Assignment 4: Lab Report

Digital Design Lab

Sarah Brown

I. Introduction

I2C and other communication protocols can be used to interface with various sensors and other chips. This assignment uses the LPC1769's I2C subsystem to create a digital thermometer. This implements a temperature sensor, an I/O expander, a 7-segement LED display, and a switch to toggle between Celsius and Fahrenheit. The display is updated at least every one second and shows the temperature in the currently selected degrees.

II. Hardware Solution

The hardware solution for this assignment has several calculations to ensure that everything operates within the various specifications. One of the first calculations is for the pull-up resistor for SDA and SCL. This is computed to be within two bounds:

$$\frac{V_{DD}}{I_{OL}} < R \ll \frac{t_{cycle}}{3C} = \frac{1}{3fC}$$

Here the I2C clock frequency must be set to an appropriate value (less than or equal to 100 kHz). C is the effective capacitance between the 3 devices tied in parallel to SCL, assuming each device is 10pF, this results in a C of 30pF for the 3 devices. V_{DD} is 3.3V and the smallest I_{OL} for the various devices is 3mA (the TC74 and the MCP23017 both at 3mA and the LPC1769 at 4mA). This results in the following bounds:

$$\frac{3.3V}{3mA} < R \ll \frac{1}{3(100kHz)(30pF)}$$
$$1.1\ kOhm < R \ll 111\ kOhm$$

Selecting a R value in the middles of 10 kOhm, gives the following values.

$$t_r \approx 3RC$$
$$t_r \approx 3(10000)(30 * 10^{-12})$$
$$t_r \approx 900\ nanoseconds$$
$$t_r \ll \frac{1}{f}$$
$$900\ ns \ll \frac{1}{100{,}000}$$
$$900\ ns \ll 10\ \mu s$$

And:

$$I = \frac{3.3V}{10\ kOhm} = 0.33mA$$

These values all hold within specifications. Next are the frequency calculations. The frequency is controlled by I2CxSCLH and I2CxSCLL.

$$Freq = \frac{PCLK}{I2CxSCLH + I2CxSCLL}$$

PCLK is set to CCLK value/4 by default. This leaves it at PCLK = 4MHz/4=1MHz. Then I2CxSCLH and I2CxSCLL are set equal to each other to ensure a duty cycle of 50%. Setting these values both to 5 gives:

$$Freq = \frac{1MHz}{5+5} = 100\ kHz$$

Finally, the LEDs in the 7-segement display are limited by the Maximum current out of the Vss pin of the MCP23017 as the 7-segement display used was common anode. This limit is 150mA and since there are 14 LEDs on at the same time, this gives:

$$\frac{150\ mA}{14} = 10.7\ mA/pin$$
$$\frac{3.3\ V}{10.7\ mA} = 308\ ohms$$

This value is rounded for safety and convenience to 470 ohms. The final schematic can be seen in the appendix.

III. Software Solution

The software solution relied on various functions, the core of which were four functions for basic I2C transactions. These were i2c-start(), i2c-stop(), i2c-read(), and i2c-write(). Additional functions were added for simplicity to help calculate the digits to display the temperature, to read the temperature from the sensor, and to read button input.

A transaction is started by calling a start condition to inform other devices that data will soon be transmitted. A stop conditions indicates that it will soon be finished. A write function is used to write and transfer data to and from the bus. The read function transfers data from the data register to other variables.

The write and read functions are composed of start and stop functions. By using various addresses unique to the different components, it is possible to transfer data across the bus. The exact protocols followed the slide notes as well as the datasheet specifications.

