Assignment 4

Carson Bolinger, Roberto Solis, Carson Wagner

March 14, 2024

1 Introduction

In Assignment 4, we were tasked with utilizing the LPC1769's I2C subsystem to establish connections with the TC74A temperature sensor and the MCP23017 input-output expander. Additionally, we integrated a button linked to a 7-segment LED display, enabling toggling between Celsius and Fahrenheit readings with each button press. As a group of three, we had additional responsibilities, including incorporating a second dual-digit 7-segment LED display to showcase a three-digit temperature display.

2 Software Code

```
1
2  /*
3   * Copyright 2022 NXP
4   * NXP confidential.
5   * This software is owned or controlled by NXP and may only be used strictly
6   * in accordance with the applicable license terms.  By expressly accepting
7   * such terms or by downloading, installing, activating and/or otherwise using
8   * the software, you are agreeing that you have read, and that you agree to
9   * comply with and are bound by, such license terms.  If you do not agree to
10  * be bound by the applicable license terms, then you may not retain, install,
11  * activate or otherwise use the software.
12  *
13  *
14  *
15  */
16
17 #ifdef __USE_CMSIS
18 #include "LPC17xx.h"
19 #endif
20
21 #include <cr_section_macros.h>
22 #include <stdbool.h>
23
24
25 #define I2C0CONSET (*(volatile unsigned int *) 0x4001C000)
26 #define I2C0CONCLR (*(volatile unsigned int *) 0x4001C018)
27 #define I2C0SCLH (*(volatile unsigned int *) 0x4001C010)
28 #define I2C0SCLL (*(volatile unsigned int *) 0x4001C014)
29
30 #define PCONP (*(volatile unsigned int *) 0x400FC0C4)
31 #define I2C0DAT (*(volatile unsigned int *) 0x4001C008)
32
33
34 #define PINSEL1 (*(volatile unsigned int *)  0x4002C004)
35 #define PCLKSEL0 (*(volatile unsigned int *)0x400FC1A8)
36
37
38 int data;
39
40 int opcodeIO1 = 0b01001110; // IO1 Write
41 int opcodeIO2 = 0b01000000; // IO2 Write
42 int opcodeTempWrite = 0b10011100; // Temp Write OpCode
43 int opcodeTempRead = 0b10011101; // Temp Read OpCode
```

```
44
45
46 bool tempUnit = 1; // 1 Celsius, 0 Fahrenheit
47 int temp = 0;
48 int sw = 1;
49 int swPush;
50 int unit;
51
52
53 int tempArray[10];
54
55
56 void initializeI2C () {
57
58   PCONP |= (1 << 7);
59   PCLKSEL0 |= (1 << 14); // change to get desired frequency, pclk divider
60   PCLKSEL0 |= (0 << 15);
61
62   // Initializing SDA0 and SCL0 (Pin 25 and Pin 26)
63   PINSEL1 &= ~(1 << 23);
64   PINSEL1 |= (1 << 22);
65
66   PINSEL1 &= ~(1 << 25);
67   PINSEL1 |= (1 << 24);
68
69   // I2C0SCLH and I2C0SCLL are Equal both are 5
70   I2C0SCLH = 5; // duty cycle divider, must be 100khz by here
71   I2C0SCLL = 5;
72
73
74   // Clear and set I2C
75   I2C0CONCLR = (1 << 6); // Clears I2C
76   I2C0CONSET = (1 << 6); // Enables I2C
77 }
78
79
80 void initializeIO() {
81
82   // Initialize IO1 A Register
83   startI2C();
84   dataWrite(opcodeIO1); // IO1 Address
85   dataWrite(0x00); // IODIRA Register
86   dataWrite(0x80); // Write A bits to be all outputs except Bit 7 (switch input) in
       GPA (1000 0000 in binary)
87   stopI2C();
88
89   // Initialize IO1 B Register
90   startI2C();
91   dataWrite(opcodeIO1); // IO1 Address
92   dataWrite(0x01); // IODIRB Register
93   dataWrite(0x00); // Write B bits to be all outputs
94   stopI2C();
95
96   // Initialize IO2 A Register
97   startI2C();
