EE 3501 Embedded Systems

Spring Semester

TIME AND TEMPERATURE DISPLAY

Kennesaw State University

Marietta, GA

Professor Craig Chin

Mitch Walker

July 19, 2024

Objective

The objective of this project is to design a system to display the current time and temperature on an LCD screen. The time must be able to be set using an external keypad and the temperature must be able to be displayed in either Celsius or Fahrenheit, using the keypad to switch between the two.

Apparatus List

- STM32-F401RE Microcontroller
- 4x4 Mechanical Keypad
- 10Kohm Potentiometer
- LM35DZ Temperature Sensor
- P181B LCD Display
- Jumper Cables
- Breadboard

Engineering Design

Variables

Before the main function, a number of global variables are defined. They are defined as follows:

- Int second – Tracks the current second
- Int minute – Tracks the current minute
- Int hour - Tracks the current hour
- Int temp - Tracks the current temperature
- Int low and int high – Used for setting and checking Digital I/O pin states
- String noon_string – Used to display AM/PM on LCD
- Bool fah_flag – Used to determine weather or not to display temperature in Fahrenheit or Celsius

Functions

There are five primary functions called within the main function. Below is the call graph of the program.

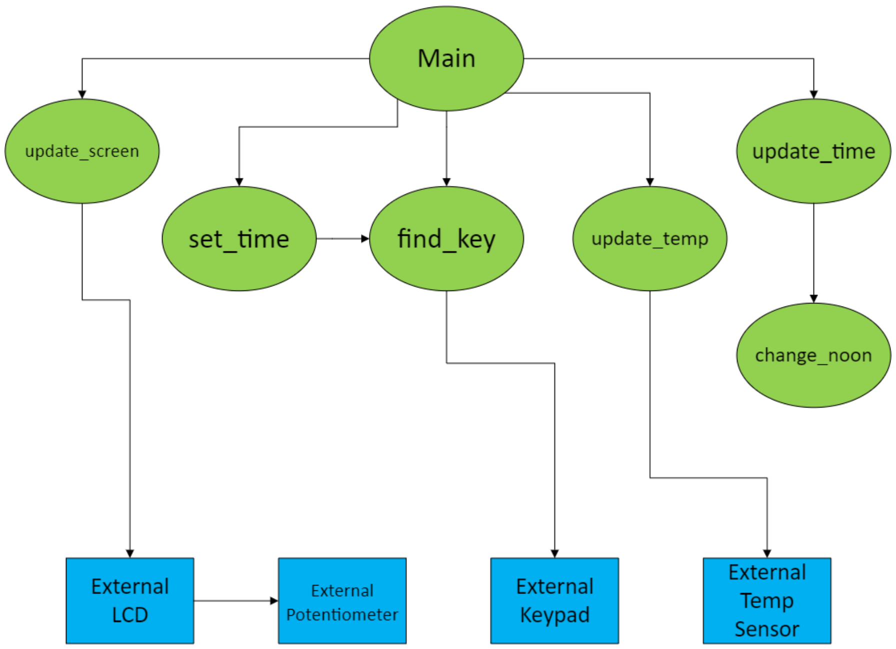

Figure 1: Program call graph

update_screen:

update_screen does not take in any variables and does not return any variables. It first clears the LCD screen, then uses the global variables to print the current hour, minute, second, temperature, F/C (depending on the state of fah_bool: True for F, and False for C) and AM/PM. If any of the aforementioned variables (with the exception of noon_string since it's a string) are less than 10 (therefore being a single digit), update_screen first prints a 0 to the LCD to maintain the required format. Below is the logical flowchart for update_screen.

