Daniel Ngo, Arjun Ganesan

ECE 4273 – Digital Design Lab

December 10, 2021

Final Project – What a Beautiful Time to be Alive

1. General Purpose

The purpose of this project is to implement a digital clock that keeps track of the time of day and an alarm clock value, both of which are displayed on an LCD module and configurable by the user using a 4x4 keypad. When the alarm time is reached by the clock, a melody alarm sounds, and on the 00, 15, 30, and 45 minutes past the hour, the appropriate sections of the Westminster chimes are played. We plan on displaying the time of day as well as the alarm time using an LCD module with a HD44780 controller, using a multiplexed switch keypad to allow user input to set the time of day and alarm clock time, and the LPC1769's built in Timer, Alarm, and PWM modules to implement the time-of-day clock, the alarm clock, and the alarm and chimes sounds.

Below is a list of functions and features of the circuit, along with their point value and status:

Feature	**Points**	**Status**
Time of Day	1	Working
Alarm Clock	1	Working
Chimes	1	Working
Melody Alarm	0.5	Working
Serially Controlled Display (I2C)	0.5	Working
Character LCD w/ HD44780	0.5	Working
Keypad	0.5	Working
TOTAL	5	Working

2. User Manual

Introduction: Ensure that the necessary software has been built and uploaded to the LPC1769 microcontroller. Once the program has been uploaded to the LPC1769, the clock timer will start counting. Both the clock and alarm times are initialized by the software, but the user can modify these values after the program begins. Initially, the display shows the current time and alarm time.

To set the clock time, the 'C' key must be pressed. To set the alarm time, the 'A' key must be pressed. In either case, the LCD display will prompt the user for the time in the form HH:MM:SS, where each value must be set sequentially. After entering in a time, the display will go back to showing the clock time and alarm time, updated as necessary. When the alarm time

is reached a quick melody will be played through the speaker, and a message notifying the user the alarm time has been reached will be displayed on the LCD

The appropriate Westminster chimes play on the 00-, 15-, 30-, and 45-minute markers of each hour. However, if the clock system is busy (user is setting alarm/clock time or another sound is playing), the chimes or alarm will not play. A message regarding the specific chime will also be displayed on the LCD to the user.

3. Design Analysis – Component Values and Configuration

a. 4x4 Multiplexed Switch Keypad

The multiplexed switch keypad is configured as active-low. We drive the selected row low and the others with high impedance to determine the correct row and column of the selected key.

b. LM386 Audio Amplifier

For our audio amplifier, we use a 10kOhm-1kOhm voltage divider on the input, 100nF capacitors to the LM386 audio amplifier (Bypass (Pin 7) to GND, Vs (Pin 6) to GND (Pin 4), Vout (Pin 5) to GND (Pin 4)). We also have a 1mF load capacitor (Vout (Pin5) to Speaker). See the hardware schematic for more details.

c. Character LCD w/ HD44780

To interface between the LCD and the LPC1769, we use an MCP23017 I/O expander using I2C. To find our lower bound for the SDA and SCLK pull-up resistor value, we consider weakest device I_{OL} parameter: 3.0mA (SDA in MCP23017). Thus, our lower bound for R:

$$R > \frac{V_{DD}}{I_{OL}} = \frac{3.3V}{3.0mA} = 1100\Omega$$

Setting the PCLKSEL0's bit 14 and 15 to 00, and choosing I2C0SCLH=I2C0SCLL=5 to obtain an I2C frequency of 100kHz and 50% duty cycle, we chose a I2C clock value to be:

