Sidharth Daga, Nick Khormaei

2/14/2023

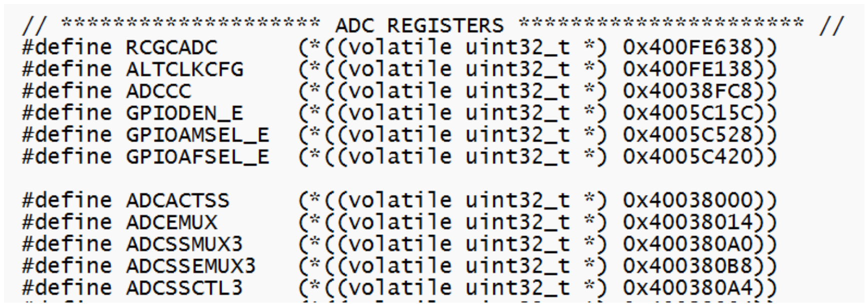
EE 474

Lab 3 Report

Procedure

Task 1:

For task 1a, we first gathered the materials and figured out how to set up the ADC registers on the TIVA board. To approach the setup part of things, we followed the directions layed out in the lab spec and configured the correct registers.

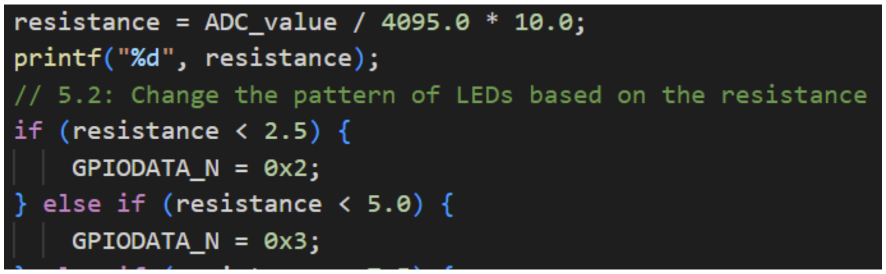


```
// ******************** ADC REGISTERS ********************** //
#define RCGCADC      (*((volatile uint32_t *) 0x400FE638))
#define ALTCLKCFG    (*((volatile uint32_t *) 0x400FE138))
#define ADCCC        (*((volatile uint32_t *) 0x40038FC8))
#define GPIODEN_E    (*((volatile uint32_t *) 0x4005C15C))
#define GPIOAMSEL_E  (*((volatile uint32_t *) 0x4005C528))
#define GPIOAFSEL_E  (*((volatile uint32_t *) 0x4005C420))

#define ADCACTSS     (*((volatile uint32_t *) 0x40038000))
#define ADCEMUX      (*((volatile uint32_t *) 0x40038014))
#define ADCSSMUX3    (*((volatile uint32_t *) 0x400380A0))
#define ADCSSEMUX3   (*((volatile uint32_t *) 0x400380B8))
#define ADCSSCTL3    (*((volatile uint32_t *) 0x400380A4))
```

Figure 1. Registers configured for ADC

In our main program we then implemented an ADC handler function which cleared the ADC0 interrupt flag and saved the ADC value to global variable ADC_value. From there, we converted the ADC value to a readable resistance which turned on specific on board leds.



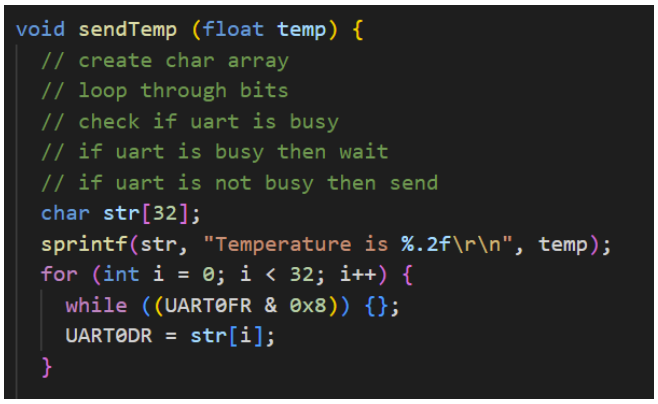
```
resistance = ADC_value / 4095.0 * 10.0;
printf("%d", resistance);
// 5.2: Change the pattern of LEDs based on the resistance
if (resistance < 2.5) {
    GPIODATA_N = 0x2;
} else if (resistance < 5.0) {
    GPIODATA_N = 0x3;
```

Figure 2. Certain on board leds turning on depending on resistance

For task 1b, we had to switch gears a bit since we were looking at temperature. We accomplished this by turning a bit on that was previously disabled for temperature readings. Then we had a continuous check in the while (1) loop that checked if the buttons had been pressed which then switched the clock speeds.

