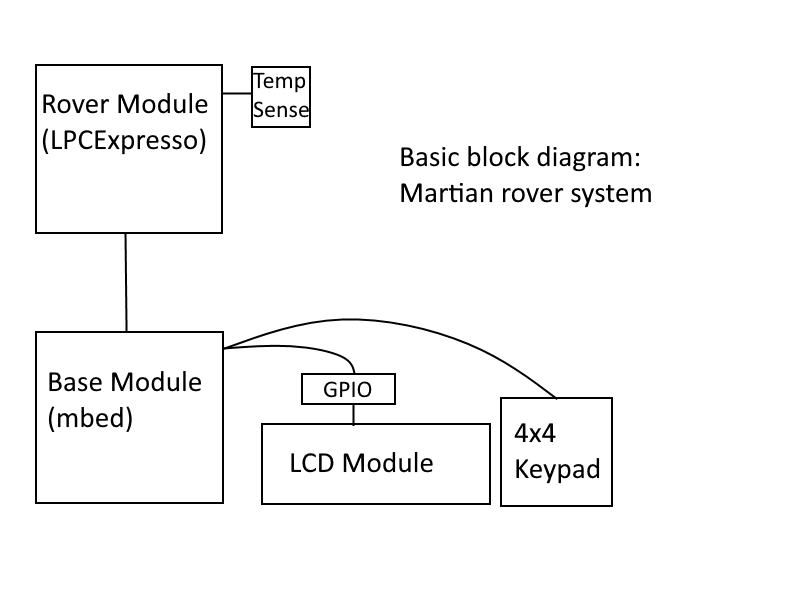
Final Design Report

Taylor Rainwater, Triston Luzanta

**General Description:**

The purpose of this project is to create a digital system in which two microcontrollers communicate to each other via a serial communication link, akin to a planetary rover and a remote control station. In this system the rover module should be able to collect certain data and relay it to the control module, while the control module should be able to relay commands to the rover module. During the design we are also tasked in adding features to both our rover and control station. Each feature is worth a different amount of points. The goal of this project is to implement features that add up to at least 5 points.Below are the proposed features we would like to implement to our design. 

Science Features

**Temperature - 1 point**

System Features

**Scheduling - 2 points**

**Data Logging - 1 point**

Implementation Features

**LCD - 0.5 points**

**Keypad - 0.5 points**

**Directions For Use:**

The Martian Rover system consists of two modules; A base/ground control module, and a rover module. The rover module features a temperature sensor for measuring the temperature of the Martian environment, as well as the ability to take these measurements at specified times as per an internal schedule stored in the rover device. The rover also stores all measurements and times as a data point for later viewing. The rover can schedule and store up to 32 measurements at once. The base module features an LCD display, and a 4x4 keypad for user interface. The two modules communicate through a serial data communication bus.

Once both the base device and rover device have been powered on, the the user will be presented with the following text on the LCD display:

A: Get a point.

B: Create a new schedule.

C: Update schedule.

This is the home screen. From here all the functionality of the system can be accessed.

1. **Getting a data point:**

The “Get a point.” function allows the user to retrieve a data point from the previously stored data points in the rover and display them on the LCD. When the user presses the “A” key on the keypad while on the home screen, the LCD screen will change and display the following text:

Display point: \_

On this screen the user can either enter a number between 1 and 32 in nn format to retrieve the corresponding data point, or the user may press the “#” to return to the home screen. After a point is retrieved, the data will be displayed in the following format:

Point nn

Time: hh:mm

Temperature: tt C

From here the user may press the “#” key to return to the home screen.

1. **Creating a new measurement schedule:**

The “Create a new schedule.” command allows the user to create a completely new schedule and store it in the base module’s memory. When the user presses the “B” key while on the home screen, the LCD will display the following text:

Enter time 01: \_

From here the user can either enter a time from a 24 hour clock format in the form hhmm, or the user can press the “#” key to return to the home screen. Whenever a user enters a time the schedule increments to the next point and asks the user to enter another time. The user can create a schedule with any number of points between 1 and 32. If the user wishes to create a schedule with less than 32 times, the “#” key can be pressed at any time during the create schedule command to return to the home screen, and any previously entered times will be saved in the newly created schedule.

1. **Updating the rover schedule:**

The “Update schedule.” command sends the schedule that is currently saved in the base module’s memory to the rover module and sets that as the rover’s current schedule. When the user presses the “C” key while on the home screen the base module will automatically send the saved schedule to the rover without any further input from the user. While the schedule is updating the LCD will display the following text:

Updating schedule...