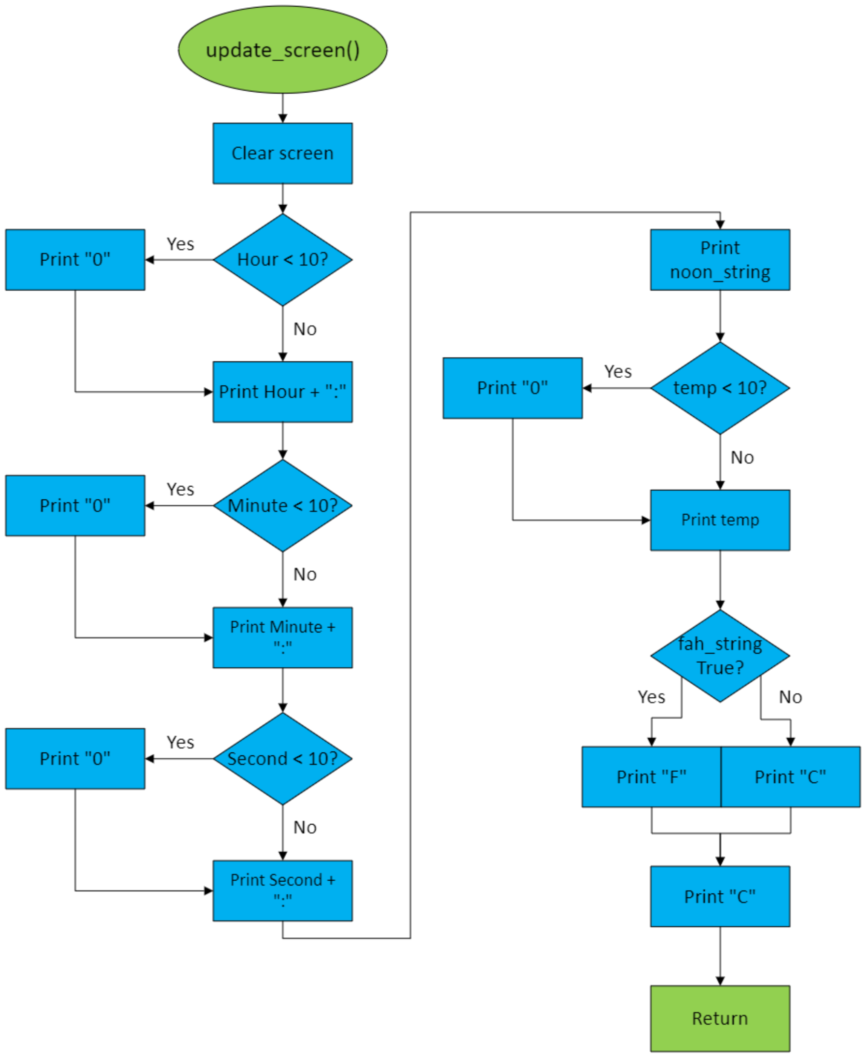

Figure 2: update_screen flowchart

Set_time

Set_time takes in no variables and returns nothing. It allows the user to set the current hour, minute, and AM/PM. It first clears the screen, then enters a while loop to set the hour. This loop calls the function find_key. If the user presses nothing, the loop simply does another iteration and calls find_key again. If the user pressed anything other than '*', it converts the char to an int and adds it to an int variable called 'input'. Once the user presses '*', it makes checks to ensure that input is within a range of 1-12. If it is not, then "ERROR" is printed on the LCD, input is reset to 0, and the loop starts over. If input is within a range of 1-12, then the global hour variable is set to the value stored in input, and the while loop is exited.

Once the hour is set, another while loop starts for setting the minute. The process is identical to setting the hour but the range that is checked after '*' is pressed is 1-59 instead of 1-12. The function them prompts the user to press 1 to set AM, and 2 to set PM. If anything else is pressed the function displays "ERROR" to the LCD and it prompts the user again. Below is a flowchart showing the logic to set the hour.

