Assignment 3 Documentation

Marcus Pilgaard Korre

Contents

1 Requirements **3**

2 Hardware **4**

3 Timing **5**

4 File/Program Structure **6**

5 Results and Conclusion **8**

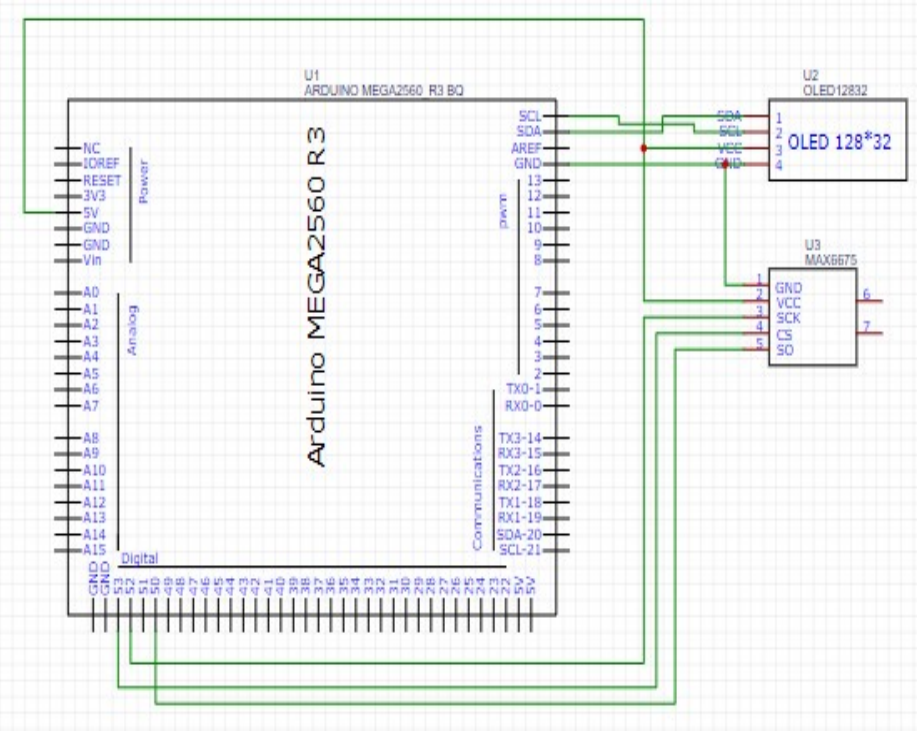
6 Code **10**
6.1 main.c: 10
6.2 SPILib.c: 11
6.3 SPILib.h: 12

1 Requirements

The assignment ask for a program that allows the Arduino to read a temperature from a MAX6675 temperature sensor which communicates over SPI.
The program must adhere to the following requirements:

- The program must read an input from a temperature sensor
- The program must display the temperature to an OLED display
- The program must utilize SPI for communication with the temperature sensor

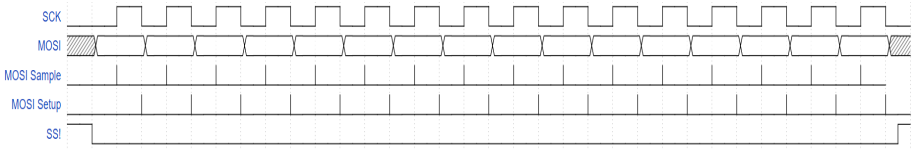
2 Hardware



The MAX6675 board contains no input logic, therefore it does not have a MOSI pin which means it doesn't matter what the Arduino transmits, as long as it transmits something, the MAX6675 will return the temperature.
For the Arduino's pins, we the data sheet for the Arduino to see which outputs are used for SPI.
For the OLED we simply connect the clock and data pins to the I2C clock and data pins on the Arduino.
To program the SPI we use the data direction register and the port/pin register for setting the direction of the pins and its value, we also use the control register to determine the Arduino's role and when it transmits, finally we can write a transmission value in to the data register, and read what the slave returns from the data register.