98   dataWrite(opcodeIO2); // IO2 Address
99   dataWrite(0x00); // IODIRA Register
100  dataWrite(0x00); // Set all A bits to outputs
101  stopI2C();
102
103  // Initialize IO2 B Register
104  startI2C();
105  dataWrite(opcodeIO2); // IO2 Address
106  dataWrite(0x00); // IODIRB Register
107  dataWrite(0x00); // Set all B bits to outputs
108  stopI2C();
109
110 h
111
112 }
113
```

```
114
115 void initializeTempSensor() {
116
117   startI2C();
118   dataWrite(opcodeTempWrite); // Send Write OpCode for Sensor
119   dataWrite(0x00); // Read Temp Address
120   stopI2C();
121 }
122
123
124
125 void dataWrite(int data) {
126
127
128   // Write Data to I2C then clear SI bit
129   I2C0DAT = data;
130   I2C0CONCLR = (1 << 3);
131
132   while(!((I2C0CONSET >> 3) & 1)) {
133
134   }
135
136 }
137
138 int dataRead() {
139
140
141   // Clear SI Bit and wait for it to go back to 1
142   I2C0CONCLR = (1 << 3);
143
144   while(!((I2C0CONSET >> 3) & 1)) {
145
146   }
147
148   int data = I2C0DAT;
149   return data;
150
151 }
152
153
154
155 int readSwitch() {
156
157   int opcodeIO1Read = 0b01001111; // IO1 Read
158
159
160   startI2C();
161   dataWrite(opcodeIO1); // IO1 Write Address
162   dataWrite(0x12); // GPIOA Register
163   startI2C();
164   I2C0CONCLR = (1 << 2); // Clear Ack bit
165   dataWrite(opcodeIO1Read); // IO1 Read Address
166
167
168   // Clear SI bit and wait for it to go back to 1
169   I2C0CONCLR = (1 << 3);
170
171     while(!((I2C0CONSET >> 3) & 1)) {
172
173     }
174
175   // Write Read data
176   data = I2C0DAT;
177
178   data = ((data >> 7) & 1); // Store Bit 7 to data
179
180   stopI2C();
181
182   return data;
183
184
```

```
185
186 }
187
188
189
190
191
192 int readTemp() {
193
194
195   startI2C();
196   dataWrite(opcodeTempWrite); // Write Register
197   dataWrite(0x00); // Read Temperature Register
198   startI2C();
199   dataWrite(opcodeTempRead); // Read Register to read bit
200   I2C0CONCLR = (1 << 2); // Clear AA bit
201
202
203   temp = dataRead();
204
205   stopI2C();
206
207   return temp;
208
209
210 }
211
212 int segConvert(int digit) {
213
214   switch (digit) {
215     case 0:
216       return 0b00111111; // Display 0
217       case 1:
218           return 0b00000110; // Display 1
219       case 2:
220           return 0b01011011; // Display 2
221       case 3:
222           return 0b01001111; // Display 3
223       case 4:
224           return 0b01100110; // Display 4
225       case 5:
226           return 0b01101101; // Display 5
227       case 6:
228           return 0b01111101; // Display 6
229         case 7:
230           return 0b00000111; // Display 7
231       case 8:
232             return 0b01111111; // Display 8
233         case 9:
234           return 0b01101111; // Display 9
235       case 'C':
236           return 0b00111001; // Display C
237       case 'F':
238           return 0b01110001; // Display F
239        case '-':
240            return 0b01000000; // Display -
241        default:
242          return 0;
243   }
244
245 }
246
247
248 void writeTemp(int temp) {
249
250
251
252   // unitTemp = 1 (Celsius), unitTemp = 0;
253   if(tempUnit == 0) {
254
255     unit = segConvert('F'); // Selects F
```

```
  }
  else if(tempUnit == 1) {

    unit = segConvert('C');
  }

  // Check if 1, 2, or 3 digit value
  if(temp < 10) {

    int tempDigit = segConvert(temp);
    int digitZero = segConvert(0);
