Lab Report 4 - Taylor Rainwater, Triston Luzanta

**Objectives**

The objective of this assignment is to familiarize ourselves with the LPC 1769’s I2C subsytem. We will be creating a temperature sensor that displays the temperature in both Celsius and Farenheit. The sensor will display Celsius in default. However, the user can switch between Celsius and Fahrenheit measurements by pressing a button. To achieve this, we will be using the LPC 1769’s I2C subsystem interfaced with a PCT2075 temperature sensor and an MCP23017 I/O expander to a switch and a dual-digit 7-segment LED display to display the resulting temperature.

**Design**

The primary consideration of the design of this project are the SDA and SCL pull up resistors, the LED resistors on the display, and the SCLH and SCLL values for the I2C clock speed.

To determine the pull up resistors on SDA and SCL we began by determining the upper and lower limits for the values using eqn.1 And 2.

Where *f* = 100kHz (the selected I2C clock speed), and *C* = 20pF (10pF per device).

Where *VDD* = 3.3V and *IOL* = 0.003mA (from the GPIO datasheet).

We selected 5.2K𝛀 because it is well above the minimum and should be low enough to allow the devices to communicate.

For the LED resistors we selected *R* = 270𝛀, and performed calculations using eqn.3.

Where 2.05V is the bias voltage of the LEDs, and 14 is the number of possible LEDs that can be on at once.

The resistors keep the current below the maximum threshold.

For the SCLH and SCLL values we selected them so that the I2C clock would be 100kHz for the fastest possible clock speed. We calculated SCLH and SCLL using eqn.4 assuming SCLH = SCLL = X.

Where *CCLK* = 4MHz (default clock of the LPC1769).

**Hardware**

Hardware schematic

**Source Code**

/\*

===============================================================================

Name : Lab\_4.c

Author : Taylor Rainwater, Triston Luzanta

Version :

Copyright : $(copyright)

Description : main definition

===============================================================================

\*/

#ifdef \_\_USE\_CMSIS

#include "LPC17xx.h"

#endif

#include <cr\_section\_macros.h>

#include <stdio.h>

//LPC registers related to I2C initialization

#define PCONP (\*(volatile unsigned int\*) 0x400FC0C4)

// Power Control for Peripherals Register. This register contains control bits that enable and disable individual peripheral functions, allowing elimination of power consumption by peripherals that are not needed

#define PCLKSEL0 (\*(volatile unsigned int\*) 0x400FC1A8)

// Peripheral Clock Selection registers 0. A pair of bits in a Peripheral Clock Selection register controls the rate of the clock signal that will be supplied to the corresponding peripheral as specified in Table 40,

#define I20SCLH (\*(volatile unsigned int\*) 0x4001C010)

// SCH Duty Cycle Register High Half Word. Determines the high time of the I2C clock.

#define I2C0SCLL (\*(volatile unsigned int\*) 0x4001C014)

// SCL Duty Cycle Register Low Half Word. Determines the low time of the I2C clock. I2nSCLL and I2nSCLH together determine the clock frequency generated by an I2C master and certain times used in slave mode.

#define PINSEL1 (\*(volatile unsigned int\*) 0x4002C004)

// The PINSEL1 register controls the functions of the upper half of Port 0. The direction control bit in the FIO0DIR register is effective only when the GPIO function is selected for a pin. For other functions the direction is controlled automatically.

#define I2C0CONSET (\*(volatile unsigned int\*) 0x4001C000)

// I2C Control Set Register. When a one is written to a bit of this register, the corresponding bit in the I2C control register is set. Writing a zero has no effect on the corresponding bit in the I2C control register.

#define I2C0CONCLR (\*(volatile unsigned int\*) 0x4001C018)

// I2C Control Clear Register. When a one is written to a bit of this register, the corresponding bit in the I 2C control register is cleared. Writing a zero has no effect on the corresponding bit in the I2C control register.

#define I2C0DAT (\*(volatile unsigned int\*) 0x4001C008)

// I2C Data Register. During master or slave transmit mode, data to be transmitted is written to this register. During master or slave receive mode, data that has been received may be read from this register

#define I2C0STAT (\*(volatile unsigned int\*) 0x4001C004)

// I2C Status Register. During I2C operation, this register provides detailed status codes that allow software to determine the next action needed.