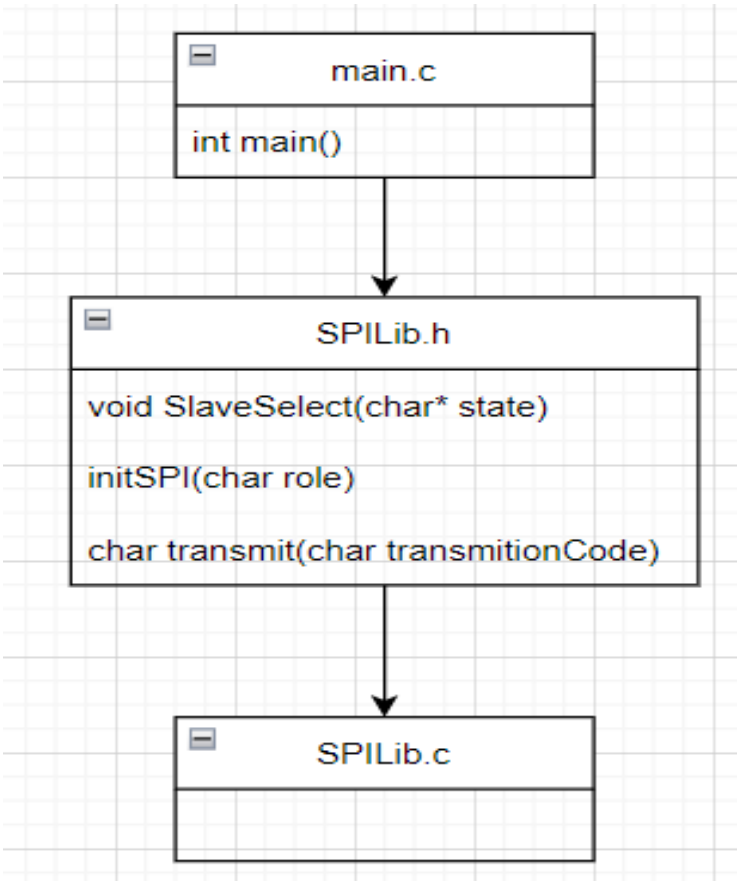
3 Timing

When timing the program, it is important that we have read the data sheet for the MAX6675, as it tells us that the board will re-sample the temperature when the chip-select goes high, which means the entire 16-bit register will be reset. Therefore we must keep chip-select low until we have transmitted and received 2 bytes.
The Arduino has to sample on a leading and rising edge, since sampling on a trailing edge would disrupt the data we read, since we would not read the first bit.
Putting this all together, assuming both bytes are transmitted in immediate succession, we get the following diagram:

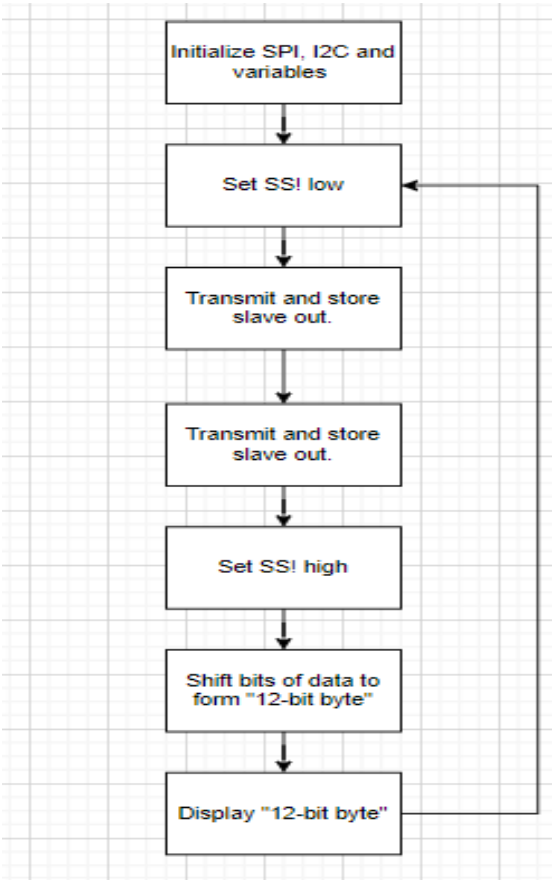


4 File/Program Structure

All functions used for SPI communication has been defined in the SPILib library, these functions are then implemented in main, where they are used to initialize and transmit.
The SlaveSelect either sets SS! high or low, initSPI loads the correct values into the correct register to declare the unit as either a master or a slave, transmit, transmits a byte.

