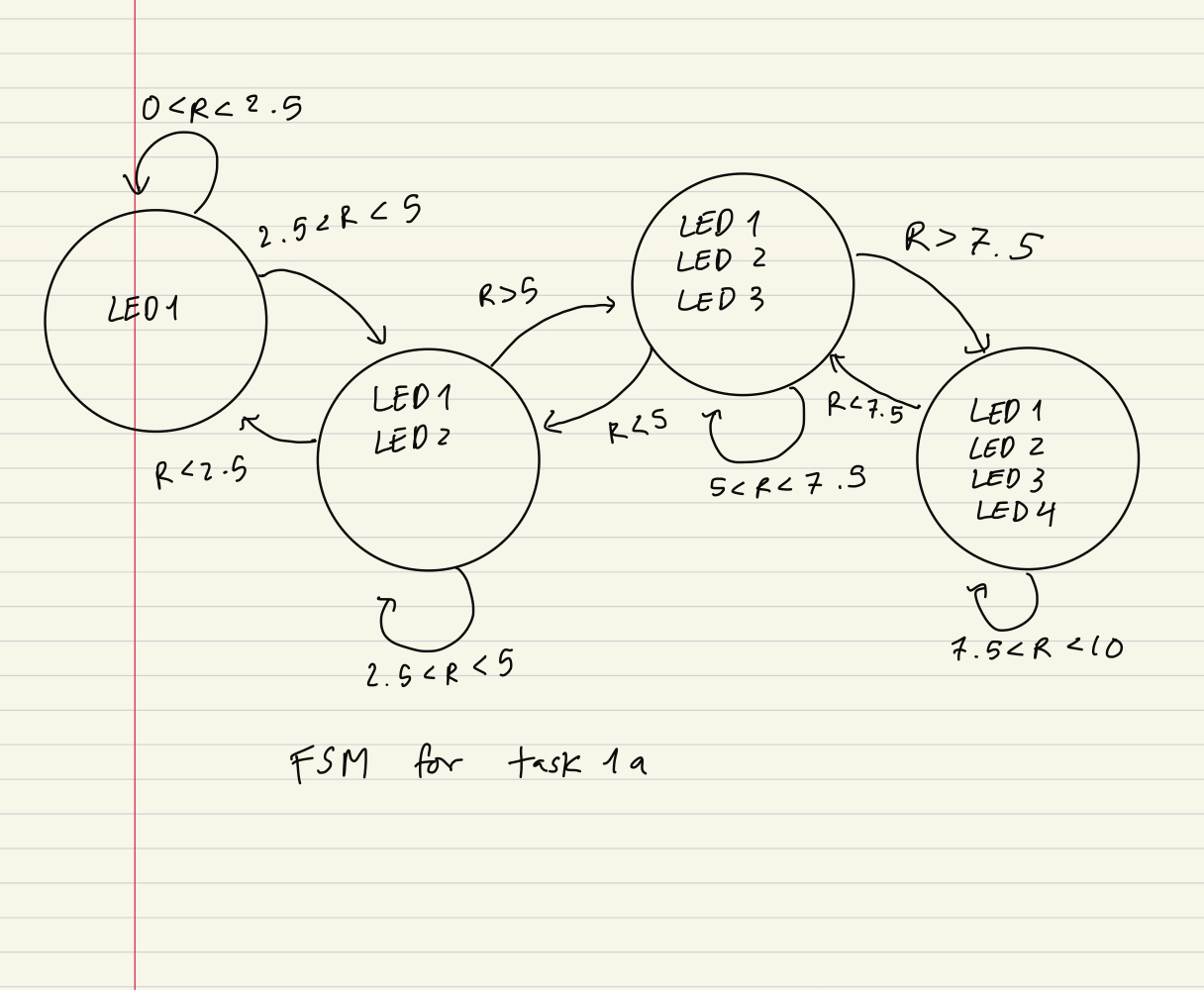


Figure 3:

FSM

For task 1b we were tasked with displaying the microcontroller's internal temperature on the output window. Our program would read the temperature data from an internal temperature sensor located in the board. We would then use 2 on-board switches to manipulate the system temperature by changing the frequency of the system clock to 12MHz or 120MHz based on which switch was pressed. The larger frequency (120 MHz) was used to heat up our system temperature and the smaller frequency (12MHz) was used to cool it down. We needed to show a difference of around +-/2 degrees celsius between both frequency temperatures. Utilizing the code for the ADC0 for 1a we configured the ADCSSCTL and enabled its pins to include the system temperature. Then to convert the ADC value, provided in our handler method, to degrees celsius we used the equation 15.3.6 from the datasheet, Fig.3. We were able to find the temperature by setting the values in our equation to: VREFP = 3.3V (upper limit) and VREFN = 0V (lower limit) and setting ADCcode = the ADC0 value. The temperature value was then stored in our temp float variable. Once we were able to print the temperature at 60MHz onto the output terminal, our next step was to include the switches to control the system’s clock frequency and change our temperature according to the specification. Our switch one was used to decrease the system clock to 12MHz to cool down the system. Our switch 2 was used to increase the system clock to 120MHz to heat up the system. Upon clicking on either switch, the system would change clock frequencies and the new temperature would be shown on the output terminal. We were then able to achieve a +/- 2 degrees celsius difference between the 2 frequencies. The temperature equation we adapted into our code is shown below in Fig 4.



Figure 4: Temperature in Celsius equation

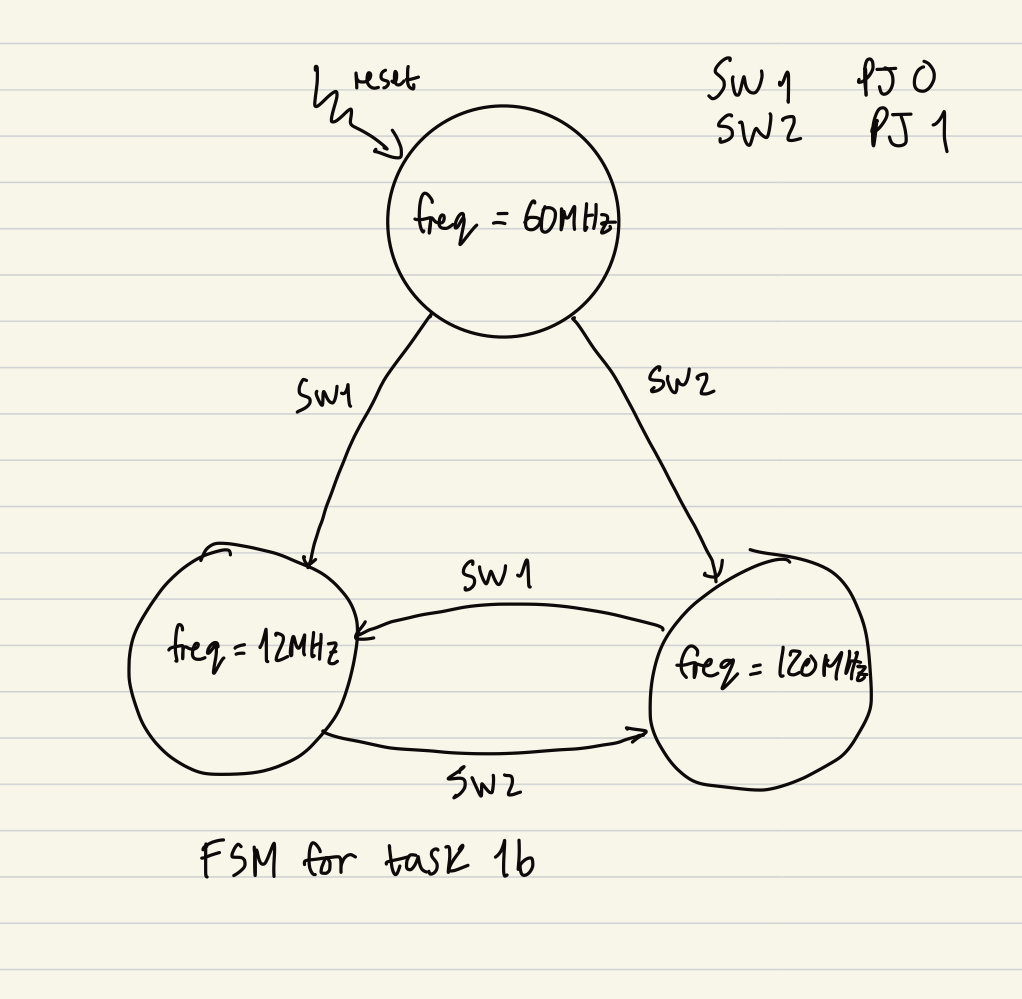
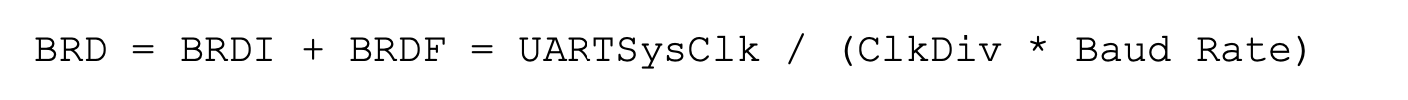