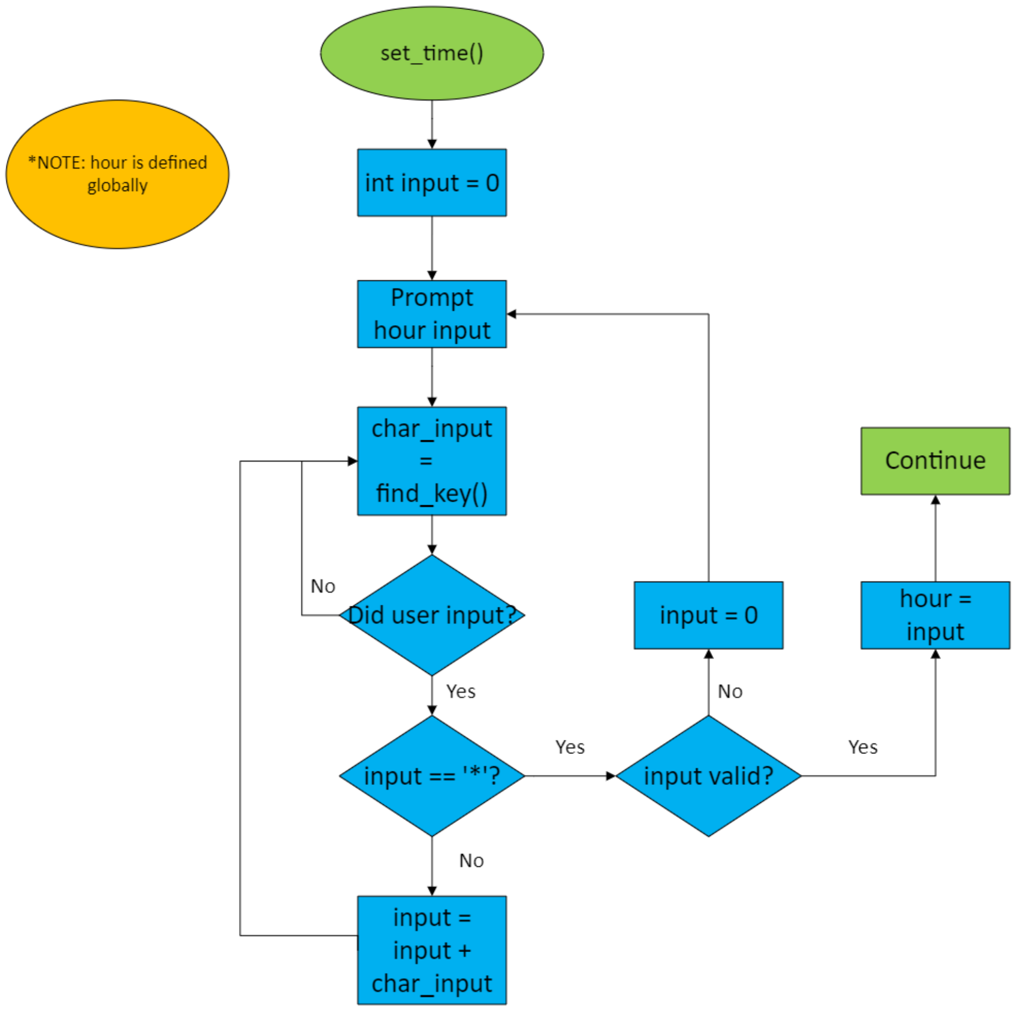

Figure 3: set_time flow chart to set hour

<u>Find_key</u>

Find_key takes in no variables and returns a character based on which key, if any, are pressed on an external keypad. The function consists of two for loops; one nested within the other. The outer for loop sets one of the row pins to high, loops through the inner for loop, then sets the pin to low if the function didn't return within the inner for loop. The inner for loop checks the column pins and if one is high, sets the row pin to low and returns a character from a 2x2 character array indexed by the values of the for loops. If no pin reads high the inner loop exits, the outer loop iterates, then the inner loop runs again. If the outer loop exits without returning a character, then a null character ('\0') is returned. Below is a flow chart showing the logic of the function.