    int unit;


    writeDigitIO1(tempDigit, unit);
    writeDigitIO2(digitZero, digitZero);

  }
  else if(temp > 99) {

    // Calculate individual numbers
    int hundPlace = temp / 100;
    int tenthsPlace = (temp - 100) / 10;
    int onesPlace = (temp - 100) % 10;

    // Convert values to binary
    int digitZero = segConvert(0);
    int hundPlaceDigit = segConvert(hundPlace);
    int tenthsPlaceDigit = segConvert(tenthsPlace);
    int onesPlaceDigit = segConvert(onesPlace);


    writeDigitIO1(onesPlaceDigit, unit);
    writeDigitIO2(hundPlaceDigit, tenthsPlace);
  }
  else {

    int tenthsPlace = temp / 10;
    int onesPlace = temp % 10;

    int tenthsPlaceDigit = segConvert(tenthsPlace);
    int onesPlaceDigit = segConvert(onesPlace);
    int digitZero = segConvert(0);

    writeDigitIO1(onesPlaceDigit, unit);
    writeDigitIO2(digitZero, tenthsPlaceDigit);
  }



}


void writeDigitIO1(int right, int left) {



  startI2C();
  dataWrite(opcodeIO1); // Write IO1 Opcode
  dataWrite(0x12); // Write GPIOA register address
  dataWrite(left); // Give data for GPIOA
  dataWrite(right); // Writing GPIOB data automatically
  stopI2C();


}

void writeDigitIO2(int right, int left) {

  startI2C();
  dataWrite(opcodeIO2); // Write IO1 Opcode
```

```
  dataWrite(0x12); // Write GPIOA register address
  dataWrite(left); // Give data for GPIOA
  dataWrite(right); // Writing GPIOB data automatically
  stopI2C();


}

void startI2C () {


  // Clear and set I2C
  I2C0CONCLR = (1 << 6); // Clears I2C
  I2C0CONSET = (1 << 6); // Enables I2C


  I2C0CONSET = (1 << 3); // Set SI bit
  I2C0CONSET = (1 << 5); // Set Start Bit
  I2C0CONCLR = (1 << 3); // Clear SI bit


  // Wait for SI bit to go to 1
  while(!((I2C0CONSET >> 3) & 1)) {

  }

  // Clear Start Bit
  I2C0CONCLR = (1 << 5);
}



void stopI2C() {

  // Set STOP bit
  I2C0CONSET = (1 << 4);
  I2C0CONCLR = (1 << 3); // Clear SI Bit


  // Wait for STOP bit to be set
  while(((I2C0CONSET >> 4) & 1)) {

  }
}


// Function to create a delay in milliseconds
void wait_ms(int ms) {
    volatile int i;

    float m = 0.002715;
    float b = 0.1;

    ms = (ms-b)/m;

    for (i = 0; i < ms; i++) {
    //do nothing
    }

}


int main(void) {

  // Initialize I2C, IO1/IO2, and Temperature Sensor
    initializeI2C();
    initializeIO();
    initializeTempSensor();


    while(1) {
```

```
398
399
400         // Read and store temperature data
401         temp = (int) readTemp();
402
403         // Read switch and store data
404         swPush = readSwitch();
405
406
407         // Use for toggling
408         int swPressed = 1;
409
410         // If Switch is Pressed change to F
411         if(swPush == 0) {
412           tempUnit = !tempUnit; // Change to Fahrenheit (0) or Celsius (1)
413           swPressed = 0; // True, switch pressed
414
415           wait_ms(200);
416         }
417
418         //temp = (int) readTemp();
419
420
421         // If Unit is F, convert C to F (0 is F, 1 is C)
422         if(tempUnit == 0) {
423
424           temp = ((temp*9)/5) + 32;
425         }
426
427         writeTemp(temp);
428
429
430         // For toggling the switch to allow it to stay at Celsius or Fahrenheit
431         while((swPressed == 0) && (readSwitch() == 0))
432         {
433           // Wait until switch is pressed again
434         }
435
436     }
437
438     return 0 ;
439
440 }
```