Task 2:

For task 2a, we first approached it by looking at the datasheet and figuring out how to set up the registers for UART. We then followed the directions and saw how to configure putty with the on board UART connection.



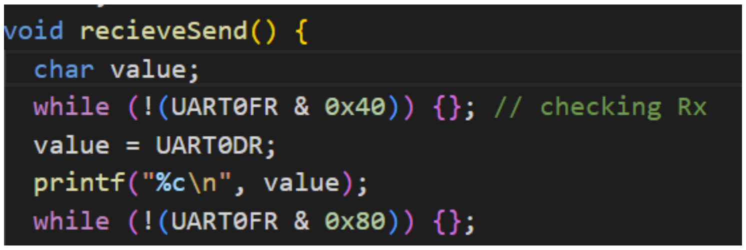
```
void sendTemp (float temp) {
  // create char array
  // loop through bits
  // check if uart is busy
  // if uart is busy then wait
  // if uart is not busy then send
  char str[32];
  sprintf(str, "Temperature is %.2f\r\n", temp);
  for (int i = 0; i < 32; i++) {
    while ((UART0FR & 0x8)) {};
    UART0DR = str[i];
  }
```

Figure 3. sendTemp function

Figure 3. Took temperature from ADC conversion and converted it to string

Then we implemented this function as seen above to produce a char array for the putty terminal.

For task 2b, we approached this by first switching up the header file to reflect UART2 registers rather than UART0. Then we changed the definitions in the initialization file. From there, we had to implement some logic that would take a char in putty then send to the microcontroller via bluetooth then back to putty.



```
void recieveSend() {
  char value;
  while (!(UART0FR & 0x40)) {}; // checking Rx
  value = UART0DR;
  printf("%c\n", value);
  while (!(UART0FR & 0x80)) {};
```

Figure 4. receiveSend() function

Results

Task 1:

For task 1a, we were able to have the leds change in an increasing order depending on the resistance of the potentiometer.

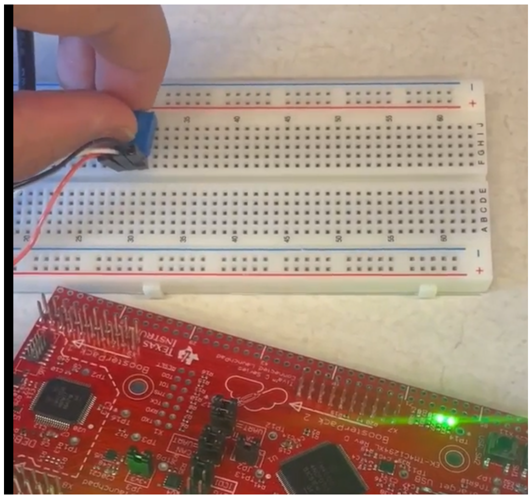
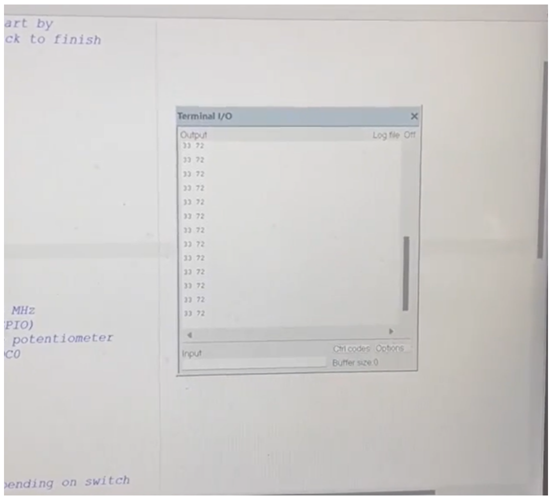