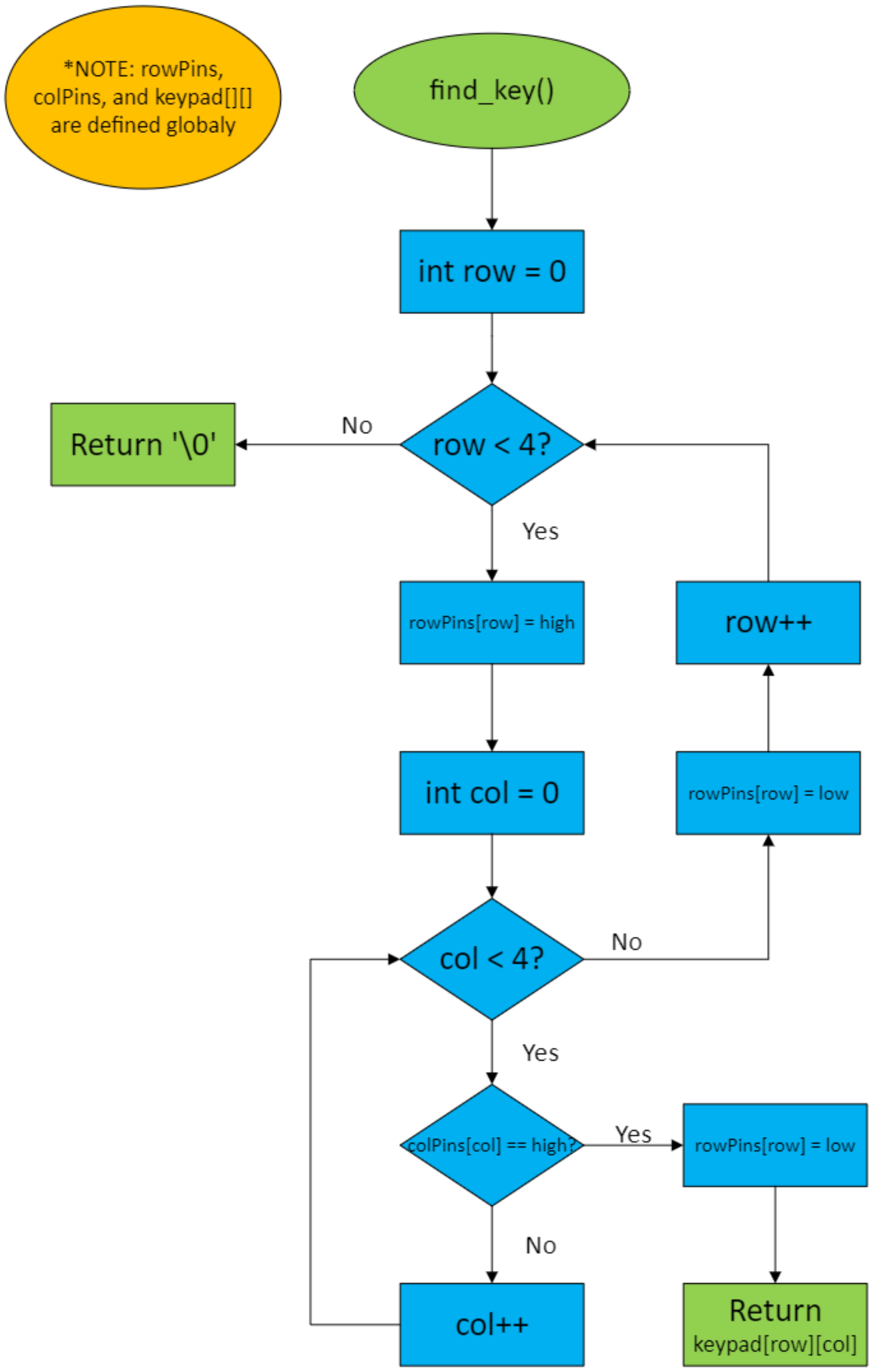

Figure 4: find_key flowchart

Update_temp

Update_temp takes in no variable and returns nothing. Its purpose is to read in from the external temperature sensor and set the global temperature variable to the current temperature. It does this by reading the voltage from sensor then converting the reading to the temperature in Celsius. If the global fah_flag is true, then it converts the temperature to Fahrenheit. The logical flowchart is below.

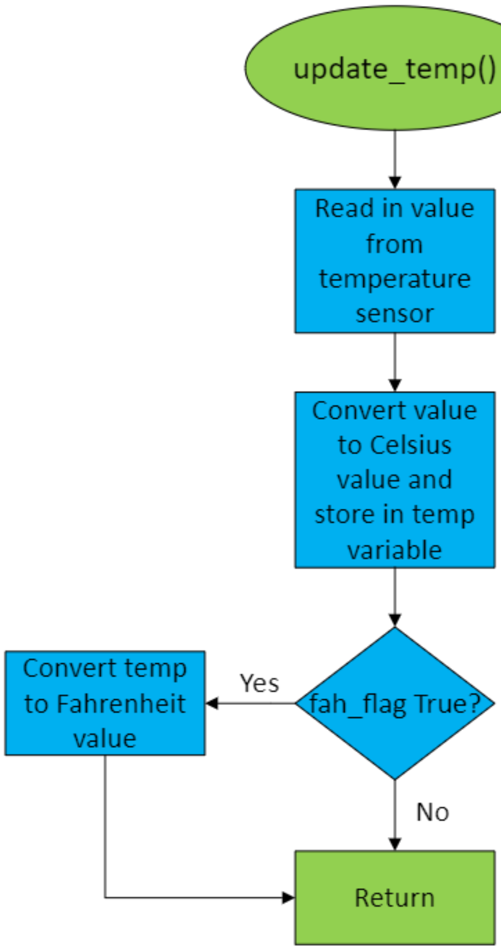
update_temp()
Read in value from temperature sensor
Convert value to Celsius value and store in temp variable
fah_flag True?
Yes
Convert temp to Fahrenheit value
No
Return

Figure 5: update_temp flowchart

Update_time

Update_time takes in no variables and returns nothing. It adds 1 to the global second variable. If second is greater than 59 then it is reset to 0 and 1 is added to minute. If minute is greater than 59 then it is reset to 0 and 1 is added to hour. If hour goes from 11 to 12 then change_noon is called. If hour is greater than 12 then it is reset to 1. The logical flowchart is below.