IV. Appendix

A. *Schematic*

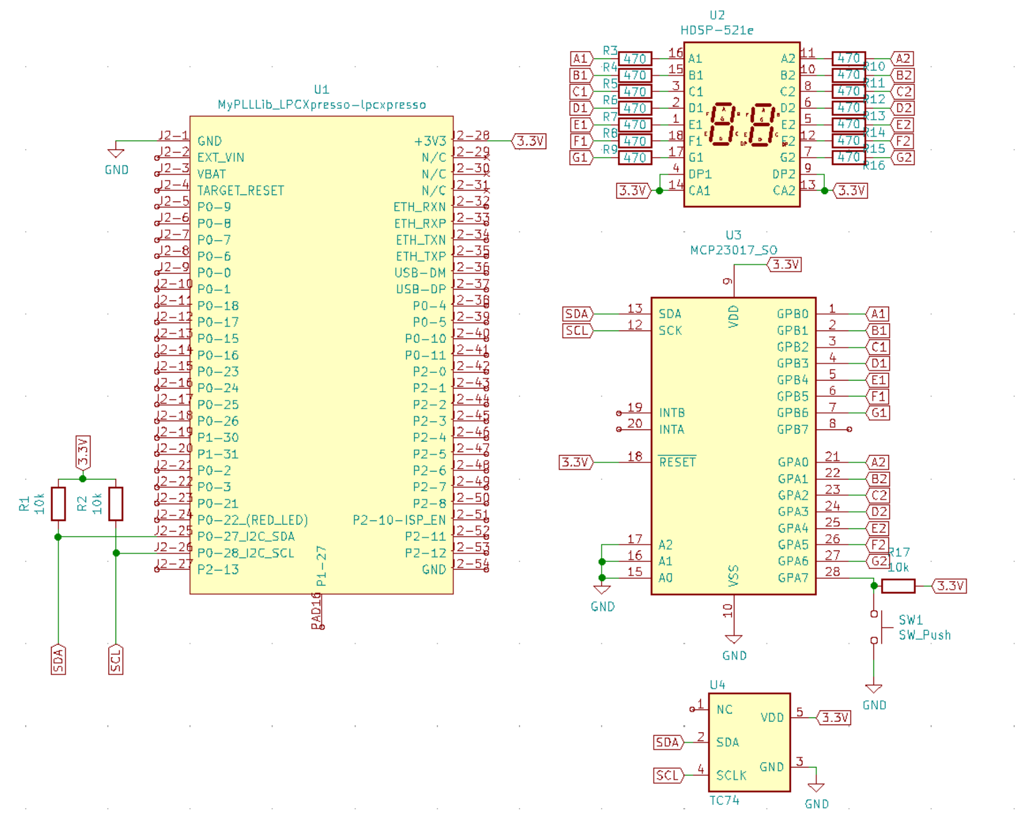

Figure 1: Schematic of hardware design

B. Code

```
// define LPC1769 I2C registers
#define PINSEL1       (*(volatile unsigned int *)0x4002C004) // pin function select
#define PCLKSEL0      (*(volatile unsigned int *)0x400FC1A8) // peripheral clock select
#define I2C_SCLH      (*(volatile unsigned int *)0x4001C010) // sclh duty cycle
#define I2C_SCLL      (*(volatile unsigned int *)0x4001C014) // scll duty cycle
#define I2C_CONSET    (*(volatile unsigned int *)0x4001C000) // control set
#define I2C_CONCLR    (*(volatile unsigned int *)0x4001C018) // control clear
#define I2C_DAT       (*(volatile unsigned int *)0x4001C008) // data
#define I2C_STAT      (*(volatile unsigned int *)0x4001C004) // status

// define logic to be logical
#define TRUE 1
#define FALSE 0

// I2C addresses
#define TC74_ADDR_W 0x90 // temp sensor I2C ADDR write
#define TC74_ADDR_R 0x91 // temp sensor I2C ADDR read
#define MCP_ADDR_W 0x40// MCP I2C ADDR write
#define MCP_ADDR_R 0x41// MCP I2C ADDR read
#define MCP_IODIRA 0x0 // MCP IO Pins DirA
#define MCP_IODIRB 0x1 // MCP IO Pins DirB
#define LEFTSIDE 0x13 // GPIOB
#define RIGHTSIDE 0x12 // GPIOA


// constants
#define CEL 0
#define FAH 1

// global variables
int cur_mode = CEL;

/*
 * Function to waste time
 *
 * @param ticks - x variable in function
 * y = 2.73x + 8.5. units: y - usec, x - ticks
 */
void wait_ticks(int ticks) {
           volatile int count;
           for (count=0; count<ticks; count++) {
                      //do nothing
           }
}

// I2C Start
void i2c_start() {
           I2C_CONSET = (1 << 3); // set SI bit in i2c cont
           I2C_CONSET = (1 << 5); // set STA bit in i2c cont
    I2C_CONCLR = (1 << 3); // clear SI bit

    while (((I2C_CONSET >> 3) & 1) == 0) {} // wait for SI bit to return to 1

    I2C_CONCLR = (1 << 5); // clear STA bit
}

// I2C Write
void i2c_write(int data) {
           I2C_DAT = data;
    I2C_CONCLR = (1 << 3); // clear SI bit
    while (((I2C_CONSET >> 3) & 1) == 0) {} // wait for SI bit to return to 1

}

// I2C Read
int i2c_read(int stop) {
           // wait for data
    I2C_CONCLR = (1 << 3); // clear SI bit
    while (((I2C_CONSET >> 3) & 1) == 0) {} // wait for SI bit to return to 1

           int data = I2C_DAT; // save the data

           if (stop) { // send nack
        I2C_CONCLR = (1 << 2);
           }

           else { //send ack
        I2C_CONSET = (1 << 2);
           }

           return data;
}

// I2C Stop
void i2c_stop() {
           I2C_CONSET = (1 << 4); // set STO bit in i2c cont
    I2C_CONCLR = (1 << 3); // clear SI bit
    while (((I2C_CONSET >> 4) & 1) == 0) {} // wait for STO bit to return to 1
}

// Write Single Digit Num
void write_num(int num, int side) {
        // i2c and mcp stuff
        i2c_start();
        i2c_write(MCP_ADDR_W);
        i2c_write(side);

        switch (num) {
                default:
                        i2c_write(0x40); break;