However in most instances this screen will display for less than a second and will appear to the user as a flash of the screen. If the screen displays this text for a prolonged period of time, a problem has occurred and a system restart may be necessary. After the update successfully completes the LCD will return to the home screen automatically.

**Design:**

The primary consideration of the design of this project are the SDA and SCL pull up resistors, 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).

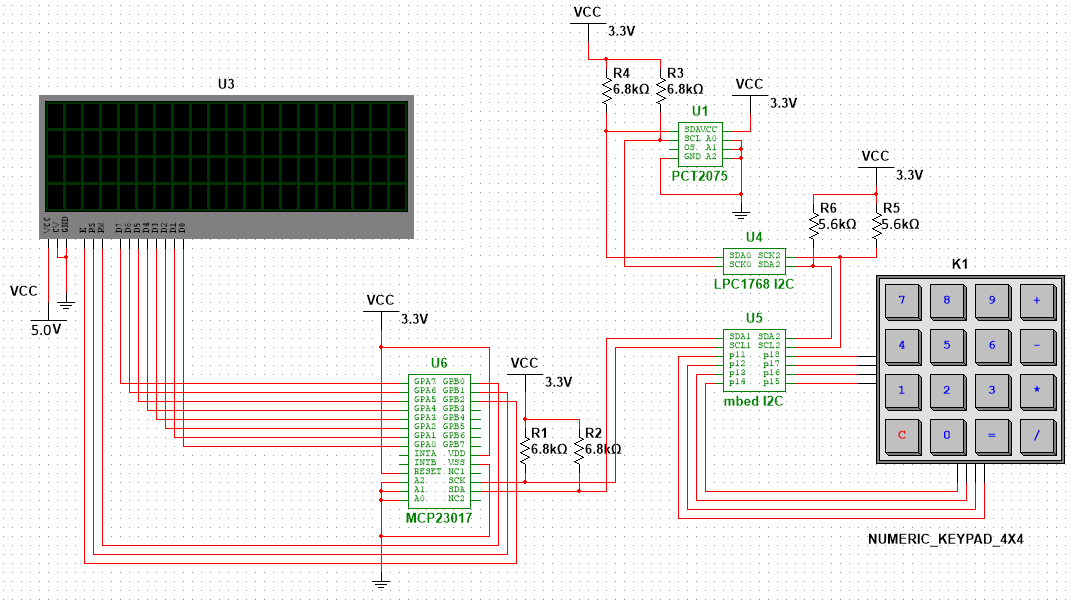
Where *VDD* = 3.3V and *IOL* = 0.003mA (from the GPIO datasheet).

We selected 6.8K𝛀 because it is well above the minimum and should be low enough to allow the devices to communicate. 5.6k𝛀 was used on the communication between the rover and the base since we ran out of 6.8k but it will provide similar performance.

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.3 assuming SCLH = SCLL = X.

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

**Schematic:**

In this project we will be using an Mbed microcontroller acting as a ground control module, interfacing to a 4x4 keypad and an LCD module through an I2C IO expander. We are also using a LPCXpresso microcontroller interfacing to an I2C temperature sensor for the rover module. Both microcontrollers use the LPC1768 chipset, and communicate through I2C.

**Software:**

/\*

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

Name : set\_time.c

Author : Taylor Rainwater, Triston Luzanta

Version :

Copyright : $(copyright)

Description : sets the rover clock to the current time

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

\*/

#ifdef \_\_USE\_CMSIS

#include "LPC17xx.h"

#endif

#include <cr\_section\_macros.h>

#include <stdio.h>

#include <time.h>

#define SEC (\*(volatile unsigned int\*) 0x40024020)

#define MIN (\*(volatile unsigned int\*) 0x40024024)

#define HOUR (\*(volatile unsigned int\*) 0x40024028)

#define CCR (\*(volatile unsigned int\*) 0x40024008)

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

int main(void) {

PCONP |= (1<<9);

CCR = 0;

time\_t seconds;

time(&seconds);

int sec = seconds % 60;

int min = (seconds / 60) % 60;

int hrs = (((seconds / 60) / 60) % 24) - 5;

SEC = sec;

MIN = min;

HOUR = hrs;

}

/\*

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

Name : Rover\_Code.c

Author : Taylor Rainwater, Triston Luzanta

Version :