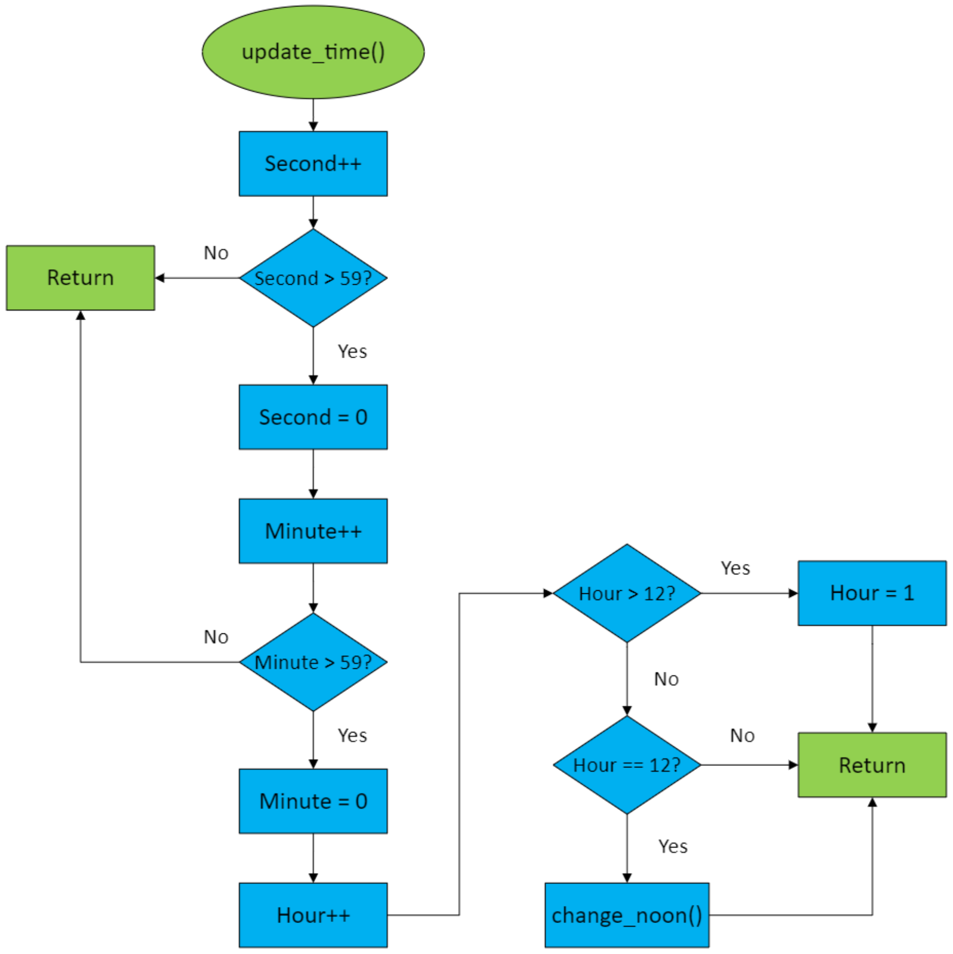

Figure 6: update_time flowchart

Change_noon

Change_noon takes in no variables and returns none. If noon_string is currently set to “AM” then it is set to “PM”. If noon_string is currently set to “PM” then it is set to “AM”. There is no flowchart for this function.

Main

The main function first instantiates all of the colPins to be PullDown pins and all of the rowPins to low. It then clears the screen, calls update_temp, then calls update_screen before entering the primary while loop. The loop first calls find_key to check if it should call set_time or to invert fah_flag. If ‘*’ is entered then set_time is called and fah_flag is inverted if ‘1’ is entered. The loop then calls update_time, update_temp, update_screen, then waits for 550ms to make each loop approximately one second. The loop then restarts.

Test Plan

The system was tested in stages. First, the LCD was wired based on the specification sheet and the provided example code was run to verify that the solder points were made correctly and that everything was wired correctly. Once the LCD was verified, the keypad was connected, and test code was written to output the pressed key to the LCD display. Once the keypad was verified, the temperature sensor was connected, and test code was written to verify the sensor worked and to determine how to convert the measured value to a measurement in degrees Celsius.