$$f_{I2C} = \frac{CCLK/4}{(I2C0SCLH + I2C0SCLL)} = \frac{1MHz}{10} = 100kHz$$

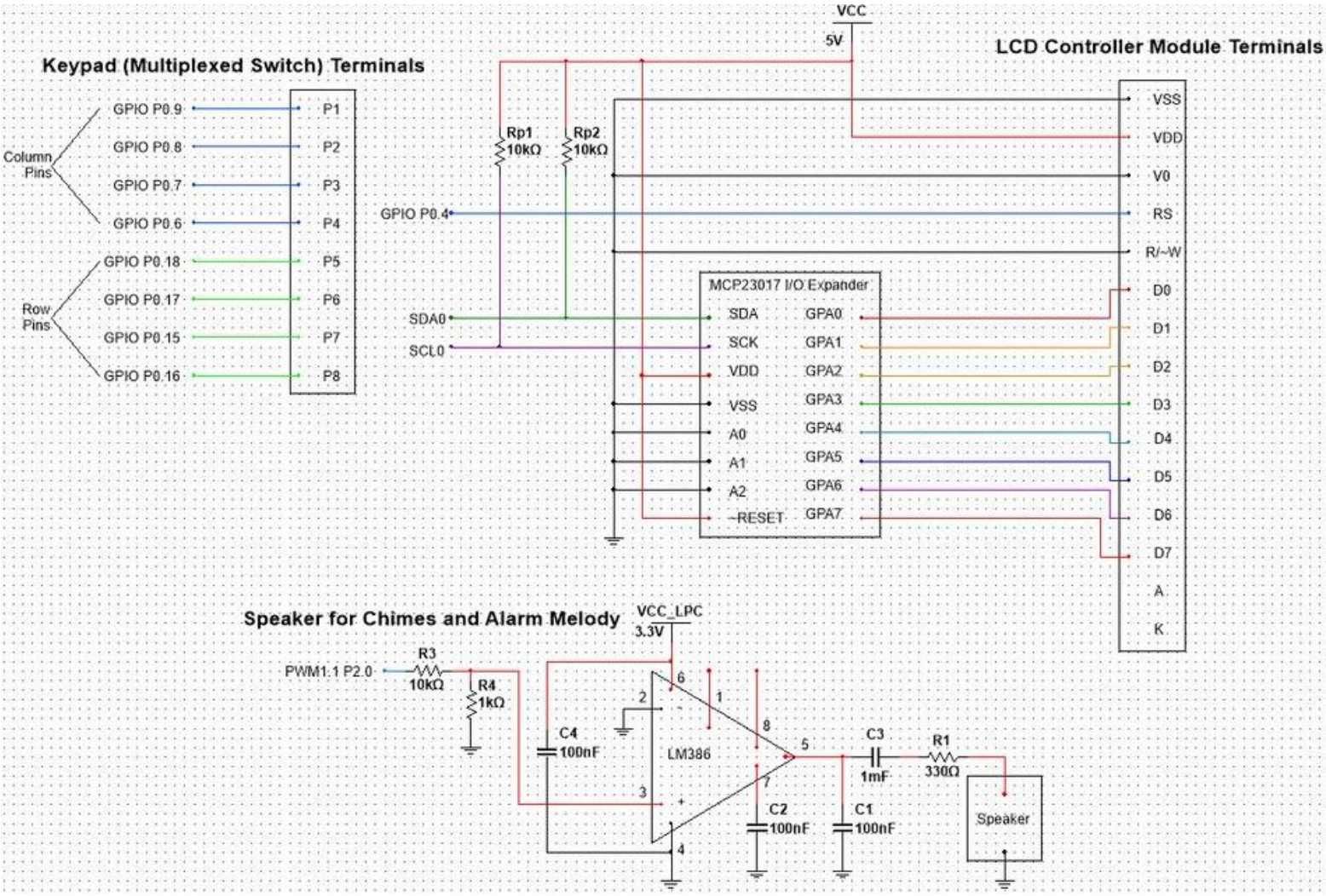
Finding the upper bound for our pull-up resistor, and checking that $t_r \ll t_{cycle}$:

$$R \ll \frac{t_{cycle}}{3C} = \frac{1}{3fC} = \frac{1}{3(100kHz)(3 \cdot 10pF)} = 111k\Omega$$

$$t_r \approx 3RC = 3(10k\Omega)(3 \cdot 10pF) = 90ns \ll 1\mu s = \frac{1}{f_{I2C}} = t_{cycle}$$

Thus, our pull-up resistor value $R = 10k\Omega$ satisfies the lower bound and upper bound.