Copyright : $(copyright)

Description : implements rover module functionality

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

\*/

#ifdef \_\_USE\_CMSIS

#include "LPC17xx.h"

#endif

#include <cr\_section\_macros.h>

#include <stdio.h>

//////////////////////////////////////

///LPC internal register addresses.///

//////////////////////////////////////

//MISC LPC register definitions

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

#define PCLKSEL1 (\*(volatile unsigned int\*) 0x400FC1A8)

#define PINMODE0 (\*(volatile unsigned int\*) 0x4002C040)

#define PINMODE\_OD0 (\*(volatile unsigned int\*) 0x4002C068)

#define PINSEL0 (\*(volatile unsigned int\*) 0x4002C000)

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

//I2C0 LPC register definitions

#define I2C0SCLH (\*(volatile unsigned int\*) 0x4001C010)

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

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

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

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

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

//I2C2 LPC register definitions

#define I2C2SCLH (\*(volatile unsigned int\*) 0x400A0010)

#define I2C2SCLL (\*(volatile unsigned int\*) 0x400A0014)

#define I2C2CONSET (\*(volatile unsigned int\*) 0x400A0000)

#define I2C2CONCLR (\*(volatile unsigned int\*) 0x400A0018)

#define I2C2DAT (\*(volatile unsigned int\*) 0x400A0008)

#define I2C2STAT (\*(volatile unsigned int\*) 0x400A0004)

#define I2C2ADR1 (\*(volatile unsigned int\*) 0x400A0020)

#define I2C2MASK1 (\*(volatile unsigned int\*) 0x400A0034)

//RTC subsystem LPC register definitions

#define ILR (\*(volatile unsigned int\*) 0x40024000)

#define CCR (\*(volatile unsigned int\*) 0x40024008)

#define AMR (\*(volatile unsigned int\*) 0x40024010)

#define ALMIN (\*(volatile unsigned int\*) 0x40024064)

#define ALHOUR (\*(volatile unsigned int\*) 0x40024068)

#define SEC (\*(volatile unsigned int\*) 0x40024020)

#define MIN (\*(volatile unsigned int\*) 0x40024024)

#define HOUR (\*(volatile unsigned int\*) 0x40024028)

////////////////////////////////////////////////////////

///Peripheral device and internal register addresses.///

////////////////////////////////////////////////////////

//Rover address

#define ROVER 0xAA

//Rover command

#define UPSCHED 0xFF

//Temp sensor device address

#define TEMPADDR 0x90

//Temp sensor internal addresses

#define TEMPREG 0x00

//////////////////////////////////////////////////////////

///I2C protocol to communicate with temperature sensor.///

//////////////////////////////////////////////////////////

//sends start signal to slave device

void start(){

I2C0CONSET = (1<<3);

I2C0CONSET = (1<<5);

I2C0CONCLR = (1<<3);

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

}

I2C0CONCLR = (1<<5);

}

//reads 8 bit data from slave device

int read(int final){

if (final == 0){

I2C0CONCLR = (1<<2);

}

else if (final == 1){

I2C0CONSET = (1<<2);

}

I2C0CONCLR = (1<<3);

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

}

int data = I2C0DAT;

return data;

}

//sends 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);

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

}

}

//sends stop signal to slave device

void stop(){

I2C0CONSET = (1<<4);

I2C0CONCLR = (1<<3);

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

}

}

//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++){

}

}

///////////////////////////////

///Primary rover functions.////

///////////////////////////////

//global rover variables

const int invalid = 0xff;

int validPoints = 0;

int currentDataPoint = 0;

int alHour = 0;

int alMin = 0;

int commandBuffer = 0;

int tempMem [128];

int schedule[32][2];

int memMap[32][4];

int t = 0;

//stores the time data in the memory

void scheduleToMem(){

for(int i = 0; i < 32; i++){

tempMem[memMap[i][0]] = schedule[i][0];

tempMem[memMap[i][1]] = schedule[i][1];

tempMem[memMap[i][2]] = 0;

tempMem[memMap[i][3]] = invalid;

}

}

//reads temperature and stores data in corresponding data point

void storeTemp(int dataPoint){

int temp = 0;

start();

write(TEMPADDR,0);

write(TEMPREG,0);

start();

write(TEMPADDR,1);

temp = read(1);

read(0);

stop();

tempMem[memMap[dataPoint][2]] = temp;