Once each component was verified and physically connected, code was written to implement the entire system. First was the clock. The update_time, update_screen, change_noon, and update_temp functions were implemented. A wait function was added to the main function to ensure that the clock incremented once a second. The find_key and set_time functions were then implemented to allow the user to set the time. The time set mode was then entered in the clock and on each screen, incorrect values were intentionally entered to ensure that the error detection was working correctly. The time was then set correctly to ensure that the time was able to be set. Once those were ensured, the time was then set to 11:59 AM and 11:59 PM to ensure that the clock was incrementing time correctly.

All test code is labeled in the **Code** section.

Conclusion

One of the advantages to the design approach is the use of global variables. There were many instances in which functions did not need anything passed to it, nor did they need to return anything (with the exception of find_key). This allowed for more consistent naming of variables and made much of the code easier to follow. It also allowed for a general purpose update_screen function that operated the same no matter what the time was. There were only 1 special case that had to be considered when displaying the time (when any number was less than 10).

Given the chance to re-write the code for the clock, timers and interrupts would have been used to ensure that the time was kept more accurately. The set_time function would also have been split into 2-3 functions. It works perfectly well as is, but it would be easier to follow if there was a function for setting the hour, minute, and AM/PM separately.

Code

LCD Test Code

```
#include "mbed.h"
#include "TextLCD.h"
#include <string>
#include <iostream>
using namespace std;


//TextLCD lcd(RS, E, D4, D5, D6, D7);
TextLCD lcd(PA_0, PA_1, PA_4, PB_0, PC_1, PB_5);

int main(){
   string ampm = "AM";
   int temp = 97;
   while(true){
       lcd.cls();
       lcd.printf("12:59 %s %iF\n", ampm, temp);
       wait(1.0);
   }

}
```

Keypad & LCD Test Code

```
#include "mbed.h"
#include "TextLCD.h"
#include <string>
#include <iostream>
using namespace std;

DigitalOut rowPins[4] = {PA_6, PA_7, PB_6, PC_7};
DigitalIn colPins[4] = {PA_9, PA_8, PB_10, PB_4};
```

```
//DigitalOut row1(PA_6, PullDown);
//rowPins[0] = row1;

// Instantiate keypad character vector
char keypad[4][4] = {
        {'1', '2', '3', 'A'},
        {'4', '5', '6', 'B'},
        {'7', '8', '9', 'C'},
        {'*', '0', '#', 'D'}
};

int low = 0;
int high = 1;

// function to determine, and return the pressed key. Returns null char (\0) if
none are pressed
char findKey() {
    // Loops through the row pins
    for (int row = 0; row < 4; row++) {
        // Sets the current pin to low (0), then rests for 100ms
        rowPins[row] = high;
        wait(0.01);

        // Loops through the collumn pins
        for (int col = 0; col < 4; col++) {
            // Checks if the current pin is high. If so, the button
            // corresponding to the current collumn and row is pressed
            if (colPins[col] == high) {
                // Reset the current row pin then rests for 100ms
                rowPins[row] = low;
                // Returns the appropriate character
                return keypad[row][col];
            }
            // Rests to ensure nothing is being overloaded
            //wait(0.1);
        }
        // Reset the current row pin then rests for 100ms
        rowPins[row] = low;
        wait(0.01);
    }
    // Returns null character
    return '\0';
}
```

```
//TextLCD lcd(RS, E, D4, D5, D6, D7);
TextLCD lcd(PA_0, PA_1, PA_4, PB_0, PC_1, PB_5);

int main(){
    //Instantiate all col pins to PullDown and all row pins to low
    for(int i = 0; i < 4; i++) {
        colPins[i].mode(PullDown);
        rowPins[i] = low;
    }
    //Char to read keypad input
   char input;
   while(true){
       //Read keypad input
       input = findKey();
       //Print input if any was measured
       if (input != '\0') {
           lcd.cls();
           lcd.printf("Pressed %c", input);
       }
       wait(1.0);
   }
}
```