4. Hardware Schematic



5. **Software (Source Code)**
    a. **Alarm and Chimes Code**

```
/*
 * AlarmAndChimes.h
 *
 *  Created on: Nov 18, 2021
 *      Author: Arjun Ganesan and Daniel Ngo
 */

#ifndef AlarmAndChimes_H_
#define AlarmAndChimes_H_

// PWM Registers
#define PWM1MCR (*(volatile unsigned int *) 0x40018014)
#define PWM1MR0 (*(volatile unsigned int *) 0x40018018)
#define PWM1MR1 (*(volatile unsigned int *) 0x4001801C)
#define PWM1PCR (*(volatile unsigned int *) 0x4001804C)
#define PWM1TCR (*(volatile unsigned int *) 0x40018004)
#define PWM1TC (*(volatile unsigned int *) 0x40018008)
#define PWM1LER (*(volatile unsigned int *) 0x40018050)

/**
 * Initialize PWM to FREQ = PCLK / periodParam
 */
void setupPWM(int period, float dc);

void setAlarmTime(int sec, int min, int hour);

void alarmListener();

void chimeListener();

void playChimes(int* chime);

void playAlarm();

#endif /* AlarmAndChimes_H_ */
```

```
/*
 * AlarmAndChimes.c
 *
 *  Created on: Nov 18, 2021
 *      Author: Arjun Ganesan and Daniel Ngo
```

```
 */

#include "GeneralLPC.h"
#include "AlarmAndChimes.h"
#include "RTC.h"

// C4, G4, E4, A4, B4, A4, G4, E4, C4
int melody_notes[] = { 261, 392, 329, 440, 494, 440, 392, 329, 261 };

// time length of quarter note
float qNoteTime = 0.50;

// G#4, F#4, E4, B3
int wmChime1[] = { 415, 370, 330, 247 };
// E4, G#4, F#4, B3
int wmChime2[] = { 330, 415, 370, 247 };
// E4, F#4, G#4, E4
int wmChime3[] = { 330, 370, 415, 330 };
// G#4, E4, F#4, B3
int wmChime4[] = { 415, 330, 370, 247 };
// B3, F#4, G#4, E4
int wmChime5[] = { 247, 370, 415, 330 };

/**
 * Initialize PWM to FREQ = PCLK / periodParam
 */
void setupPWM(int period, float dc) {
    PCLKSEL0 |= (1<<12);                    // PWM PCLK = CCLK/1
    PWM1TCR = 0; PWM1TC = 0;
    PWM1MR0 = period;                       // set PCLK cycle period (8-bit
equivalent)
    PINSEL4 |= ~(1<<1); PINSEL4 |= (1<<0);  // Configure P2.0 as PWM1.1
    PWM1MCR = (1<<1);                       // Reset counter on MR0 match
    PWM1PCR = (1<<9);                       // Single edge mode and enable PWM1.1
only
    PWM1TCR = (1<<0) | (1<<3);              // Enable counter and PWM modeW
    PWM1MR1 = (int)(period * dc);           // change MR1 to output new 8-bit D
sample value (50% DC)
    PWM1LER = (0b11<<0);                    // set bits 1:0 to 11 to use new MR1
and MR0 values
}

void setAlarmTime(int sec, int min, int hour) {
    ALSEC = sec;
    ALMIN = min;
```

```
    ALHOUR = hour;
}

int alarm_flg = 0;  // marks if alarm has played
int chime_flg = 0;  // marks if chime has played

// checks to see if alarm should go off
void alarmListener() {
    // if alarm has not already played and the alarm time is close to the current
time, set alarm flag to 1 and play alarm
    // gives some buffer time by only checking if the current time is some
seconds greater than the alarm time rather than checking if the current second
matches the exact alarm second
    if (!alarm_flg &&
        ((HOUR == ALHOUR && MIN == ALMIN && SEC >= ALSEC))
    ) {
        alarm_flg = 1;
        LCDWriteCommand(0x01); wait_us(4000);
        LCDWriteString("ALARM!", 6);
        playAlarm();
    } else if (MIN != ALMIN) alarm_flg = 0; // if the current minute does not
match the alarm minute, then the alarm flag is reset to 0
}

// checks to see if chimes should go off
void chimeListener() {
    if (!chime_flg) {
        if (MIN == 00) {
            chime_flg = 1;
            // clear display and move cursor to top left
            LCDWriteCommand(0x01); wait_us(4000);
            LCDWriteString("TOP OF THE HOUR!", 16);
            playChimes(wmChime2);
            playChimes(wmChime3);
            playChimes(wmChime4);
            playChimes(wmChime5);
            wait(1);
            // chimes whole notes (E3) for each hour (1 strike for 1 o'clock, 2
strikes for 2 o'clock, etc.)
            for(int i = 0; i < HOUR % 12; i++) {
                setupPWM((int)(4000000/165), 0.5);
                wait(4*qNoteTime);
                setupPWM(1000, 0);
                wait(0.25*qNoteTime);
            }
```

```
        } else if (MIN == 15) {
            chime_flg = 1;
            LCDWriteCommand(0x01); wait_us(4000);
            LCDWriteString("15 PAST THE HOUR!", 17);
            playChimes(wmChime1);
        } else if (MIN == 30) {
            chime_flg = 1;
            LCDWriteCommand(0x01); wait_us(4000);
            LCDWriteString("HALF PAST THE HOUR!", 19);
            playChimes(wmChime2);
            playChimes(wmChime3);
        } else if (MIN == 45 && SEC <= 3) {
            chime_flg = 1;
            LCDWriteCommand(0x01); wait_us(4000);
            LCDWriteString("15 TIL THE HOUR!", 19);
            playChimes(wmChime4);
            playChimes(wmChime5);
            playChimes(wmChime1);
        } else; // do nothing
    } else if (MIN % 15 != 0) chime_flg = 0;
}

void playChimes(int* chime) {
    for(int i = 0; i < 4; i++) {
        setupPWM((int)(4000000/chime[i]), 0.5);
        if(i != 3) {
            wait(qNoteTime);
        }
        else {
            wait(2*qNoteTime);
        }
    }
    setupPWM(1000, 0.0);
}

void playAlarm() {
    for(int i = 0; i < 8; i++) {
        setupPWM((int)(4000000/melody_notes[i]), 0.5);
        wait(qNoteTime);
    }
    setupPWM(1000, 0.0);
}
```