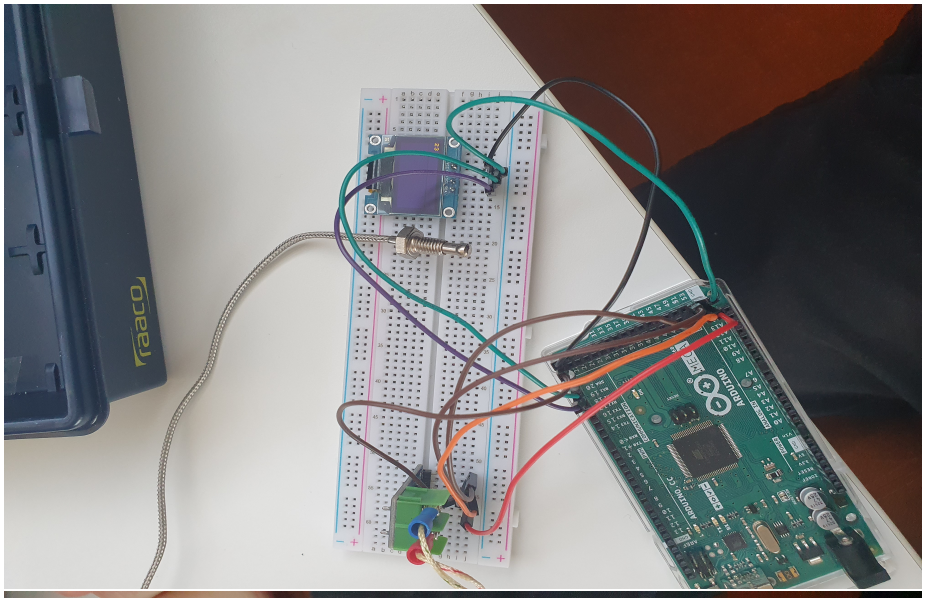


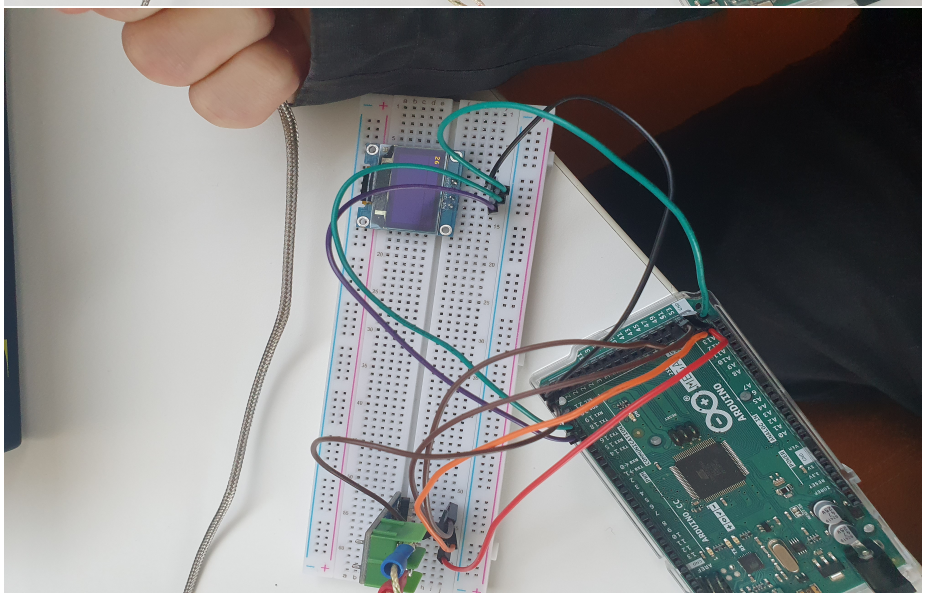
The structure of the program is very simple, first it initialized everything that has to be initialized, then it set SS! low so it can transmit, then it transmits 2 bytes and sets SS! high again, then it shifts the input so we can display it on the OLED display.