Temp Sensor Test Code

```
#include "mbed.h"
#include "TextLCD.h"
#include <string>
#include <iostream>
using namespace std;

//TextLCD lcd(RS, E, D4, D5, D6, D7);
TextLCD lcd(PA_0, PA_1, PA_4, PB_0, PC_1, PB_5);

//Temperature sensor
AnalogIn therm(A5);

int main() {
    //Int and float to display and read the temperature
    int temp;
    float therm_input;
    while (true) {
        //Read in temperature sensor reading
        therm_input = therm.read();
        //Convert temperature reading to degrees Celsius
        temp = (therm_input * 148) + 2;
        //Clear LCD
        lcd.cls();
        //lcd.printf("Read: %.1f\n", therm_input);
        lcd.printf("Temp: %d C", temp);
        wait(1.0);
    }
}
```

Source Code

The following code is also included in the assignment submission.

```
#include "mbed.h"
#include "TextLCD.h"
#include <string>
#include <iostream>
using namespace std;

//Instantiate LCD object
//TextLCD lcd(RS, E, D4, D5, D6, D7);
TextLCD lcd(PA_0, PA_1, PA_4, PB_0, PC_1, PB_5);

//Instantiate Digital I/O pins for keypad
DigitalOut rowPins[] = {PA_6, PA_7, PB_6, PC_7};
DigitalIn colPins[] = {PA_9, PA_8, PB_10, PB_4};

//Instatiates Analog in object for temperature sensor
AnalogIn therm(A5);

//Instantiate keypad character matrix
char keypad[4][4] = {
        {'1', '2', '3', 'A'},
        {'4', '5', '6', 'B'},
        {'7', '8', '9', 'C'},
        {'*', '0', '#', 'D'}
};
```

```
//Instantiates string to track AM/PM
string noon_string = "AM";

//Instantiates integers to track time and temperature
int second = 0, minute = 0, hour = 12, temp;

//Ints for setting and checking pin values
int low = 0, high = 1;

//Boolean to determine F/C
bool fah_flag = true;

void update_temp() {
    //Read temperature sensor reading
    float therm_input = therm.read();
    //Convert reading to degrees C
    temp = (therm_input * 148) + 2;
    //If fahrenheit flag is true, convert to fahrenheit
    if (fah_flag) {
        temp = temp * (9/5) + 32;
    }
}

void change_noon() {
    //If noon = AM, make it PM
    if (noon_string == "AM") {
        noon_string = "PM";
    }
    //If noon = PM, make it AM
    else {
        noon_string = "AM";
    }
}

void update_time() {
    //Add 1 to seconds
    second++;
    //If seconds reaches 60, reset second to 0 and add 1 to minute
    if (second > 59) {
        second = 0;
        minute++;

        //If minute reaches 60, reset minute to 0 and add 1 to hour
        if (minute > 59) {
            minute = 0;
            hour++;

            //If hour increments from 11 to 12, flip noon
            if (hour == 12 && minute == 0) {
                change_noon();
            }

            //If hour passes 12, reset to 1
            if (hour > 12) {
                hour = 1;
            }
        }
    }
}

char find_key() {
    bool pressed_flag = false;
    //Loops through the row pins
    for (int row = 0; row < 4; row++) {
        //Sets the current pin to high (1), then rests for 100ms
        rowPins[row] = high;
        wait(0.01);

        //Loops through the collumn pins
        for (int col = 0; col < 4; col++) {
            //Checks if the current pin is high. If so, the button
            //corresponding to the current collumn and row is pressed
            if (colPins[col] == high) {
                pressed_flag = true;
                //Continually loops until key is released
                //Effectively suspends execution until key is released
                while (pressed_flag) {
                    if (colPins[col] == low) {
                        pressed_flag = false;
                    }
                }

                //Reset the current row pin
                rowPins[row] = low;
                //Returns the appropriate character
                return keypad[row][col];
            }
        }
        //Reset the current row pin then rests for 100ms
        rowPins[row] = low;
        wait(0.1);
    }
    //Returns null character
    return '\0';
}

void set_time() {
    //**************************************************
    //Set hour
    //clear LCD
    lcd.cls();

    //Flag for while loop
    //Char for input
    //Int for setting hour var from input
    bool input_flag = true;
    char char_input = '\0';
    int input = 0;

    //Print to LCD
    lcd.printf("Hour: ");

    //WHile loop for setting hour
    while (input_flag) {
        //Aquire input from keypad
        char_input = find_key();

        //User presses '*' (Enter)
        if (char_input == '*') {
            //Checks for valid input and sets hour and sets flag to false to exit
while loop
            if (input <= 12 && input >= 1) {
                hour = input;
                input_flag = false;
            }
            //If input is invalid, prints "ERROR" and resets everything
            else {