b. General LPC Code

```
/*
 * GeneralLPC.h
 *
 *  Created on: Nov 18, 2021
 *      Author: Arjun Ganesan and Daniel Ngo
 */

#ifndef GeneralLPC_H_
#define GeneralLPC_H_

#define PCLKSEL0 (*(volatile unsigned int *) 0x400FC1A8)
#define PINSEL1 (*(volatile unsigned int *) 0x4002C004)
#define PINSEL4 (*(volatile unsigned int *) 0x4002C010)
#define FIO0DIR (*(volatile unsigned int *) 0x2009C000)
#define FIO0PIN (*(volatile unsigned int *) 0x2009C014)
#define T0TCR (*(volatile unsigned int *) 0x40004004)
#define T0TC (*(volatile unsigned int *) 0x40004008)

void wait_us(int us);
void wait(float sec);

#endif /* GeneralLPC_H_ */
```

```
/*
 * GeneralLPC.c
 *
 *  Created on: Nov 18, 2021
 *      Author: Arjun Ganesan and Daniel Ngo
 */

#include "GeneralLPC.h"
#include "I2C.h"

//functions taken from Timer Lecture
void wait_us(int us){
    int start = T0TC;  // note starting time
    T0TCR |= (1<<0);   // start timer
    while ((T0TC-start)<us) {} // wait for time to pass
}
void wait(float sec){
    wait_us(sec*1000000); // convert seconds to microseconds
}
```