Figure 5: FSM for Task 1b

**Task2:**

For task 2a we were tasked with printing the temperature readings from task 1b into a PUTTY terminal that communicated with our microcontroller through UART0. We first went through the PUTTY specifications to appropriately set up and configure our port connections to communicate with the microcontroller. After that, we configured UART0 and all of its required registers to initialize and set up the UART0 communication line using port A as its virtual pins through the RCGCUART and RCGCPGPIO registers. In our UART configuration, we had to calculate its baud rate divisor and include it in the registers UARTIBRD which holds the integer part of the value, and UARTFBRD which holds the fractional part of the value. To configure this, we used the equations provided in the datasheet shown below in Fig. 6:   


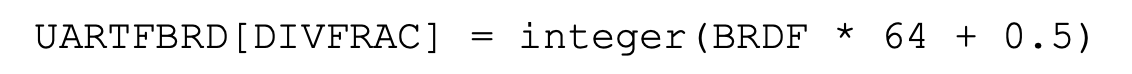


Figure 6: Baud Rate Divisor Equations

From our calculations, we got a value of 390 for UARTIBRD and 40 for UARTFBRD. After configuring these registers, we set up the remaining registers UARTCTL, UARTLCRH, UARTDR and UARTCC to enable transmit (PA0) and receive (PA1) pins for data that is 8 bit, no parity and 1 stop bit. We then configured the GPIO registers for port A’s pins accordingly. After initializing and setting up our UART0, we moved to the main and created our transmit method. Using the data sheet, we knew that we wanted to only transmit data from the computer to the microcontroller when the microcontroller was ready to receive data. Therefore we used a while loop that would check the UARTFR flag’s busy bit and wait until it is unasserted before continuing to transmit data to it. Our transmit method would take a character input and transmit it to the microcontroller. We then created a stringConvert method that would take in a string input, instead of a single character, and transmit it through our transmitter method by using a pointer and incrementing it with each character transmission. Once this was set up and working, we were now able to print out the first part of our PUTTY output: “Temperature in c:”. Our next step was then to convert our temp float value (calculated in task 1b) into a string to be transmitted through our stringConvert method. We did this by using the function ‘sprintf’ to convert our temp float value and store it as a string. This was then inputted into the stringConvert method and transmitted successfully, showing the microcontroller internal temperature in our PUTTY terminal.

For Task 2b, we were tasked with sending information over to our microcontroller through a Bluetooth module connected to our TM4C board. We would then communicate with our microcontroller through a PUTTY terminal. In our task, we had to program the microcontroller to read information from the computer PUTTY terminal through the Bluetooth module and then transmit that same information back to the computer PUTTY terminal. Our first step was to set up the bluetooth module and connect it to our microcontroller. We decided to use UART4 Port K: PK0 and PK1 for our bluetooth module. To set up our bluetooth module, we connected the UART GPIO TX line pin (PK0) to the Bluetooth RXD pin and the UART RX line pin (PK1) to the Bluetooth TXD pin. We then connected the Bluetooth VCC pin to the 5V and GND pin to GND on the microcontroller. After we had set up our bluetooth, we had to set our PUTTY port configurations to enable communication with the Bluetooth module. After that was all set up, we then moved to our code to configure the Port K pins involved and set up UART4. Since we chose to use UART4 due to the pin similarity between PA0/PA1 and PK0/PK1, we were able to use most of our code from task 2a with little adjustment to adapt it to UART4 and the port K pins. The UART4 communication line was now all set and configured. Our next step was then implementing the actual logic to transmit a computer PUTTY input through the bluetooth module and then receive the same input back to the computer as an output on the PUTTY terminal. To do this, we implemented logic using the UARTFR register receiver FIFO full bit to check when the UART receiver FIFO was full as this would mean that the character inputted into PUTTY had been successfully received by the microcontroller and is currently in the FIFO. If this UARTFR flag is raised, we would then transmit the character from the microcontroller back to the computer to be outputted onto the PUTTY terminal. This would then allow us to see our character keyboard inputs to the microcontroller output on the PUTTY terminal as we press them.