}

//sets alarm registers to generate interrupt at specified time.

void setAlarm(int dataPoint){

alHour = schedule[dataPoint][0];

alMin = schedule[dataPoint][1];

}

int main(void){

//initializes I2C pins on LPC

PCONP |= (1<<7);

PCLKSEL1 &= ~(1<<14) & ~(1<<15);

I2C0SCLH = 5;

I2C0SCLL = 5;

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

I2C0CONCLR = (1<<6);

I2C0CONSET = (1<<6);

//initialize I2C2 pins on LPC

PINMODE0 |= (1<<21) | (1<<23);

PINMODE0 &= ~(1<<20) & ~(1<<22);

PINMODE\_OD0 |= (1<<10) | (1<<11);

PCONP |= (1<<26);

PCLKSEL1 &= ~(1<<20) & ~(1<<21);

//I2C2SCLH = 5;

//I2C2SCLL = 5;

PINSEL0 |= (1<<21) | (1<<23);

PINSEL0 &= ~(1<<20) & ~(1<<22);

I2C2ADR1 = ROVER;

I2C2MASK1 = 0xFF;

I2C2CONCLR = (1<<5);

I2C2CONCLR = (1<<4);

I2C2CONCLR = (1<<3);

I2C2CONSET = (1<<2);

I2C2CONCLR = (1<<6);

I2C2CONSET = (1<<6);

//Initialize memory map

int hour = 0;

int min = 1;

int temp = 2;

int terminator = 3;

for(int i = 0; i < 32; i++){

memMap[i][0] = hour;

memMap[i][1] = min;

memMap[i][2] = temp;

memMap[i][3] = terminator;

hour += 4;

min += 4;

temp += 4;

terminator += 4;

}

//initialize schedule with one measurement at 00:00.

schedule[0][0] = 0;

schedule[0][1] = 0;;

for(int i = 1; i < 32; i++){

schedule[i][0] = invalid;

schedule[i][1] = invalid;

}

scheduleToMem();

//initialize validPoints

for(int i = 0; i < 32; i++){

if(schedule[i][0] != invalid){

validPoints++;

}

}

//initialize RTC subsystem and set alarm

PCONP |= (1<<9);

CCR = 0;

setAlarm(currentDataPoint);

CCR |= (1<<0);

while(1){

//if the base addresses the rover...

if(I2C2STAT == 0x60){

I2C2CONCLR = (1<<3);

I2C2CONSET = (1<<2);

while(I2C2STAT != 0x80){

}

commandBuffer = I2C2DAT;

//...and the base requests a data point

if(commandBuffer != UPSCHED){

I2C2CONCLR = (1<<3);

I2C2CONCLR = (1<<2);

while(I2C2STAT != 0xA0){

}

I2C2CONCLR = (1<<3);

I2C2CONSET = (1<<2);

while(I2C2STAT != 0xA8){

}

I2C2DAT = tempMem[memMap[commandBuffer][0]];

I2C2CONCLR = (1<<3);

I2C2CONSET = (1<<2);

while(I2C2STAT != 0xB8){

}

I2C2DAT = tempMem[memMap[commandBuffer][1]];

I2C2CONCLR = (1<<3);

I2C2CONSET = (1<<2);

while(I2C2STAT != 0xB8){

}

I2C2DAT = tempMem[memMap[commandBuffer][2]];

I2C2CONCLR = (1<<3);

I2C2CONSET = (1<<2);

while(I2C2STAT != 0xC0){

}

I2C2CONCLR = (1<<3);

I2C2CONSET = (1<<2);

}

//...or the base commands a schedule update

else{

I2C2CONCLR = (1<<3);

I2C2CONSET = (1<<2);

for(int i = 0; i < 32; i++){

schedule[i][0] = invalid;

schedule[i][1] = invalid;

}

for(int i = 0; i < 32; i++){

while(I2C2STAT != 0x80){

}

schedule[i][0] = I2C2DAT;

I2C2CONCLR = (1<<3);

I2C2CONSET = (1<<2);

while(I2C2STAT != 0x80){

}

schedule[i][1] = I2C2DAT;

I2C2CONCLR = (1<<3);

I2C2CONSET = (1<<2);

}

while(I2C2STAT != 0xA0){

}

I2C2CONCLR = (1<<3);

I2C2CONSET = (1<<2);

validPoints = 0;

for(int i = 0; i < 32; i++){

if(schedule[i][0] != invalid){

validPoints++;