c. I2C Code

```
/*
 * I2C.h
 *
 *  Created on: Nov 18, 2021
 *      Author: Arjun Ganesan and Daniel Ngo
 */

#ifndef I2C_H_
#define I2C_H_

#define I2C0CONSET (*(volatile unsigned int *)0x4001C000)
#define I2C0STAT (*(volatile unsigned int *)0x4001C004)
#define I2C0DAT (*(volatile unsigned int *)0x4001C008)
#define I2C0SCLH (*(volatile unsigned int *)0x4001C010)
#define I2C0SCLL (*(volatile unsigned int *)0x4001C014)
#define I2C0CONCLR (*(volatile unsigned int *)0x4001C018)

// MCP23017 I/O expander definitions
#define MCP_OPCODE (volatile unsigned int)0b0100000
#define MCP_GPIOA (volatile unsigned int)0x12
#define MCP_IODIRA (volatile unsigned int)0x00

// READ/WRITE bit constants
#define WRITE (volatile unsigned int)0
#define READ (volatile unsigned int)1

void start();

void stop();

void write(int data);

void writeToGPIOA(int data);

void setup_lpc_for_I2C();

void setup_mcp();

#endif /* I2C_H_ */
```

```
/*
 * I2C.c
```

```
 *
 *  Created on: Nov 18, 2021
 *      Author: Arjun Ganesan and Daniel Ngo
 */

#include "GeneralLPC.h"
#include "I2C.h"

void start() {
    I2C0CONSET = (1<<3); // set SI bit
    I2C0CONSET = (1<<5); // set STA bit
    I2C0CONCLR = (1<<3); // clear SI bit
    while (((I2C0CONSET>>3) & 1) != 1); // wait for SI bit to return to 1
    I2C0CONCLR = (1<<5); // clear STA bit
}
void stop() {
    I2C0CONSET = (1<<4); // set STO bit
    I2C0CONCLR = (1<<3); // clear SI bit
    while (((I2C0CONSET>>4) & 1) != 0); // wait for STO bit to return to 0
}

/**
 * @param 8-bit input data to write
 * write data to I2C given 8-bit data
 */
void write(int data){
    I2C0DAT = data; // assign data to I2CxDAT
    I2C0CONCLR = (1<<3); // clear SI bit
    while (((I2C0CONSET>>3) & 1) != 1); // wait for SI bit to return to 1
    // optional: check I2CxSTAT for acknowledge
}

/**
 * @param 8-bit input data to write
 * writes data to GPIOA
 */
void writeToGPIOA(int data){
    start();
    write((MCP_OPCODE<<1) | (WRITE<<0));
    write(MCP_GPIOA);
    write(data);
    stop();
}

/**
```

```
 * Sets up microcontroller for I2C operation
 */
void setup_lpc_for_I2C() {
    // set up PINSEL for I2C SDA0
    PINSEL1 &= ~(1<<23); PINSEL1 |= (1<<22);

    // set up PINSEL for I2C SCL0
    PINSEL1 &= ~(1<<25); PINSEL1 |= (1<<24);

    // FREQUENCY = PCLK / (I2CxSCLH + I2CxSCLL)
    // PCLK = CCLK/4 by default but can change with PCLKSELx
    // CCLK default is 4 MHz

    // set up PCLKSEL0 bit 15:14 to 00 (PCLK_I2C0)
    PCLKSEL0 &= ~(1<<15) & ~(1<<14);

    // f=100kHz, 50% duty cycle
    I2C0SCLL = 5;
    I2C0SCLH = I2C0SCLL + 1;

    // clr and reset en bit
    I2C0CONCLR = (1<<6);
    I2C0CONSET = (1<<6);
}

 /**
 * Sets up IO Expander for I2C operation
 */
void setup_mcp() {
    // configure GPIOA as output
    start();
    write((MCP_OPCODE<<1) | (WRITE<<0));
    write(MCP_IODIRA);
    write(0x00);
    stop();
}
```