2.1 Software Calculations

In our software calculations, we knew that the system clock is operating at four megahertz. From there, we synchronized the clock inside the microcontroller, where $PLCK$ is equal to CLK. This connection was then represented by the equation:

$$\frac{PLCK}{(SCLH + SCLL)}$$

where we let $SCLH + SCLL$ be equal to a single variable, X. We used the 400K (Fastmode) and set it equal to:

$$\frac{PLCK}{X}$$

where we then solved for X, where we got X = 10 which we then split in half due to making it one single variable eariler which we then obtained $SCLH = 5$ and $SCLL = 5$.

3 Hardware

While the hardware in this lab was not as conceptually intense as in previous labs it still required some calculations. The main one is what the value of the pullup resistors on the SDA and SCL lines should be. To calculate this we used;

$$R << \frac{1}{3fC}$$

and

$$R > \frac{V_{DD}}{I_{OL}}$$

Where F is the frequency of the I^2C interface, C is the capacitance of the components on the interface, V_{DD} is the voltage supplied by the microcontroller, and I_{OL} is the current syncing capacity of the devices. Using these equations we were able to determine that the should be between 1kΩ and 10kΩ. To fit these constraints we chose to use 2.2k$Omega$ for the pullup resistors.

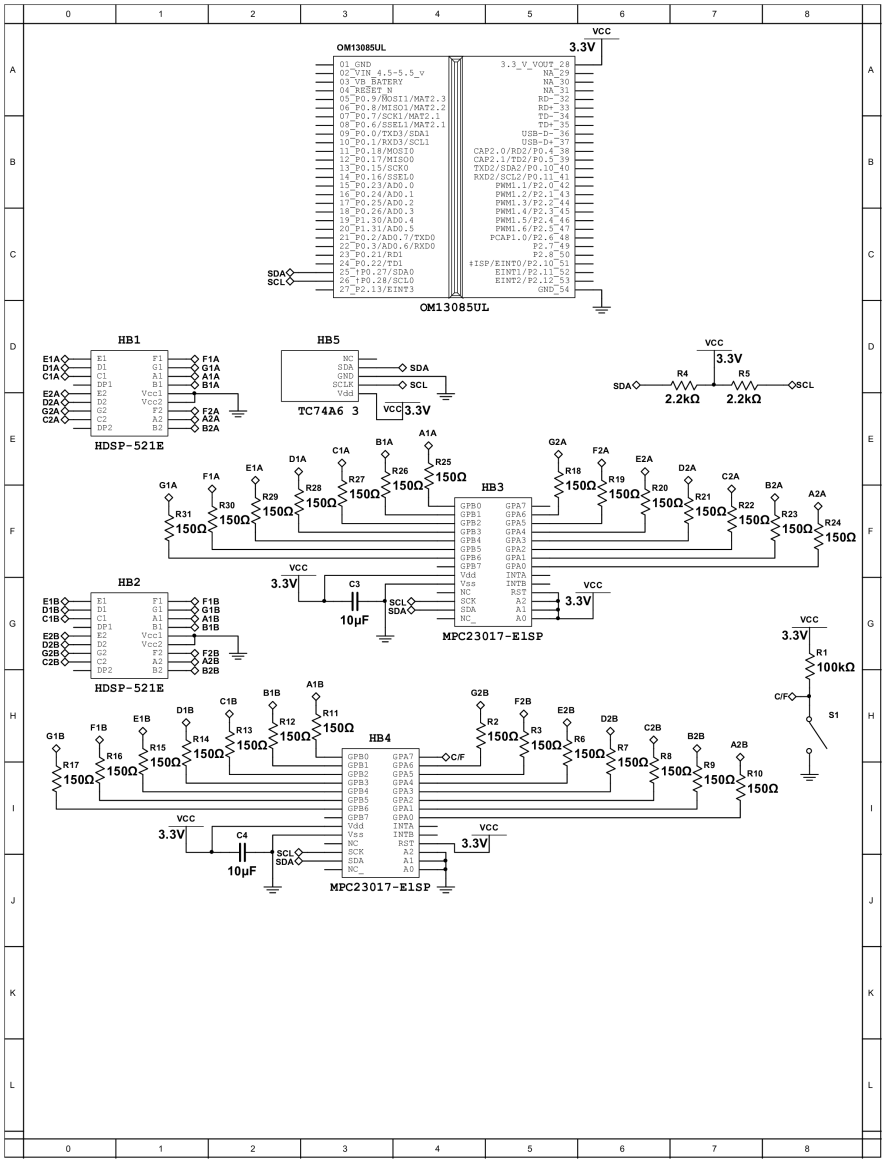
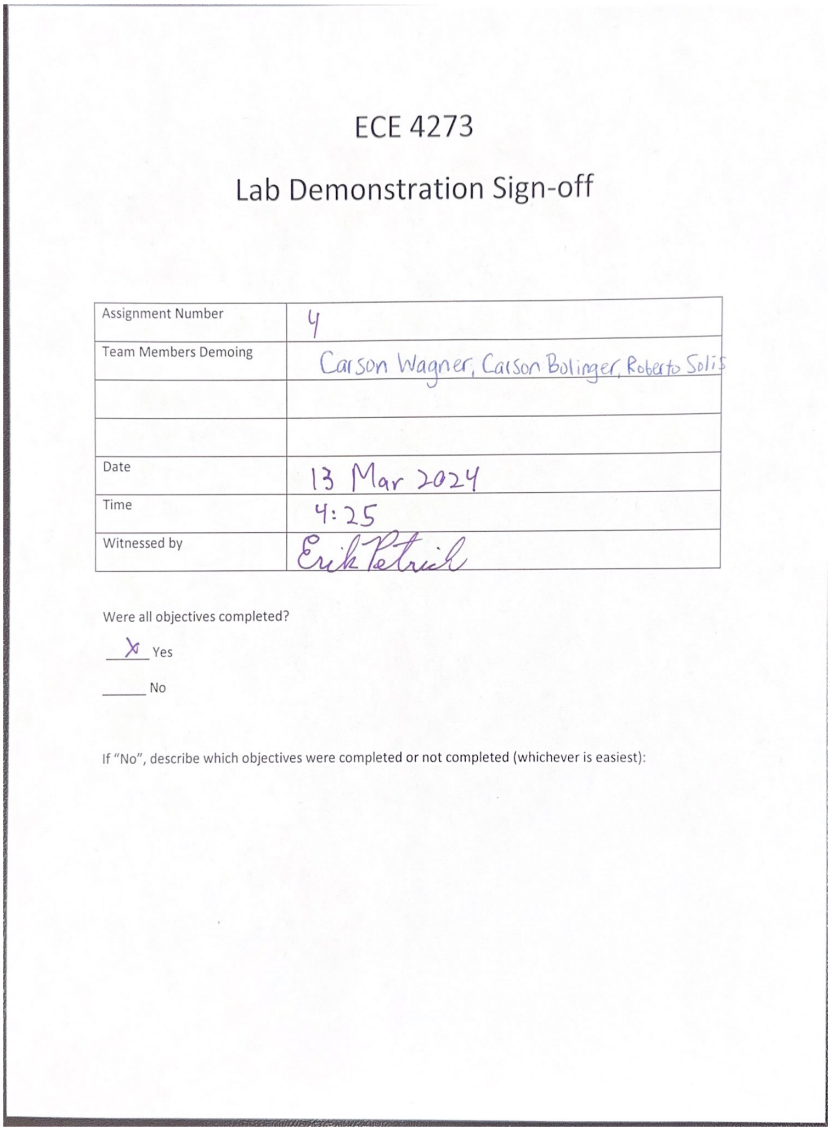


Figure 1: Hardware Schematic

4 Lab Demonstration Sheet



ECE 4273

Lab Demonstration Sign-off

Assignment Number	4
Team Members Demoing	Carson Wagner, Carson Bolinger, Roberto Solis
Date	13 Mar 2024
Time	4:25
Witnessed by	Erik Petrich

Were all objectives completed?

X Yes

_____ No

If "No", describe which objectives were completed or not completed (whichever is easiest):

Figure 2: Lab Demonstration Sign-off.

5 Contributions

Carson Bolinger (Electrical Engineering)

Designed and built the hardware and did the calculations for the pullup resistors. I also helped troubleshoot the code.

Roberto Solis (Electrical Engineering)

Helped troubleshoot code, and helped write the document.

Carson Wagner (Computer Engineering)

Worked on code and calculated the clock for the I2C and troubleshooted code.