5 Results and Conclusion

Upon testing the temperature sensor, we see that it reads room temperature to be 23 degrees celcius, which seems reasonable, and after i hold it in my hand for a little while, the temperature slowly climbs to 26-30 degrees depending on my hands temperature, as the pictures depict:





So in conclusion the temperature sensor can sense a change in temperature and appears to be accurate, the code runs just fine and so i must conclude that the assignment has been completed and that everything works as intended.

6 Code

6.1 main.c:

```
/*
 * GccApplication1.c
 *
 * Created: 2/24/2022 2:50:31 PM
 * Author : marcu
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "I2C.h"  //include library for i2c driver
#include "ssd1306.h" //include display driver
#include "./SPILib.h"

#define F_CPU 16000000UL
#include <util/delay.h>


int main(void)
{
   _i2c_address = 0X78;
   initSPI(MASTER);
   I2C_Init();
   InitializeDisplay();
   print_fonts();
   clear_display();
   char skrr[20];
   int temperature = 0;
   int totalByte;
   int* slaveOutput = (int*)calloc(2, sizeof(int));
   int* tempBytes = (int*)calloc(3, sizeof(int));
   int midBitsMask = 0b00000111;
   int highestBitsMask = 0b01111000;
   int lowestBitsMask = 0b11111000;
   int offSet = 256;
   float resoLution = 0.25;
    while(1){
      slaveSelect(LOW);
      for(int i = 0; i < 2; i++){
         slaveOutput[i] = transmit('s');
      }
      slaveSelect(HIGH);
      tempBytes[0] = (slaveOutput[0] & midBitsMask) << 5;
      tempBytes[1] = (slaveOutput[0] & highestBitsMask) >> 3;
      tempBytes[2] = (slaveOutput[1] & lowestBitsMask) >> 3;
      tempBytes[0] |= tempBytes[2];
      tempBytes[1] *= offSet;
      totalByte = tempBytes[1] + tempBytes[0];
      temperature = (int)((float)totalByte*resoLution);

```

```
53        sprintf(skrr, "%d", temperature);
54        clear_display();
55        sendStrXY(skrr, 0, 0);
56        _delay_ms(1000);
57     }
58 }
```

6.2 SPILib.c:

```
1  /*
2   * SPILib.c
3   *
4   * Created: 2/24/2022 2:56:10 PM
5   * Author : Marcus Korre
6   */
7
8  #include <avr/io.h>
9  #include <stdlib.h>
10 #include <avr/interrupt.h>
11 #include "./SPILib.h"
12 #include <string.h>
13
14 /* Replace with your library code */
15
16 void slaveSelect(char* state){
17    if(strcmp(state, "HIGH") == 0){
18       PORTB |= (1<<PB0);
19    }
20    else if(strcmp(state, "LOW") == 0){
21       PORTB = !(1<<PB0);
22    }
23
24 }
25
26 int initSPI(char role){
27    sei();
28    if(role == MASTER){
29       DDRB = (1<<PB2)|(1<<PB1)|(1<<PB0); //Sets ss-not, MOSI and SCK to be
         ↪  outputs
30       slaveSelect(HIGH);
31       SPCR = (1<<SPE)|(1<<MSTR); //Initiates control register
32       return 0;
33    }
34    else if(role == SLAVE){
35       DDRB = (1<<PB3); //Sets MISO as output
36       PORTB = (1<<PB0); //Sets clock to pull high
37       SPCR = (1<<SPE); //Initiates control register
38       return 0;
39    }
40    return 1;
41 }
42
43 char transmit(char transmitionCode){
44    if((DDRB & 0b00000111) == 0b00000111){ //If this is a master
45       if((PORTB & 0b00000001) == 0b00000001){ //If !ss is set to high
```

```
46          return 0;
47       }
48       SPDR = transmitionCode;
49       while((SPSR & 0b10000000) != 0b10000000){} //Waits for transmition to
         ↪  finish
50       return SPDR;
51    }
52    return 1;
53 }
54
55 ISR(SPI_STC_vect){
56
57 }
```

6.3 SPILib.h:

```
1  #ifndef SPILib_H
2  #define SPILib_H
3
4  #define MASTER 0x00
5  #define SLAVE 0xff
6  #define HIGH "HIGH"
7  #define LOW "LOW"
8
9  void slaveSelect(char* state);
10 int initSPI(char role);
11 char transmit(char transmitionCode);
12
13 #endif // SPILib_H
```