d. LCD Module Code

```
/*
 * LCDModule.h
 *
 *  Created on: Nov 18, 2021
 *      Author: Arjun Ganesan and Daniel Ngo
 */

#ifndef LCDModule_H_
#define LCDModule_H_

#define RSLCDPin (volatile unsigned int) 4
#define ELCDPin (volatile unsigned int) 5

void initLCD();

void LCDWriteCommand(int commandCode);

void LCDWriteData(int data);

void LCDWriteString(char string[], int length);

#endif /* LCDModule_H_ */
```

```
/*
 * LCDModule.c
 *
 *  Created on: Nov 18, 2021
 *      Author: Arjun Ganesan and Daniel Ngo
 */

#include "GeneralLPC.h"
#include "I2C.h"
#include "LCDModule.h"

void initLCD() {
    // configure RS, R/W, and E as outputs to LCD and drive low
    FIO0DIR |= (1<<RSLCDPin);
    FIO0DIR |= (1<<ELCDPin);
    FIO0PIN &= ~(1<<RSLCDPin);
    FIO0PIN &= ~(1<<ELCDPin);

    // configure data signals (DB0-DB7) as outputs to LCD
```

```
    // we configure mpc io expander to set as output already
    setup_mcp();

    wait_us(4000); // wait 4 ms
    LCDWriteCommand(0x38); // write 0x38
    LCDWriteCommand(0x06); // write 0x06
    LCDWriteCommand(0x0c); // write 0x0c
    LCDWriteCommand(0x01); // write 0x01
    wait_us(4000); // wait 4 ms
}

void LCDWriteCommand(int commandCode) {
    // update D0-D7 with command code
    writeToGPIOA(commandCode);
    // drive RS low to indicate command
    FIO0PIN &= ~(1<<RSLCDPin);
    // drive E high then low (might need short delay)
    FIO0PIN |= (1<<ELCDPin);
    wait_us(10);
    FIO0PIN &= ~(1<<ELCDPin);
    // wait 100 us
    wait_us(100);
}

void LCDWriteData(int data) {
    // update D0-D7 with data
    writeToGPIOA(data);
    // drive RS high to indicate data
    FIO0PIN |= (1<<RSLCDPin);
    // drive E high then low (might need short delay)
    FIO0PIN |= (1<<ELCDPin);
    wait_us(10);
    FIO0PIN &= ~(1<<ELCDPin);
    // wait 100 us
    wait_us(100);
}

void LCDWriteString(char string[], int length) {
    for(int i = 0; i < length; ++i) {
        LCDWriteData(string[i]);
    }
}
```

e. RTC Code

```
/*
 * RTC.h
 *
 *  Created on: Nov 18, 2021
 *      Author: Arjun Ganesan and Daniel Ngo
 */

#ifndef RTC_H_
#define RTC_H_

// RTC Registers (P572)
#define CCR (*(volatile unsigned int *)0x40024008)

#define SEC (*(volatile unsigned int *)0x40024020)
#define MIN (*(volatile unsigned int *)0x40024024)
#define HOUR (*(volatile unsigned int *)0x40024028)

#define ALSEC (*(volatile unsigned int *)0x40024060)
#define ALMIN (*(volatile unsigned int *)0x40024064)
#define ALHOUR (*(volatile unsigned int *)0x40024068)

#define CTIME0 (*(volatile unsigned int *)0x40024014)

#endif /* RTC_H_ */
```