                lcd.cls();
                lcd.printf("ERROR");
                wait(1);
                input = 0;
                lcd.cls();
                lcd.printf("Hour: ");
            }
        }
        //If user didn't press '*' prints user input and adds it to the int input
        else if (char_input != '\0') {
            lcd.printf("%c",char_input);
            input = (input * 10) + (char_input - '0');
        }
    }

    //***************************************************
    //Set minute
    //clear LCD
    lcd.cls();

    //Flag for while loop, char for input, int for setting minute var from input
    input_flag = true;
    char_input = '\0';
    input = 0;

    //Print to LCD
    lcd.printf("Minute: ");

    //WHile loop for setting minute
    while (input_flag) {
        //Aquire input from keypad
        char_input = find_key();

        //User presses '*' (Enter)
        if (char_input == '*') {
            //Checks for valid input and sets minute and sets flag to false to
exit while loop
            if (input <= 59 && input >= 0) {
                minute = input;
                input_flag = false;
            }
            //If input is invalid, prints "ERROR" and resets everything
            else {
                lcd.cls();
                lcd.printf("ERROR");
                wait(1);
                input = 0;
                lcd.cls();
                lcd.printf("minute: ");
            }
        }
        //If user didn't press '*' prints user input and adds it to the int input
        else if (char_input != '\0') {
            lcd.printf("%c",char_input);
            input = (input * 10) + (char_input - '0');
        }
    }


    //**************************
    //Set AM/PM

    //Clear LCD
    lcd.cls();

    //Flag for while loop, char for input and int for choice
    input_flag = true;
    char_input = '\0';
    input = 0;

    //Prints options to LCD
    lcd.printf("AM(1) or PM(2)\n");

    //While loop for setting AM/PM
    while (input_flag) {
        //Aquire input from keypad
        char_input = find_key();

        //If user presses '*' set noon or throw error
        if (char_input == '*') {
            //If input is 1 set noon to AM
            if (input == 1) {
                noon_string = "AM";
                input_flag = false;
            }
            //If input is 2 set noon to PM
            else if (input == 2) {
                noon_string = "PM";
                input_flag = false;
            }
            //If input is invalid, prints "ERROR" and resets everything
            else {
                lcd.cls();
                lcd.printf("ERROR");
                wait(1);
                input = 0;
                lcd.cls();
                lcd.printf("AM(1) or PM(2)\n");
            }
        }
        //If user didn't press '*', store input and print it
        else if (char_input != '\0') {
            input = char_input - '0';
            lcd.printf("%d", input);
        }

    }

    //*****************************
    //Set second to 0
    second = 0;
}

void update_screen() {
    //clear LCD
    lcd.cls();

    //Print the current hour to the lcd (prints an extra zero if less than 10)
    if (hour < 10) {
        lcd.printf("0");
    }
    lcd.printf("%d:", hour);

    //Print the current minute to the lcd (prints an extra zero if less than 10)
    if (minute < 10) {
        lcd.printf("0");
    }
    lcd.printf("%d:", minute);

    //Print the current second to the lcd (prints an extra zero if less than 10)
    if (second < 10) {
        lcd.printf("0");
    }
    lcd.printf("%d ", second);

    //Prints AM/PM
    lcd.printf("%s ", noon_string.c_str());

    //Print the current temperature to the lcd (prints an extra zero if less than
10)
    if (temp < 10) {
        lcd.printf("0");
    }
    lcd.printf("%d ", temp);
    if (fah_flag) {
        lcd.printf("F");
    }
    else {
        lcd.printf("C");
    }
}

int main() {
    //Set the input pins to PullDown and output pins to 0
    for(int i = 0; i < 4; i++) {
        colPins[i].mode(PullDown);
        rowPins[i] = low;
    }

    char input;
    //clear LCD
    lcd.cls();

    //Get the current temperature
    update_temp();

    //Update Screen
    update_screen();

    while(true) {
        //Check if user is pressing a key
        input = find_key();
        //Enter set time mode if '*' is pressed
        if (input == '*') {
            set_time();
        }
        //Invert fahrenheit flag if '#' is pressed
        if (input == '1') {
            fah_flag = !fah_flag;
        }
        //Update time
        update_time();
        //Update temp
        update_temp();
        //Update screen
        update_screen();
        //Wait to make each loop more approximate to 1 second
        wait(0.55);
        }
}
```