Figure 5. Final results of task 1a

For task 1b, we were able to have the temperature read on the terminal I/O of the IAR workbench. Also, depending on the button presses we were able to have the temperature increase as clock speed was increased and vice versa for decreasing clock speed.



```
art by
ck to finish
```

```
33 72
33 72
33 72
33 72
33 72
33 72
33 72
33 72
33 72
33 72
33 72
33 72
33 72
```

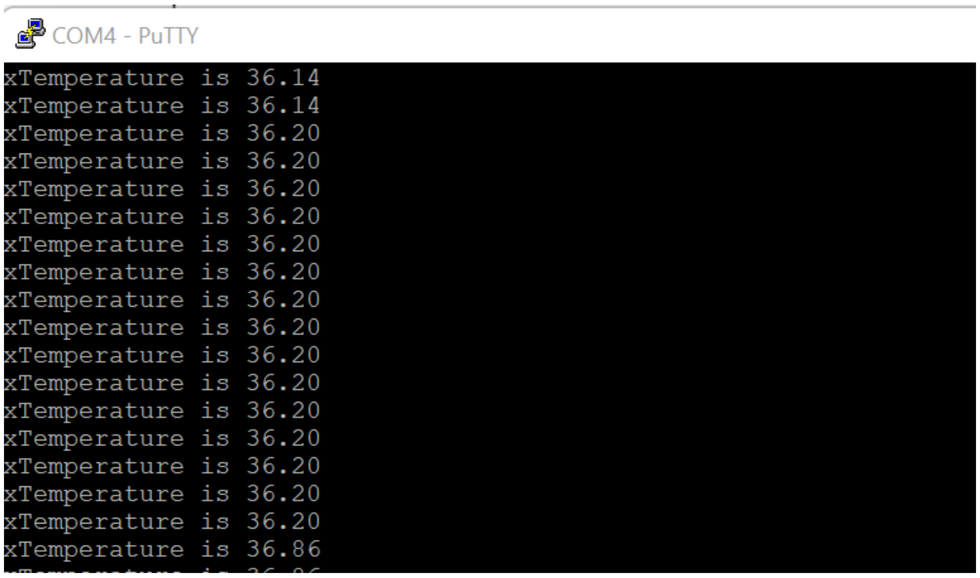
```
 MHz
PIO)
 potentiometer
CO

pending on switch
```

Figure 6. Temperature values printing to Terminal I/O

Task 2:

For task 2a, we were able to have the temperature readings print out on the putty terminal using UART. This relates as earlier we had to set up all the necessary functions and registers in order to accomplish this.


COM4 - PuTTY

```
xTemperature is 36.14
xTemperature is 36.14
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.20
xTemperature is 36.86
```

Figure 7. Results from task 2a, printed temp values into putty terminal

For task 2b, we were able to receive a single character via UART and immediately have it transmit back the character that it received through putty. It looked like this below.

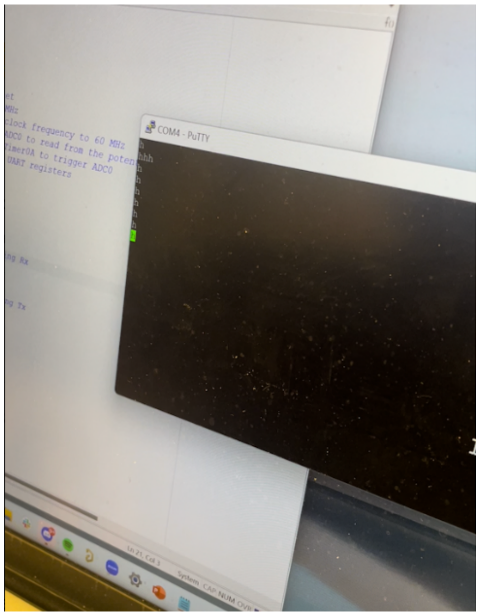


Figure 8. Results of 2b