f. Driver Code (Main)

```
/*
===============================================================================
 Name        : FinalProject-DigitalClocks.c
 Author      : $(Arjun Ganesan, Daniel Ngo)
 Version     :
 Copyright   : $(copyright)
 Description : main definition
===============================================================================
*/

#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include <cr_section_macros.h>
#include "AlarmAndChimes.h"
#include "GeneralLPC.h"
#include "I2C.h"
#include "LCDModule.h"
#include "RTC.h"

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

// All on Port 0
int swColPins[] = {6, 7, 8, 9};
int swRowPins[] = {16, 15, 17, 18};

void setupKeypadPins() {
  // set all column pins to input
  for (int i = 0; i < 4; i++) {
    FIO0DIR &= ~(1<<swColPins[i]);
  }
}

/**
 * updates rowAndCol w/ pressed the row and column of the keypad
 */
void checkKeypadPress(int* rowAndCol, int loop) {
  int btnR = -1;
  int btnC = -1;

  do {
    // constantly go thru each row of keypad
```

```
    for (int r = 0; r < 4; r++) {
      // select current row (set to output)
      FIO0DIR |= (1<<swRowPins[r]); // output
      FIO0PIN &= ~(1<<swRowPins[r]);  // drive low

      // check which key is pressed
      for (int c = 0; c < 4; c++) {
        if(~(FIO0PIN>>swColPins[c]) & 1) {
          // wait until user lets go
          while (~(FIO0PIN>>swColPins[c]) & 1);

          // update pressed key row and column
          btnR = r;
          btnC = c;
          FIO0PIN |= (1<<10); // turn on test LED
          break;
        } else {
            btnR = -1;
            btnC= -1;
            FIO0PIN &= ~(1<<10); // turn off test LED
        }
      }

      wait_us(1);

      // update input array
      rowAndCol[0] = btnR;
      rowAndCol[1] = btnC;

      // de-select current row (set to input) for next loop
      FIO0DIR &= ~(1<<swRowPins[r]);
      wait_us(1);

      // key has been pressed already
      if (btnR != -1 && btnC != -1) break;
    }
  } while (loop && btnR == -1 && btnC == -1);
}

void setTime(int sec, int min, int hour) {
  CCR &= ~(1<<0); // disable clock
  CCR |=(1<<1); // reset clock

  // assign time
  SEC = sec;
```

```
  MIN = min;
  HOUR = hour;

  CCR = (1<<0); // enable clock
}

// Takes row and column of buton press and decodes the value of the button
char decodeKeyPress(int r, int c) {
  if (r==0 && c==0) return '1';
  else if (r==0 && c==1) return '2';
  else if (r==0 && c==2) return '3';
  else if (r==0 && c==3) return 'A';
  else if (r==1 && c==0) return '4';
  else if (r==1 && c==1) return '5';
  else if (r==1 && c==2) return '6';
  else if (r==1 && c==3) return 'B';
  else if (r==2 && c==0) return '7';
  else if (r==2 && c==1) return '8';
  else if (r==2 && c==2) return '9';
  else if (r==2 && c==3) return 'C';
  else if (r==3 && c==0) return '*';    // not used currently
  else if (r==3 && c==1) return '0';
  else if (r==3 && c==2) return '#';    // not used currently
  else if (r==3 && c==3) return 'D';    // not used currently
  else;
  return 'x';
}

/**
 * get hours, min, secs from key presses (in order)
 * assume user is perfect and only enters valid values
 */
void getTimeFromKeyPress(int* HMS, char* time) {
  int numRC[2];

  // move cursor to second line
  LCDWriteCommand(0x80 + 0x40);