}

}

currentDataPoint = 0;

setAlarm(currentDataPoint);

scheduleToMem();

}

}

//checks the time and stores the temperature if the time matches the alarm

if(MIN == alMin && HOUR == alHour){

if(currentDataPoint < validPoints && schedule[currentDataPoint][0] != invalid && schedule[currentDataPoint][1] != invalid){

storeTemp(currentDataPoint);

currentDataPoint++;

if(currentDataPoint >= validPoints){

currentDataPoint = 0;

}

setAlarm(currentDataPoint);

}

}

//prints the time

printf("Current Time: %02d:%02d:%02d\n", HOUR, MIN, SEC);

}

}

/\*

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

Name : Base\_Code.c

Author : Taylor Rainwater, Triston Luzanta

Version :

Copyright : $(copyright)

Description : implements base module functionality

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

\*/

#ifdef \_\_USE\_CMSIS

#include "LPC17xx.h"

#endif

#include <cr\_section\_macros.h>

#include <stdio.h>

//////////////////////////////////////

///LPC internal register addresses.///

//////////////////////////////////////

//MISC LPC register definitions

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

#define PCLKSEL1 (\*(volatile unsigned int\*) 0x400FC1AC)

#define PINSEL0 (\*(volatile unsigned int\*) 0x4002C000)

#define PINMODE0 (\*(volatile unsigned int\*) 0x4002C040)

#define PINMODE1 (\*(volatile unsigned int\*) 0x4002C044)

#define PINMODE\_OD0 (\*(volatile unsigned int\*) 0x4002C068)

#define FIO0DIR (\*(volatile unsigned int\*) 0x2009c000)

#define FIO0PIN (\*(volatile unsigned int\*) 0x2009c014)

//I2C1 LPC register definitions

#define I2C1SCLH (\*(volatile unsigned int\*) 0x4005C010)

#define I2C1SCLL (\*(volatile unsigned int\*) 0x4005C014)

#define I2C1CONSET (\*(volatile unsigned int\*) 0x4005C000)

#define I2C1CONCLR (\*(volatile unsigned int\*) 0x4005C018)

#define I2C1DAT (\*(volatile unsigned int\*) 0x4005C008)

#define I2C1STAT (\*(volatile unsigned int\*) 0x4005C004)

//I2C2 LPC register definitions

#define I2C2SCLH (\*(volatile unsigned int\*) 0x400A0010)

#define I2C2SCLL (\*(volatile unsigned int\*) 0x400A0014)

#define I2C2CONSET (\*(volatile unsigned int\*) 0x400A0000)

#define I2C2CONCLR (\*(volatile unsigned int\*) 0x400A0018)

#define I2C2DAT (\*(volatile unsigned int\*) 0x400A0008)

#define I2C2STAT (\*(volatile unsigned int\*) 0x400A0004)

////////////////////////////////////////////////////////

///Peripheral device and internal register addresses.///

////////////////////////////////////////////////////////

//GPIO expander device address

#define GPIOADDR 0x40

//GPIO Internal addresses

#define IODIRA 0x00

#define IODIRB 0x01

#define GPPUA 0x0C

#define GPPUB 0x0D

#define GPIOA 0x12

#define GPIOB 0x13

//Rover address

#define ROVER 0xAA

//Rover commands

#define UPSCHED 0xFF

///////////////////////////////////////////////////

///I2C protocol to communicate with LCD display.///

///////////////////////////////////////////////////

//sends start signal to SDA bus

void start(){

I2C1CONSET = (1<<3); // 3rd bit set to 1

I2C1CONSET = (1<<5); // 5th bit set to 1

I2C1CONCLR = (1<<3); // 3rd bit cleared to 0

while(!((I2C1CONSET>>3) & 1)){

// Wait for 3rd bit to 1

}

I2C1CONCLR = (1<<5); // 5th bit cleared to 0

}

//reads 8 bit data from slave device

int read(int final){

if (final == 0){

I2C1CONCLR = (1<<2); // 2nd bit cleared to 0

}

else if (final == 1){

I2C1CONSET = (1<<2); // 2nd bit set to 1

}

I2C1CONCLR = (1<<3); // 3rd bit clear to 0

while(!((I2C1CONSET>>3) & 1)){

// Wait for 3rd bit to 1

}

int temp = I2C1DAT;

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){

I2C1DAT = DATA + RW;