//temp sensor I2C address and internal register addresses

#define TEMPADDR 0x90

// Slave address

#define TEMPREG 0x00

// Internal address of temperature register

//GPIO expander I2C address and internal register addresses

#define GPIOADDR 0x40

// Slave address

// Internal addresses

#define IODIRA 0x00

#define IODIRB 0x01

#define GPPUA 0x0C

#define GPPUB 0x0D

#define GPIOA 0x12

#define GPIOB 0x13

//sends start signal to SDA bus

void start(){

I2C0CONSET = (1<<3); // 3rd bit set to 1

I2C0CONSET = (1<<5); // 5th bet set to 1

I2C0CONCLR = (1<<3); // 3rd bit cleared to 0

while(!((I2C0CONSET>>3) & 1)){

// Wait for 3rd bit to 1

}

I2C0CONCLR = (1<<5); // 5th bit cleared to 0

}

//reads 8 bit data from slave device

int read(int final){

if (final == 0){

I2C0CONCLR = (1<<2); // 2nd bit cleared to 0

}

else if (final == 1){

I2C0CONSET = (1<<2); // 2nd bit set to 1

}

I2C0CONCLR = (1<<3); // 3rd bit clear to 0

while(!((I2C0CONSET>>3) & 1)){

// Wait for 3rd bit to 1

}

int temp = I2C0DAT;

return temp; // Returns that read data

}

//writes 8 bit data to slave device,

//if sending opcode use RW bit to indicate whether the next command is a read or a write

void write(int DATA, int RW){

I2C0DAT = DATA + RW;

I2C0CONCLR = (1<<3); // 3rd bit set to 0

while(!((I2C0CONSET>>3) & 1)){

// Wait for 3rd bit to 1

}

}

//sends stop signal to SDA bus

void stop(){

I2C0CONSET = (1<<4);

I2C0CONCLR = (1<<3);

while(((I2C0CONSET>>4) & 1)){

// Wait for 4th bit to 1

}

}

//converts celcius to fahrenheit

int C2F(int tempC){

return (int)((((float)tempC \* 9.0) / 5.0) + 32.0 + 0.5);

}

//parses a 2 digit temperature value into its individual digits

void parseDigits(int temp, int arr[]){

arr[0] = temp / 10;

arr[1] = temp % 10;

}

//decodes a 1 digit number to the outputs on the GPIO expander (aka. 7 Seg display)

int decodeDigit(int digit){

switch(digit){

case 0:

return 0x80;

case 1:

return 0xF2;

case 2:

return 0x48;

case 3:

return 0x60;

case 4:

return 0x32;

case 5:

return 0x24;

case 6:

return 0x04;

case 7:

return 0xF0;

case 8:

return 0x00;

case 9:

return 0x20;

default:

return 0x0C;

}

}

//wait function with seconds parameter

void wait(float sec)

{

volatile int ticks = (int)((sec - 0.00000843) / 0.00000237);

volatile int count;

for (count=0; count<ticks; count++){

}

}

int CF = 0;

int main(void) {

//initializes I2C pins on LPC

PCONP |= (1<<7);

PCLKSEL0 &= ~(1<<14) & ~(1<<15);

I20SCLH = 5;

I2C0SCLL = 5;

PINSEL1 |= (1<<22) | (1<<24);

I2C0CONCLR = (1<<6);

I2C0CONSET = (1<<6);

//initializes the GPIO expander

start();

write(GPIOADDR,0);

write(IODIRA,0);

write(0x01,0);

stop();

start();

write(GPIOADDR,0);

write(IODIRB,0);

write(0x00,0);

stop();

start();

write(GPIOADDR,0);

write(GPPUA,0);

write(0x01,0);

stop();

start();

write(GPIOADDR,0);

write(GPPUB,0);

write(0x00,0);

stop();

//variables to handle temperature, digits, unit, and button state

int temp;

int digits[2];

int button;

//main loop of program to read and display temperature

while(1){

//get temperature from sensor

start();

write(TEMPADDR,0);

write(TEMPREG,0);

start();

write(TEMPADDR,1);

temp = read(1);

read(0);

stop();

//read switch state from GPIO

start();

write(GPIOADDR,0);

write(GPIOA,0);

start();

write(GPIOADDR,1);

button = read(0);

stop();

//perform calculations with temperature data and switch state

if(!(button &= 1)){

CF = !CF;

}

wait(0.075);

if (CF == 1){

temp = C2F(temp);

}

parseDigits(temp, digits);

digits[0] = decodeDigit(digits[0]);

digits[1] = decodeDigit(digits[1]);

//write data to display

start();

write(GPIOADDR,0);

write(GPIOA,0);

write(digits[0],0);

stop();

start();

write(GPIOADDR,0);

write(GPIOB,0);

write(digits[1],0);

stop();

}

}