**Section 2: Results**

**Task 1:**

For task 1a we set up the 10kOhm potentiometer to control the LEDs. We used the middle pin to connect to GPIO port E which had ADC capability. After configuring our ADC0 with its corresponding registers, we set up the equation to turn the ADC0 value, which was a voltage value, into its resistance utilizing the equation given to us. We then set up the logic that turned the LEDs on and off depending on the resistance value, from Table 1. After setting it all up we ran into problems, and nothing was happening when the potentiometer was turned. We were only getting one LED on and it wasn't turning off. This led us to check the ADC0 registers to make sure we configured and enabled everything correctly. We did not see any mistake with the setup. So our next thought was to check if we had set up the cables to the potentiometer correctly, or if the potentiometer was even working. Looking back at the spec on how to set up the potentiometer we noticed our potentiometer was backward meaning the pins were actually in different locations. Once the potentiometer was flipped everything looked good on the hardware, and we went back into our code. We were finally able to make the LEDs change, but not in the correct order. The problem seemed to be in our logic, and it was because we hadn’t turned on the correct LEDs and turned off the wrong ones in our logic. We found that this was the result of a problem in our if statements, we had forgotten to turn off the remaining LEDs as we transitioned between our LED states. Once we made the change to the logic the potentiometer was controlling the LEDs and turning them on in the correct order.

For Task 1b we were tasked to display the internal temperature of the microcontroller. We first started by using the code from 1a and enabled the ADCSSCTL to read the internal temperature. We used the equation from the datasheet to convert our ADC value (voltage) into temperature in degrees Celsius. We moved on to implement the switches that would change the clock speed to 12MHz and 120MHz. This change in clock speed will make the system cool down or heat up respectively. However, when we were trying to print the temperature we kept printing an incorrect value. We printed the ADC value and used a calculator to prove that the equation was correct and found that we were getting about 27 degrees for 60MHz. We determined that the problem was not in our ADC value, it was in our equation. We started troubleshooting our temperature equation code, by adding and removing brackets. As we started adding more brackets we found that it was an issue with our bracket placement. Once we corrected the brackets the equation started printing the temperature correctly.

**Task 2:**

For task 2a we were tasked with printing the temperature readings from task 1b into a PUTTY terminal that communicated with our microcontroller through UART0. Using our temperature equation from the previous task, we were able to print the temperature readings into the putty terminal using UART serial communication. We were having issues with sending the characters to PUTTY. While we could send characters individually, we were having trouble figuring out how to print our temperature string from the previous task. When we tried to just send the string as a whole character, it would cause PUTTY to freak out and print mumbo jumbo. Instead, we had to create a method that would send data character by character. We had to convert our temperature string into a character array to get the correct characters in putty. To also get the right characters, we had to make sure our baud rate was correct so that the clocks were synchronized. Using the equation from the datasheet, we calculated the divisor registers UARTIBRD and UARTFBRD to get the correct baud rate. We originally thought the baud rate we used in class was the correct baud rate for this lab, but when we kept getting random characters, we realized that our clocks were not synchronized and our baud rate was incorrect. While doing some debugging, we realized that in the lecture, the frequency we were using in class was 16 MHz, while the system clock we were using was 60 MHz. We then redid the equation to get the correct baud rate, which was 390 for UARTIBRD and 40 for UARTFBRG.

For Task 2b, we were tasked with sending information over to our microcontroller through a Bluetooth module connected to our TM4C board. We wanted to type input in our PUTTY which would then send to the microcontroller and it would send it back and print in PUTTY. We were having issues with the Bluetooth it would connect and stop blinking but would not communicate with the microcontroller. We noticed the Bluetooth pins were in the reverse positions after fixing this silly mistake we also noticed that we were using the busy bit (bit3) in register UARTFR. Instead, we used bit 6 UART Receive FIFO Full, so that when it detected the UART was full it would transmit the character to our PUTTY.