I2C1CONCLR = (1<<3); // 3rd bit set to 0

while(!((I2C1CONSET>>3) & 1)){

// Wait for 3rd bit to 1

}

}

//sends stop signal to SDA bus

void stop(){

I2C1CONSET = (1<<4);

I2C1CONCLR = (1<<3);

while(((I2C1CONSET>>4) & 1)){

// Wait for 4th bit to 1

}

}

////////////////////////////////////////////

///Communication link from base to rover.///

////////////////////////////////////////////

//sends start signal to SDA bus

void start2(){

I2C2CONSET = (1<<3); // 3rd bit set to 1

I2C2CONSET = (1<<5); // 5th bit set to 1

I2C2CONCLR = (1<<3); // 3rd bit cleared to 0

while(!((I2C2CONSET>>3) & 1)){

// Wait for 3rd bit to 1

}

I2C2CONCLR = (1<<5); // 5th bit cleared to 0

}

//reads 8 bit data from slave device

int read2(int final){

if (final == 0){

I2C2CONCLR = (1<<2); // 2nd bit cleared to 0

}

else if (final == 1){

I2C2CONSET = (1<<2); // 2nd bit set to 1

}

I2C2CONCLR = (1<<3); // 3rd bit clear to 0

while(!((I2C2CONSET>>3) & 1)){

// Wait for 3rd bit to 1

}

int temp = I2C2DAT;

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 write2(int DATA, int RW){

I2C2DAT = DATA + RW;

I2C2CONCLR = (1<<3); // 3rd bit set to 0

while(!((I2C2CONSET>>3) & 1)){

// Wait for 3rd bit to 1

}

}

//sends stop signal to SDA bus

void stop2(){

I2C2CONSET = (1<<4);

I2C2CONCLR = (1<<3);

while(((I2C2CONSET>>4) & 1)){

// Wait for 4th bit to 1

}

}

//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++){

}

}

///////////////////

///LCD protocol.///

///////////////////

//writes a command to the LCD display

void writeLCDCommand(int command){

start();

write(GPIOADDR, 0);

write(GPIOA, 0);

write(command, 0);

stop();

start();

write(GPIOADDR, 0);

write(GPIOB, 0);

write(0x00, 0);

stop();

start();

write(GPIOADDR, 0);

write(GPIOB, 0);

write(0x04, 0);

stop();

start();

write(GPIOADDR, 0);

write(GPIOB, 0);

write(0x00, 0);

stop();

wait(0.0001);

}

//writes a character to the LCD display

void writeLCDData(char letter){

start();

write(GPIOADDR, 0);

write(GPIOA, 0);

write(letter, 0);

stop();

start();

write(GPIOADDR, 0);

write(GPIOB, 0);

write(0x02, 0);

stop();

start();

write(GPIOADDR, 0);

write(GPIOB, 0);

write(0x06, 0);

stop();

start();

write(GPIOADDR, 0);

write(GPIOB, 0);

write(0x02, 0);

stop();

wait(0.0001);

}

//////////////////////////////

///Primary base functions.////

//////////////////////////////

//global base variables.

char keys[4][4] = {{'1', '2', '3', 'A'},

{'4', '5', '6', 'B'},

{'7', '8', '9', 'C'},

{'\*', '0', '#', 'D'}};

int invalid = 0xff;

int schedule[32][2];

//returns the char of the current pressed key

char searchKey(){

FIO0DIR |= (1<<26);

FIO0PIN |= (1<<26);

FIO0PIN &= ~(1<<25);

FIO0PIN &= ~(1<<24);

FIO0PIN &= ~(1<<23);

if((FIO0PIN>>16) & 1){

return keys[0][0];

}

if((FIO0PIN>>15) & 1){

return keys[0][1];

}

if((FIO0PIN>>17) & 1){

return keys[0][2];

}

if((FIO0PIN>>18) & 1){

return keys[0][3];

}

FIO0DIR |= (1<<25);

FIO0PIN |= (1<<25);

FIO0PIN &= ~(1<<24);

FIO0PIN &= ~(1<<23);

FIO0PIN &= ~(1<<26);

if((FIO0PIN>>16) & 1){

return keys[1][0];

}

if((FIO0PIN>>15) & 1){

return keys[1][1];

}

if((FIO0PIN>>17) & 1){

return keys[1][2];

}

if((FIO0PIN>>18) & 1){

return keys[1][3];

}

FIO0DIR |= (1<<24);