                case 0:
                        i2c_write(0x40); break;
                case 1:
                        i2c_write(0x79); break;
                case 2:
                        i2c_write(0x24); break;
                case 3:
                        i2c_write(0x30); break;
                case 4:
                        i2c_write(0x19); break;
                case 5:
                        i2c_write(0x12); break;
                case 6:
                        i2c_write(0x03); break;
                case 7:
                        i2c_write(0x78); break;
                case 8:
                        i2c_write(0x00); break;
                case 9:
                        i2c_write(0x18); break;
        }
        i2c_stop();
}

// Write Two Digit Num
void write_two_digit(int temp) {
        int tens_place = temp/10;
        write_num(tens_place, LEFTSIDE);

        int ones_place = temp%10;
        write_num(ones_place, RIGHTSIDE);
}

// Read Temp
int read_temp() {
        int temp = 0;
        // read value
        wait_ticks(100);
        i2c_start(); // start i2c
        i2c_write(TC74_ADDR_W); // select temp
        i2c_write(0); // temp reg 0
        wait_ticks(100);
        i2c_start(); // start read
        i2c_write(TC74_ADDR_R); // temp reg 0

        temp = i2c_read(TRUE);
        wait_ticks(100);
        i2c_stop();

        // if negative, undo twos complement
        if ((temp & 0x80) != 0) {
                temp = -(~temp+1);
        }

        if (cur_mode == FAH) {
                temp = ((9.0/5.0)*temp)+32.0;
        }

        return temp;
}

int button_press() {
        int data;
        i2c_start();
        i2c_write(MCP_ADDR_W);
        i2c_write(RIGHTSIDE);
        i2c_start();
        i2c_write(MCP_ADDR_R);
        data = i2c_read(TRUE);
        i2c_stop();

        return data>>7;
}

void setup() {
        PINSEL1 |= (1 << 22); PINSEL1 |= (1 << 24); // sets scl and sda

        I2C_SCLL = 5; I2C_SCLH = 5; // configs i2c freq

        I2C_CONCLR = 0x38; I2C_CONSET = 0x40;  // inits i2c

    // setup mcp
    // set dir a to output and one input
        i2c_start();
        i2c_write(MCP_ADDR_W);
        i2c_write(MCP_IODIRA);
```

```
            i2c_write(0x80); // set a7 to input, rest to output
            i2c_stop();

    // set all dir b to output
            i2c_start();
            i2c_write(MCP_ADDR_W);
            i2c_write(MCP_IODIRB);
            i2c_write(0x00); // set all b to output
            i2c_stop();
}

int main(void) {
            setup();
            int temper = 0;
            int button=0; int prev_button=0;
    while(1) {
            temper = read_temp();
            button = button_press();

            write_two_digit(temper);
            // write temp
            // change temp mode. toggle between when pressed
            if (button ==1 && prev_button == 0) {
                        cur_mode = !cur_mode;
            }
            prev_button=button;

    }
}
```