  // makes cursor visible and blinking (to indicate waiting for user input)
  LCDWriteCommand(0x0f);
  LCDWriteString(time, 8);
  // moves cursor back to start of second row (tens place of hour should be
blinking)
  LCDWriteCommand(0x80 + 0x40);

  // hours digits
```

```
  checkKeypadPress(numRC, 1);
  char h10Char = decodeKeyPress(numRC[0], numRC[1]);
  LCDWriteData(h10Char);
  int h10 = 10*(h10Char - '0');

  checkKeypadPress(numRC, 1);
  char h1Char = decodeKeyPress(numRC[0], numRC[1]);
  int h1 = h1Char - '0';
  LCDWriteData(h1Char);

  LCDWriteData(':');

  // minutes digits
  checkKeypadPress(numRC, 1);
  char m10Char = decodeKeyPress(numRC[0], numRC[1]);
  LCDWriteData(m10Char);
  int m10 = 10*(m10Char- '0');

  checkKeypadPress(numRC, 1);
  char m1Char = decodeKeyPress(numRC[0], numRC[1]);
  int m1 = m1Char - '0';
  LCDWriteData(m1Char);

  LCDWriteData(':');

  // seconds digits
  checkKeypadPress(numRC, 1);
  char s10Char = decodeKeyPress(numRC[0], numRC[1]);
  int s10 = 10*(s10Char - '0');
  LCDWriteData(s10Char);
  checkKeypadPress(numRC, 1);
  char s1Char = decodeKeyPress(numRC[0], numRC[1]);
  LCDWriteData(s1Char);
  int s1 = s1Char - '0';

  HMS[0] = h10+h1;
  HMS[1] = m10+m1;
  HMS[2] = s10+s1;
}

// converts time to string, primarily used to display time on LCD
void getStringFromTime(int hour, int min, int sec, char* s) {
  s[0] = '0' + hour/10;
  s[1] = '0' + hour%10;
  s[2] = ':';
```

```
  s[3] = '0' + min/10;
  s[4] = '0' + min%10;
  s[5] = ':';

  s[6] = '0' + sec/10;
  s[7] = '0' + sec%10;
}

int main(void) {
  FIO0DIR |= (1<<10);  // TEST output LED
  setupKeypadPins();
  setup_lpc_for_I2C();

  int keyRC[2]; // row and col of pressed key

  char currentTimeStr[8];
  char alarmTimeStr[8];

  setTime(0, 0, 0);
  setAlarmTime(30, 0, 0);

  initLCD();

    while(1) {
      // checks if chime or alarm should go off
      chimeListener();
      alarmListener();

      LCDWriteCommand(0x0c); // set LCD to display only (makes cursor invisible)
      checkKeypadPress(keyRC, 0);
      char key = decodeKeyPress(keyRC[0], keyRC[1]);

      // set time clock (C)
      while (key == 'C') {
        int HMS[3];
      LCDWriteCommand(0x01); wait_us(4000);
      LCDWriteString("SET TIME: ", 10);
        getTimeFromKeyPress(HMS, currentTimeStr);
        setTime(HMS[2], HMS[1], HMS[0]);
        key = 'x';
      }

      // set alarm clock (A)
      while (key == 'A') {
```

```
        int HMS[3];
      LCDWriteCommand(0x01); wait_us(4000);
      LCDWriteString("SET ALARM: ", 11);
      getTimeFromKeyPress(HMS, alarmTimeStr);
      setAlarmTime(HMS[2], HMS[1], HMS[0]);
      key = 'x';
      }

    // get times from clock and alarm
    getStringFromTime(HOUR, MIN, SEC, currentTimeStr);
    getStringFromTime(ALHOUR, ALMIN, ALSEC, alarmTimeStr);

    // clear display and move cursor to top left
    LCDWriteCommand(0x01); wait_us(4000);

    // display clock time on LCD
    LCDWriteString("TIME : ", 7);
    LCDWriteString(currentTimeStr, 8);

    // move cursor to second line
    LCDWriteCommand(0x80 + 0x40);

    // display alarm time on LCD
    LCDWriteString("ALARM: ", 7);
    LCDWriteString(alarmTimeStr, 8);

    // allow LCD to process the info
    wait(0.10);
    }
    return 0;
}
```

6. **Supporting Documentation**
    a. **Breadboard Circuit Photograph**