FIO0PIN |= (1<<24);

FIO0PIN &= ~(1<<23);

FIO0PIN &= ~(1<<26);

FIO0PIN &= ~(1<<25);

if((FIO0PIN>>16) & 1){

return keys[2][0];

}

if((FIO0PIN>>15) & 1){

return keys[2][1];

}

if((FIO0PIN>>17) & 1){

return keys[2][2];

}

if((FIO0PIN>>18) & 1){

return keys[2][3];

}

FIO0DIR |= (1<<23);

FIO0PIN |= (1<<23);

FIO0PIN &= ~(1<<26);

FIO0PIN &= ~(1<<25);

FIO0PIN &= ~(1<<24);

if((FIO0PIN>>16) & 1){

return keys[3][0];

}

if((FIO0PIN>>15) & 1){

return keys[3][1];

}

if((FIO0PIN>>17) & 1){

return keys[3][2];

}

if((FIO0PIN>>18) & 1){

return keys[3][3];

}

return 'X';

}

//displays the keys pressed on the keypad

int recordKeyPress(){

char x = 'X';

char lastPress = 'X';

while(1){

x = searchKey();

if(x == '#'){

return '#';

}

if(x != 'X' && x != lastPress){

writeLCDData(x);

lastPress = x;

wait(0.1);

if(x == 'A' || x == 'B' || x == 'C' || x == 'D' || x == '\*'){

return x;

}

else{

return (x - '0');

}

}

if(x == 'X'){

lastPress = 'X';

}

}

}

//create a new schedule and store it in base memory

void createSchedule(){

int hour = invalid;

int min = invalid;

int k = invalid;

char msg[] = "Enter time ";

char point[2];

for(int i = 0; i < 32; i++){

wait(0.004);

writeLCDCommand(0x0f);

writeLCDCommand(0x01);

wait(0.004);

for(int i = 0; i < strlen(msg); i++){

writeLCDData(msg[i]);

}

sprintf(point, "%02d", (i+1));

for(int i = 0; i < strlen(point); i++){

writeLCDData(point[i]);

}

writeLCDData(':');

writeLCDData(' ');

k = recordKeyPress();

if(k == '#'){

break;

}

hour = (10 \* k);

wait(0.1);

k = recordKeyPress();

if(k == '#'){

break;

}

hour += k;

wait(0.1);

if(hour != invalid && hour < 24){

schedule[i][0] = hour;

}

else{

schedule[i][0] = invalid;

}

k = recordKeyPress();

if(k == '#'){

break;

}

min = (10 \* k);

wait(0.1);

k = recordKeyPress();

if(k == '#'){

break;

}

min += k;

wait(0.1);

if(min != invalid && min < 60){

schedule[i][1] = min;

}

else{

schedule[i][1] = invalid;

}

}

}

//send current schedule to rover

void updateSchedule(){

wait(0.004);

writeLCDCommand(0x0c);

writeLCDCommand(0x01);

wait(0.004);

char msg[] = "Updating schedule...";

for(int i = 0; i < strlen(msg); i++){

writeLCDData(msg[i]);

}

start2();

write2(ROVER, 0);

write2(UPSCHED, 0);

for(int i = 0; i < 32; i++){

write2(schedule[i][0], 0);

write2(schedule[i][1], 0);

}

stop2();

}

//displays the selected measurement from the rover

void displayMeasurment(){

int point = invalid;

int k = 'X';

char msg1[] = "Display point:";

char msg2[] = "Point ";

char msg3[2];

char msg4[] = "Time: ";

char msg5[100];

char msg6[] = "Temperature: ";

char msg7[100];

int data[3];

wait(0.004);

writeLCDCommand(0x0f);

writeLCDCommand(0x01);

wait(0.004);

for(int i = 0; i < strlen(msg1); i++){

writeLCDData(msg1[i]);

}

k = recordKeyPress();

if(k == '#'){

return 0;

}

point = (10 \* k);

wait(0.1);

k = recordKeyPress();

if(k == '#'){

return 0;

}

point += k;

wait(0.1);

start2();

write2(ROVER, 0);

write2((point - 1), 0);

start2();

write2(ROVER, 1);

data[0] = read2(1);

data[1] = read2(1);

data[2] = read2(0);

stop2();

wait(0.004);

writeLCDCommand(0x0c);

writeLCDCommand(0x01);

wait(0.004);

for(int i = 0; i < strlen(msg2); i++){

writeLCDData(msg2[i]);

}

sprintf(msg3, "%02d", point);

for(int i = 0; i < strlen(msg3); i++){

writeLCDData(msg3[i]);

}

wait(0.004);

writeLCDCommand(0xc0);

wait(0.004);

for(int i = 0; i < strlen(msg4); i++){

writeLCDData(msg4[i]);

}

sprintf(msg5, "%02d:%02d", data[0], data[1]);

for(int i = 0; i < strlen(msg5); i++){

writeLCDData(msg5[i]);

}

wait(0.004);

writeLCDCommand(0x94);

wait(0.004);

for(int i = 0; i < strlen(msg6); i++){

writeLCDData(msg6[i]);

}

sprintf(msg7, "%d", data[2]);

for(int i = 0; i < strlen(msg7); i++){

writeLCDData(msg7[i]);

}

writeLCDData(' ');

writeLCDData('C');

k = recordKeyPress();

if(k == '#'){

return 1;

}

}

int main(void) {

//initialize I2C1 pins on LPC

PINMODE0 |= (1<<1) | (1<<3);

PINMODE0 &= ~(1<<0) & ~(1<<2);

PINMODE\_OD0 |= (1<<0) | (1<<1);

PCONP |= (1<<19);

PCLKSEL1 &= ~(1<<6) & ~(1<<7);

I2C1SCLH = 5;

I2C1SCLL = 5;

PINSEL0 |= (1<<0) | (1<<1) | (1<<2) | (1<<3);

I2C1CONCLR = (1<<6);

I2C1CONSET = (1<<6);

//initialize I2C2 pins on LPC

PINMODE0 |= (1<<21) | (1<<23);

PINMODE0 &= ~(1<<20) & ~(1<<22);

PINMODE\_OD0 |= (1<<10) | (1<<11);

PCONP |= (1<<26);

PCLKSEL1 &= ~(1<<20) & ~(1<<21);

I2C2SCLH = 5;

I2C2SCLL = 5;

PINSEL0 |= (1<<21) | (1<<23);

PINSEL0 &= ~(1<<20) & ~(1<<22);

I2C2CONCLR = (1<<6);

I2C2CONSET = (1<<6);

//Initialize keypad

for(int i = 15; i < 19; i++){

FIO0DIR &= ~(1<<i);

}

PINMODE0 |= (1<<30) | (1<<31);

PINMODE1 |= (1<<0) | (1<<1) | (1<<2) | (1<<3) | (1<<4) | (1<<5);

//Initialize schedule

for(int i = 0; i < 32; i++){

schedule[i][0] = invalid;

schedule[i][1] = invalid;

}

//initialize the GPIO expander

start();

write(GPIOADDR,0);

write(IODIRA,0);

write(0x00,0);

stop();

start();

write(GPIOADDR,0);

write(IODIRB,0);

write(0x00,0);

stop();

start();

write(GPIOADDR,0);

write(GPPUA,0);

write(0x00,0);

stop();

start();

write(GPIOADDR,0);

write(GPPUB,0);

write(0x00,0);

stop();

//initialize LCD

wait(0.004);

writeLCDCommand(0x38);

writeLCDCommand(0x06);

writeLCDCommand(0x0f);

writeLCDCommand(0x01);

wait(0.004);

while(1){

wait(0.004);

writeLCDCommand(0x0c);

writeLCDCommand(0x01);

wait(0.004);

char msgA[] = "A: Get a data point.";

for(int i = 0; i < strlen(msgA); i++){

writeLCDData(msgA[i]);

}

wait(0.004);

writeLCDCommand(0xc0);

wait(0.004);

char msgB[] = "B: Create a new sche";

for(int i = 0; i < strlen(msgB); i++){

writeLCDData(msgB[i]);

}

wait(0.004);

writeLCDCommand(0x94);

wait(0.004);

char msgBcont[] = " -dule.";

for(int i = 0; i < strlen(msgBcont); i++){

writeLCDData(msgBcont[i]);

}

wait(0.004);

writeLCDCommand(0xd4);

wait(0.004);

char msgC[] = "C: Update schedule.";

for(int i = 0; i < strlen(msgC); i++){

writeLCDData(msgC[i]);

}

int input = recordKeyPress();

if(input == 'A'){

displayMeasurment();

}

if(input == 'B'){

createSchedule();

}

if(input == 'C'){

updateSchedule();

}

}

}